ALIBABA CLOUD

Alibaba Cloud

Apsara Stack Enterprise

ApsaraDB for OceanBase

User Guide

Product Version: V3.12.0

Document Version: 20220928

Alibaba Cloud

Legal disclaimer

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.What is ApsaraDB for OceanBase?

ApsaraDB for OceanBase is a financial-grade, distributed relational database service that features high performance, high availability, and high scalability. It supports active geo-redundancy and geo-disaster recovery to ensure high availability. It also supports high scalability to meet the increasing business requirements.

These features of ApsaraDB for OceanBase help you handle the challenges that are brought by rapid business growth. ApsaraDB for OceanBase also provides scalable and low latency database services in high throughput scenarios. This ensures improved user experience. For example, during the Double 11 in 2017, ApsaraDB for OceanBase handled all the transactions and payment requests. The maximum number of transactions that were made on Alipay reached 256,000 per second. The maximum number of processed requests per second reached 42 million. ApsaraDB for OceanBase accelerates the development of Internet finance.

The distributed engine of ApsaraDB for OceanBase uses the Paxos protocol and maintains multiple replicas. For the Paxos protocol, transactions can be committed only after they are approved by a majority of the acceptors. The Paxos protocol and multiple-replica design allow ApsaraDB for OceanBase to offer high availability and disaster recovery capabilities. ApsaraDB for OceanBase can help you achieve zero downtime. ApsaraDB for OceanBase supports high-availability architectures, such as active geo-redundancy and geo-disaster recovery. You can deploy the ApsaraDB for OceanBase service across data centers, regions, or continents. ApsaraDB for OceanBase provides financial-grade availability features and ensures strong consistency of transactions.

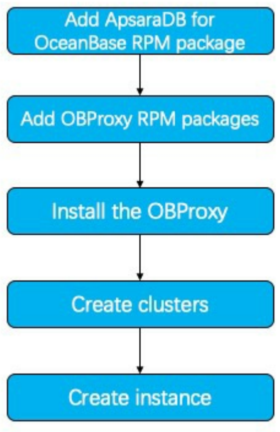
ApsaraDB for OceanBase is similar to an in-memory database and adopts a read/write splitting architecture. To ensure high efficiency for the storage engine, ApsaraDB for OceanBase stores baseline data in solid-state drives (SSDs) and stores incremental data in memory. This ensures that ApsaraDB for OceanBase offers high performance services. ApsaraDB for OceanBase is a cloud-based database service that supports multi-tenant data isolation. Each cluster of ApsaraDB for OceanBase can provide services for multiple tenants. The tenants are isolated so that they are not affected by each other.

ApsaraDB for OceanBase is compatible with most of the MySQL 5.6 features. This allows you to migrate MySQL-based services to ApsaraDB for OceanBase based on zero or small code modifications. This improves the efficiency of developing applications and migrating services. In ApsaraDB for OceanBase, you can create partitioned tables and use subpartitions. This serves as an alternative to MySQL sharding solutions. The ApsaraDB for OceanBase console provides an easy way for you to manage complex databases. For example, you can use the console to upgrade or downgrade instances, view performance data, and view optimization suggestions.

2.Quick start

2.1. Overview

This topic provides a quick start for ApsaraDB for OceanBase. This topic describes how to log on to the ApsaraDB for OceanBase console, add RPM packages of ApsaraDB for OceanBase and the OBProxy, install the OBProxy, create clusters, and create instances. After you perform these steps, you can use the created instances.



The following section describes each step in the flowchart:

1. Add OBProxy RPM packages

   In this step, add the RPM package of the OBProxy for ApsaraDB for OceanBase.
2. Install the OBProxy

   The OBProxy is a reverse proxy server that is specific to ApsaraDB for OceanBase. ApsaraDB for OceanBase provides distributed relational database services. You can use the OBProxy to prevent transient connections and ensure that backend exceptions and operations such as breakdowns, upgrades, and network jitters are transparent to users. The OBProxy is also compatible with the MySQL protocol. The OBProxy supports strong checks, hot upgrades, and multiple clusters. In terms of frontend user requests, the OBProxy provides routing and forwarding services that feature high performance and high accuracy. In terms of backend server services, the OBProxy provides disaster recovery solutions that feature high availability and high scalability.
3. Create clusters

   In this step, create an ApsaraDB for OceanBase cluster and configure the relevant parameters.
4. Create instances

   In this step, create an ApsaraDB for OceanBase instance and configure the relevant parameters.

2.2. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase

This topic describes how to use the Apsara Stack O&M system to log on to the Apsara Stack Operations console for ApsaraDB for OceanBase. The Google Chrome browser is used as an example in this topic.

Prerequisites

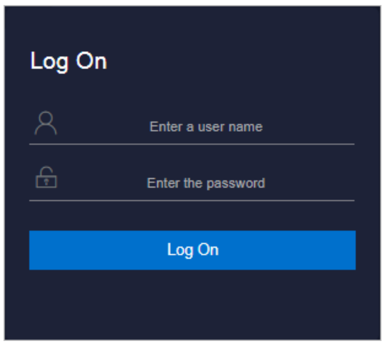
- The URL of the ASO console, and the username and password used for logging on to the console are obtained from the deployment personnel or an administrator.

  The URL of the ASO console is in the following format: *region-id*.aso.*intranet-domain-id*.com.
- A browser is available. We recommend that you use Google Chrome.
- The endpoint of the Apsara Uni-manager Operations Console and the username and password used to log on to the console are obtained from the deployment personnel or an administrator.

  The endpoint of the Apsara Uni-manager Operations Console is in the following format: *region-id*.ops.console.*intranet-domain-id*.
- A browser is available. We recommend that you use Google Chrome.

Procedure

1. In the address bar, enter the URL *region-id*.aso.*intranet-domain-id*.com and press the Enter key.

   Note You can select a language from the drop-down list in the upper-right corner of the page.

2. Enter your username and password.

   Note Obtain the username and password for logging on to the ASO console from the deployment personnel or an administrator.

   When you log on to the ASO console for the first time, you must change the password of your username as prompted.

   To enhance security, a password must meet the following requirements:

   - It must contain uppercase and lowercase letters.
   - It must contain digits.
   - It must contain special characters such as exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%).
   - It must be 10 to 20 characters in length.

3. Click **Log On** to go to the **ASO** console.
4. In the left-side navigation pane, click **Products**. Then, choose **Product List > Database Services** and click **OceanBase Cloud Platform**. You are directed to the Apsara Stack Operations console for ApsaraDB for OceanBase.

2.3. Log on to the ApsaraDB for OceanBase console

This topic describes how to use the console to log on to the ApsaraDB for OceanBase console. The Google Chrome browser is used as an example in this topic.

Prerequisites

- A browser is available. We recommend that you use Google Chrome.

Context

If you use the console to log on to the ApsaraDB for OceanBase console, you can only view monitoring information and manage instances. If you need to perform operations and maintenance (O&M) tasks on clusters, you must log on to the Apsara Stack Operations console. For more information, see Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.

2.4. Add ApsaraDB for OceanBase RPM packages

In the Apsara Stack Operations console for ApsaraDB for OceanBase, you can add an RPM Package Manager (RPM) package of the current ApsaraDB for OceanBase version.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **O&M**. In the list that appears, select **RPM Package Management**.
3. On the RPM Package Management page, click the **Database** tab. On the tab, click **Upload RPM Package**.
4. In the Upload RPM Package dialog box, select observer from the File Type drop-down list and click **Upload**. Then, select the RPM package that you want to upload from the local directory.
5. Click **OK**.

2.5. Add OBProxy RPM packages

In the Apsara Stack Operations console for ApsaraDB for OceanBase, you can add an RPM package of the OBProxy that matches the current ApsaraDB for OceanBase version.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **O&M**. In the list that appears, select **RPM Package Management**.
3. On the RPM Package Management page, click the **Database** tab. On the tab, click **Upload RPM Package**.
4. In the dialog box that appears, select obproxy from the File Type drop-down list, and click **Upload**. Then, select the RPM package that you want to upload.
5. Click **OK**.

2.6. Install the OBProxy

After you add an OBProxy RPM package, you can install the OBProxy.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **O&M**. In the list that appears, select **OBProxy**.
3. On the **Servers** tab of the **OBProxy** page, find the IP address of the OBProxy server to be installed. In the Actions column for the IP address, click **Install**.
4. In the dialog box that appears, set OBProxy Name to the name of the OBProxy based on your project name. From the OBProxy Version drop-down list, select an OBProxy version, such as obproxy-1.5.0-1410335.el7.x86_64.rpm. You can select the current time for Start Time.
5. Click **OK**.

2.7. Create clusters

Before you use ApsaraDB for OceanBase, create clusters in the Apsara Stack Operations console for ApsaraDB for OceanBase.

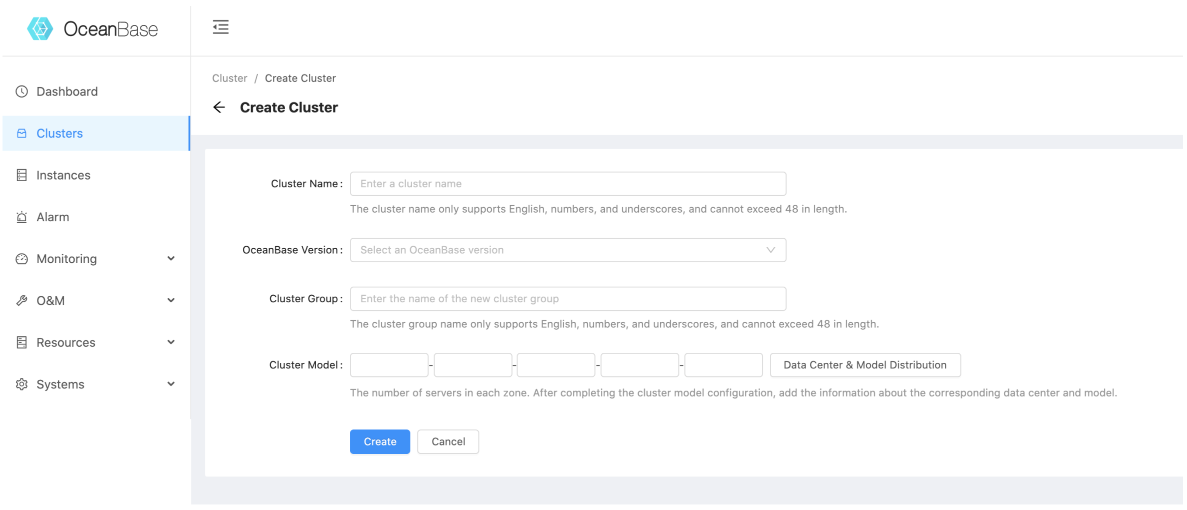
Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Clusters**.
3. On the page that appears, click **Create Cluster**. In the **Create Cluster** dialog box, configure the parameters as prompted.

   For more information about the parameters, see Parameters for creating a cluster.

Parameters for creating a cluster

Parameter	Description
Cluster Name	The name of the ApsaraDB for OceanBase cluster.
OceanBase Version	The version of the ApsaraDB for OceanBase cluster. We recommend that you select the latest version.
Cluster Group	The group to which the cluster belongs. You can enter the name of the cluster group.
Cluster Model	The cluster model. If the minimal specifications are used for ApsaraDB for OceanBase, the cluster model is 1-1-1. You can set this parameter to 2-2-2 or 3-3-3 based on your cluster model. If five replicas are deployed, set this parameter to 2-2-2-2-2. To obtain the cluster model, contact the owner of the ApsaraDB for OceanBase project.
Data Center & Model Distribution	The data center and the machine model. Select the data center and the machine model based on the specified cluster model.



Cluster Name: Enter a cluster name
The cluster name only supports English, numbers, and underscores, and cannot exceed 48 in length.

OceanBase Version: Select an OceanBase version

Cluster Group: Enter the name of the new cluster group
The cluster group name only supports English, numbers, and underscores, and cannot exceed 48 in length.

Cluster Model: - - - - Data Center & Model Distribution
The number of servers in each zone. After completing the cluster model configuration, add the information about the corresponding data center and model.

Create Cancel

4. After you specify the preceding parameters, click **Create**.

2.8. Create instances

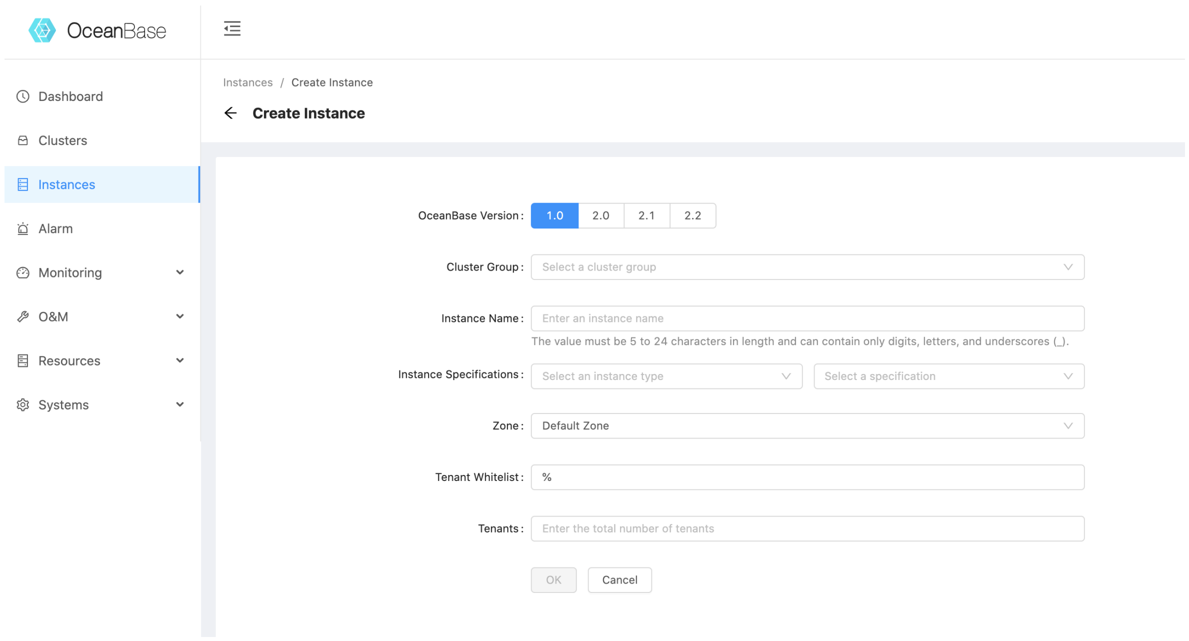
After you create ApsaraDB for OceanBase clusters, you can create OceanBase instances in the ApsaraDB for OceanBase console.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Instances**.
3. On the page that appears, click **Create Instance**. In the **Create Instance** dialog box, configure the parameters as prompted. describes the relevant parameters.

Parameters for creating an instance

Parameter	Description
OceanBase Version	The ApsaraDB for OceanBase version. Specify this parameter based on your business requirements.
Cluster Group	The cluster group where the instance is created.
Instance Name	The name of the instance. Specify this parameter based on your business requirements.
Zone	The zone where the instance is deployed. From the Zone drop-down list, select Default Zone.
Instance Specifications	The type of the instance. Specify this parameter based on your business requirements.
Tenant Whitelist	The tenant whitelist of the instance. Separate IP addresses or CIDR blocks with commas (,). The parameter value % indicates that all the tenants of the instance are added to the whitelist. We recommend that you use the default value %.
Tenants	The total number of tenants that are bound to the instance.



Create Instance

OceanBase Version: 1.0 | 2.0 | 2.1 | 2.2

Cluster Group: Select a cluster group

Instance Name: Enter an instance name

The value must be 5 to 24 characters in length and can contain only digits, letters, and underscores (_).

Instance Specifications: Select an instance type | Select a specification

Zone: Default Zone

Tenant Whitelist: %

Tenants: Enter the total number of tenants

OK | Cancel

4. Click **OK**.

3.Clusters

3.1. Overview

Before you use ApsaraDB for OceanBase, create ApsaraDB for OceanBase clusters in the ApsaraDB for OceanBase console. After you create clusters, you can view the basic information and the operation logs of the clusters. You can also restart, upgrade, scale out, and delete the clusters, and perform other operations on the clusters.

3.2. Scale out clusters online

In the ApsaraDB for OceanBase console, you can scale out ApsaraDB for OceanBase clusters based on your business requirements.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Clusters**. The Clusters page appears.
3. In the Actions column for the cluster that you want to scale out, choose **More > Scale Out Cluster**.
4. In the Scale out Cluster dialog box, specify New Servers. In the Model column, select the model of the servers that you want to add. Then, select the current date and time from the Start Time date and time picker and click **OK**.

3.3. Restart clusters

You can restart clusters in the ApsaraDB for OceanBase console.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Clusters**. The Clusters page appears.
3. In the Actions column for the cluster that you want to restart, choose **Routine Actions > Restart Cluster**.
4. In the Restart Cluster dialog box, select the zones of the cluster that you want to restart and click the Start Time field. In the date and time picker that appears, click Now and then click **OK**.

3.4. Delete clusters

You can delete clusters in the ApsaraDB for OceanBase console.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Clusters**. The Clusters page appears.
3. In the Actions column for the cluster that you want to remove, choose **More > Delete Cluster**.
4. In the Delete Cluster dialog box, click the Start Time field. In the date and time picker that appears, click Now and then click **OK**.

3.5. Upgrade clusters

In the ApsaraDB for OceanBase console, you can perform online upgrades to upgrade ApsaraDB for OceanBase clusters to the specified versions.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Clusters**. The Clusters page appears.
3. In the Actions column for the cluster that you want to upgrade, choose **Routine Operations > Cluster Upgrade**.
4. In the Cluster Upgrade dialog box, select all the rows. Each row corresponds to one zone of the cluster. From the Upgrade Version drop-down list, select the cluster version that you want to upgrade. Then, click the Start Time field. In the date and time picker that appears, click Now and then click **OK**.

3.6. View the monitoring information about clusters

In the ApsaraDB for OceanBase console, you can view the monitoring information about the major resources of each cluster, including CPU usage, memory usage, and disk usage.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Monitoring** and then click **OceanBase Data Trends**.
3. On the **OceanBase Data Trends** page, view the performance information. By default, the page shows the performance information about all the clusters. To view the performance information based on different dimensions, specify **Cluster Group**, **Clusters**, **Time**, **Zone**, and **Server** parameters based on your business requirements. Then, click **Search**.

3.7. View the real-time monitoring information about clusters

In the ApsaraDB for OceanBase console, you can view the real-time monitoring information about clusters, including queries per second (QPS) and transactions per second (TPS).

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Monitoring** and then click **Real-time Monitoring**. The Real-time Monitoring page appears.
3. On the **Real-time Monitoring** page, view the real-time monitoring information. By default, the page shows the performance information of all the clusters. To view the real-time performance information about a specific cluster group, click the name of the cluster group and then click Search. The **Real-time Performance Data** section appears. In this section, you can view the performance information about the cluster group.

3.8. View performance metrics

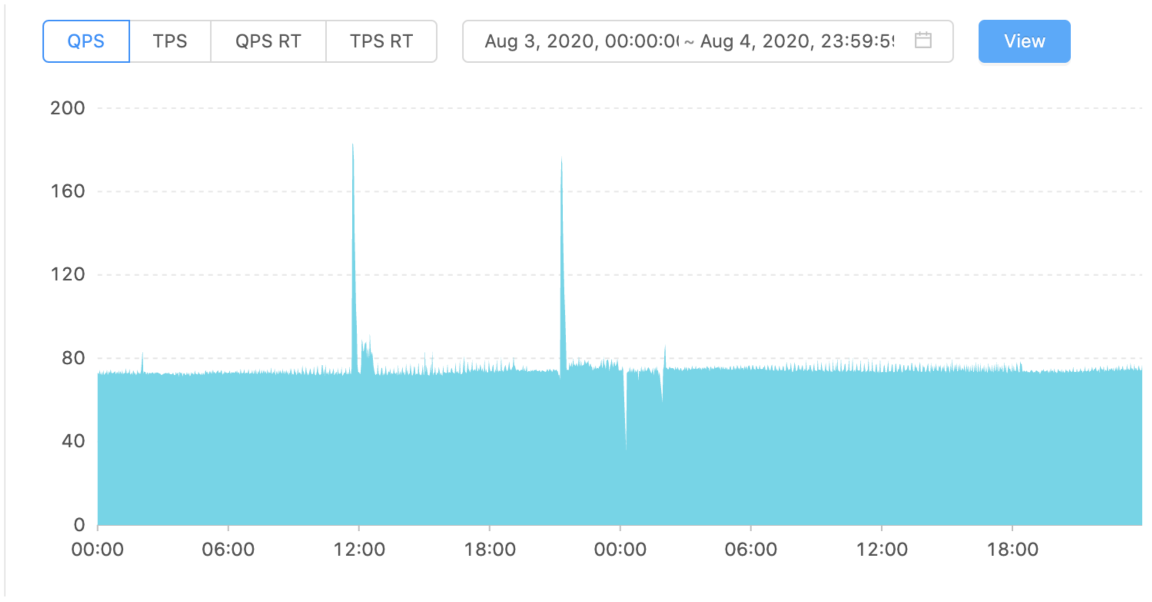
You can view a wide range of performance metrics of clusters. For example, you can view the information about QPS, TPS, QPS response time (RT), TPS RT, top five clusters by major freeze time, O&M tasks, inspection data, and server usage.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.

   After you log on to the ApsaraDB for OceanBase console, you can view the information about performance metrics on the **Dashboard** page. For example, you can view the information about QPS, TPS, QPS RT, TPS RT, top five clusters by major freeze time, O&M tasks, inspection data, and server usage. You can also view the performance trend of each metric by performing the following steps:
2. In the left-side navigation pane, click **Monitoring** and then click **OceanBase Data Trends**. The OceanBase Data Trends page appears.
3. On the **OceanBase Data Trends** page, click **QPS**, **TPS**, **QPS RT** or **TPS RT**. On the corresponding tab for the performance metric, specify the start time and end time for the search and click **View**.

   Note In the lower part of the date and time picker for each performance metric, you can also select a time duration to specify the start time and the end time for the search. By default, the performance data in the last *X* period is displayed and the end time of the period is the current time. *X* represents the time duration that you select.


QPS
TPS
QPS RT
TPS RT
Aug 3, 2020, 00:00:0(~ Aug 4, 2020, 23:59:5!
View
200
160
120
80
40
0
00:00
06:00
12:00
18:00
00:00
06:00
12:00
18:00

4.Instances

4.1. Overview

After you create instances, you can manage the instances. For example, you can view the basic information about the instances, reset the instance passwords, change the instance names, and start or stop the instances. You can also view the performance metrics, the change history, and the performance overview of each instance.

4.2. Change instance passwords

After you create instances, the default passwords for the instances are unknown. Therefore, you must log on to the console as an administrator and change the passwords.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Instances**. The Instances page appears.
3. In the Instance Name column, click the name of the instance for which you want to change the password. The **Basic Information** tab appears.
4. In the lower-left corner of the tab, click **Reset Password**.
5. In the Reset Password dialog box, enter and confirm the new password and click **OK**.

4.3. View instance details

You can view the basic information about instances, such as connection strings, instance information, and maintenance windows.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Instances**. The Instances page appears.
3. In the Instance Name column, click the name of the instance whose basic information you want to view. On the **Basic Information** tab, view the basic information about the instance.

4.4. Change instance specifications

You can change instance specifications based on your business requirements.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Instances**. The Instances page appears.
3. In the Instance Name column, click the name of the instance whose specifications that you want to change. The **Basic Information** page appears.
4. In the lower-left corner of the page that appears, click **Modify Instance Specifications**.
5. In the Modify Instance Specifications dialog box, change the instance specifications.
6. Click **OK**.

4.5. Delete instances

To ensure efficient usage of resources, you can delete the instances that are no longer required.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Instances**. The Instances page appears.
3. In the Instance Name column, click the name of the instance that you want to delete. The **Basic Information** tab appears.
4. In the lower-left corner of the page that appears, click **Delete**.
5. In the message that appears, click **OK**.

   Note The instance cannot be recovered after the instance is deleted. Proceed with caution.

4.6. View the performance metrics of instances

You can view the major performance metrics of running instances, such as QPS and TPS.

Procedure

1. Log on to the Apsara Stack Operations console for ApsaraDB for OceanBase.
2. In the left-side navigation pane, click **Instances**. The Instances page appears.
3. In the Instance Name column, click the name of the instance whose performance metrics you want to view. The **Basic Information** tab appears.
4. Click the **Performance Metrics** tab. On the tab, view the performance information about the instance.

5.SQL syntax reference

5.1. Introduction to ApsaraDB for OceanBase SQL

ApsaraDB for OceanBase supports Structured Query Language (SQL). SQL is a computer language that you can use to organize, manage, and retrieve data in databases.

ApsaraDB for OceanBase is fully compatible with the MySQL protocol. Therefore, users of ApsaraDB for OceanBase SQL can connect to ApsaraDB for OceanBase by using MySQL clients, such as MySQL command-line, Java, and C clients.

5.2. Overview of ApsaraDB for OceanBase SQL

5.2.1. Identifiers

Use identifiers

In ApsaraDB for OceanBase, identifiers are the names of schema objects in SQL statements. The schema objects include tenants, databases, tables, views, indexes, table groups, columns, and aliases.

Before you use ApsaraDB for OceanBase identifiers, pay attention to the following considerations:

describes the maximum length and the supported characters for each identifier.

ApsaraDB for OceanBase identifiers

Identifier	Maximum length (bytes)	Supported character
Username	16	A username can contain letters, digits, and underscores (_). It must start with a letter or an underscore (_). You cannot use ApsaraDB for OceanBase keywords when you specify the name.
Tenant name	64	A tenant name can contain letters, digits and underscores (_). It must start with a letter or an underscore (_). You cannot use ApsaraDB for OceanBase keywords when you specify the name.
Database name	64	A database name can contain letters, digits, underscores (_), and dollar signs (`$`).
Table group name	64	A table group name can contain letters, digits and underscores (_). It must start with a letter or an underscore (_). You cannot use ApsaraDB for OceanBase keywords when you specify the name.
Table name	64	A table name can contain letters, digits, underscores (_), and dollar signs (`$`).
Column name	64	A column name can contain letters, digits, underscores (_), and dollar signs (`$`).
Index name	64	An index name can contain letters, digits, underscores (_), and dollar signs (`$`).
Alias	255	An alias can contain letters, digits, underscores (_), and dollar signs (`$`).
Variable name	64	A variable name can contain alphanumericals, periods (.), underscores (_), and dollar signs (`$`).

Note

- In addition to the limits in the preceding table, you must meet some other identifier limits. For example, each identifier cannot contain the ASCII control character 0 or the eight-bit byte that represents 255.
- The names of databases, tables, or columns cannot end with spaces.
- We recommend that you do not use quotation marks in identifiers.

You can determine whether to use backticks (`) to enclose each identifier based on the actual scenarios. If identifiers are reserved keywords or contain special characters, you must use backticks (`) to enclose the identifiers. In ApsaraDB for OceanBase, you can use only backticks (`) to enclose identifiers.

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

Reference identifiers

ApsaraDB for OceanBase allows you to use a name that consists of one or more identifiers. If a name consists of multiple identifiers, separate the identifiers with periods (.).

lists the columns that you can reference in ApsaraDB for OceanBase.

Referable columns and descriptions

Column reference	Description
col_name	The col_name column in the table. The query retrieves data from the table.
tbl_name.col_name	The col_name column in the tbl_name table of the default database.
db_name.tbl_name.col_name	The col_name column in the tbl_name table of the db_name database.

If the identifiers that compose a name must be enclosed in backticks (`), enclose each identifier instead of the entire name. For example, `'my-tables'.'my-column'` is valid and `'my-tables.my-column'` is invalid.

If columns can be uniquely identified by column names, you do not need to specify the `tbl_name` prefix or the `db_name.tbl_name` prefix for the columns in statements. If columns cannot be uniquely identified by column names, you must specify one of the two prefixes. For example, the t1 table contains one c column and the t2 table also contains one c column. You want to execute a single SELECT statement to retrieve the c column in the t1 and t2 tables. In this scenario, the c column cannot be uniquely identified by the column name because this column exists in both t1 and t2 tables. You must specify prefixes for the two c columns to distinguish the c column in the t1 table from the c column in the t2 table: t1.c and t2.c. Similarly, you may need to execute a single statement to retrieve columns from the t table in the db1 database and the t table in the db2 database. In this scenario, you must use `db1.t.col_name` and `db2.t.col_name` to specify the databases that store the t tables.

In qualified names, each word that follows a period must be an identifier. Therefore, the word does not need to be enclosed in backticks (`). This rule applies even if the word is a reserved keyword.

The `.tbl_name` syntax specifies the `tbl_name` table in the current database. This syntax is compatible with the Open Database Connectivity (ODBC) API because table names are also prefixed with periods (`.`) in some ODBC programs.

Identifier case sensitivity

In ApsaraDB for OceanBase, identifiers are not case-sensitive. To ensure compatibility with MySQL, ApsaraDB for OceanBase introduces the `lower_case_tables_name` system parameter that is used in MySQL. This system parameter specifies whether table names and database names that function as identifiers are case-sensitive.

describes the values of the lower_case_tables_name parameters.

Description of lower_case_tables_name values

Value	Description
0	Table names and database names are case-sensitive.
1	Table names and database names are not case-sensitive.

5.2.2. Supported SQL statements

ApsaraDB for OceanBase supports the following SQL statements:

- Data definition language (DDL) statements

  CREATE DATABASE, ALTER DATABASE, DROP DATABASE, CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, ALTER VIEW, TRUNCATE TABLE, and RENAME TABLE
- Data manipulation language (DML) statements

  INSERT, REPLACE, DELETE, UPDATE, and SELECT statements and clauses
- Transaction statements

  START TRANSACTION, COMMIT, and ROLLBACK
- Prepared statements

  PREPARE, SET, EXECUTE, DEALLOCATE, DROP PREPARE, and DROP PREPARE
- Database administration statements

  CREATE RESOURCE UNIT, DROP RESOURCE UNIT, CREATE RESOURCE POOL, ALTER RESOURCE POOL, DROP RESOURCE POOL, CREATE TENANT, ALTER TENANT, DROP TENANT, CREATE TABLEGROUP, DROP TABLEGROUP, ALTER TABLEGROUP, CREATE USER, DROP USER, RENAME USER, ALTER USER, GRANTS, REVOKE, SET, SET PASSWORD, SET GLOBAL, ALTER SYSTEM, GRANT, and REVOKE
- Useful SQL statements

  SHOW, KILL, USE, DESCRIBE, EXPLAIN, HINT, HELP, and other SQL statements

5.2.3. SQL limits

This topic describes the following limits of SQL statements in ApsaraDB for OceanBase:

- ApsaraDB for OceanBase does not support custom data types or custom functions.
- ApsaraDB for OceanBase does not support updatable views, stored procedures, triggers, or cursors.
- ApsaraDB for OceanBase does not support temporary tables.
- ApsaraDB for OceanBase does not support compound statements, such as BEGIN...END, LOOP...END LOOP, REPEAT...UNTIL...END REPEAT, and WHILE...DO...END WHILE.
- ApsaraDB for OceanBase does not support flow control statements, such as IF and WHILE.
- The INSERT or REPLACE statements that contain SELECT clauses cannot be executed to modify data. The DELETE statements cannot be executed to delete data from multiple tables. The UPDATE statements cannot be executed to update data that is stored in multiple tables.
- The following list describes the quantity and maximum length limits:
  - When a table or an index is created, the maximum length of the primary key is 16 KB and the maximum length of a single row is 1.5 MB.
  - A single column of the VARCHAR data type can store a maximum of 262,143 bytes. The maximum length for the column of this type is 256 KB.
  - You can create a maximum of 64 primary keys.
  - You can store a maximum of 512 columns in a single table.
  - You can create a maximum of 128 indexes for a single table.
- The `SELECT… FOR UPDATE` statement supports only single-table queries.
- In ApsaraDB for OceanBase, you can execute the TRUNCATE TABLE statement on only a table that has one partition.

5.3. Partitions

5.3.1. Overview

ApsaraDB for OceanBase is a distributed database. If you need to distribute data in a table to multiple servers, you must create the table as a partitioned table. ApsaraDB for OceanBase automatically distributes data across ApsaraDB for OceanBase servers based on your partitions. You can also create non-partitioned tables. The data in each non-partitioned table is stored on only one ApsaraDB for OceanBase server.

The following list describes the characteristics of ApsaraDB for OceanBase partitioned tables:

- ApsaraDB for OceanBase supports `hash partitioning` , `key partitioning` and `range partitioning` .
- Tables are partitioned based on the partition key fields that you specify in the `CREATE TABLE` statements.
- The number of partitions in each partitioned table is specified in the `CREATE TABLE` statement.
- You can create non-partitioned tables. For example, you can create metadata tables that store only a few rows of service data as non-partitioned tables.
- In ApsaraDB for OceanBase, table data is distributed based on partitions. Therefore, to ensure high performance, the WHERE clauses in INSERT, REPLACE, SELECT, UPDATE, and DELETE statements must contain `PARTITION(partition_list)` . If this field is not contained, the system reports an error.

- In ApsaraDB for OceanBase, you can create a maximum of 8,192 partitions in a single partitioned table.

Introduction to partitions

In ApsaraDB for OceanBase, you can create partitions for each table based on the specified rules. The partitions can be distributed across ApsaraDB for OceanBase servers.

Partitioning functions specify the rules that you use to distribute data across partitions. Partitioning functions must be system functions.

Partitioning functions can be modulus functions, internal hash functions, or linear hash functions. To simplify the partitioning process, you can also use partitioning functions to match data against a range of continuous numeric values or a list of numeric values.

Partitioning functions are selected based on the partitioning types that you use. The partitioning functions use the values of the expressions that you provide as arguments. The expressions can represent the column values of the INT type. The expressions can also be the functions that are applied on one or more column values and return integers. The values of the expressions are passed to the partitioning functions. Then, the partitioning functions return sequence numbers that specify the partitions where the specific rows are stored. The specific rows are distinguished by the expression values.

You cannot use the partitioning functions to represent constants or random numbers.

The partitioning functions can use valid SQL expressions that return positive values. The positive values must be smaller than the maximum allowed positive integer that is specified by MAXVALUE.

Partitioning must be implemented on both data and indexes of each partitioned table. This means that you must create partitions for both data and indexes for each entire partitioned table. You cannot create partitions for only partial data of each partitioned table.

Partitioning benefits

- Partitioning offers an easy way for you to delete and add data. In most cases, you can delete the data that does not need to be stored by deleting the partitions that store the data. You can add data by creating partitions for the data.
- Partitioning allows you to optimize some queries. You can store the data that meets the conditions in the specified WHERE clauses in one or more partitions. In this scenario, when you run the corresponding queries, the system does not need to scan the other partitions. After you create partitioned tables, you can modify the partitions of the partitioned tables. Therefore, if this query optimization method is not used when you configure the partitioning scheme for the first time, you can modify the partitions to reorganize your data. This improves the efficiency of frequently run queries.
- Partitioning offers an easy method for you to process the queries that involve aggregate functions such as SUM() and COUNT() in parallel. The `SELECT salesperson_id, COUNT(orders) as order_total FROM sales GROUP BY salesperson_id;` statement is used as an example. In this example, the query can be performed on each table partition in parallel. The statement returns the final result by summarizing the result of each partition.
- Partitioning allows you to run data queries across disks to maximize the I/O throughput.

Partitioning types

The following list describes the supported partitioning types:

- Range partitioning

  Rows are assigned to partitions based on a range of continuous column values that you specify for each partition.

- Hash partitioning

  Data is assigned to partitions based on the results that are returned by user-defined expressions. The expressions use the column values in the rows that are to be inserted into the specified tables to calculate results.

  Hash functions can contain valid expressions that return non-negative integers in ApsaraDB for OceanBase.

  Partitions are automatically numbered when you create the partitions, regardless of partitioning types. The sequence numbers of partitions start from 0. When you insert a row into a partitioned table, the system uses the partition sequence number to identify the partition into which the row is inserted.

  For example, if your partitioned table has four partitions, the sequence numbers of the four tables are 0, 1, 2, and 3. If you use range partitioning, make sure that you define a range for each partition. If you use hash partitioning, each user-defined function must return an integer that is greater than zero. Partition names are not case-sensitive.

- Key partitioning

  Key partitioning is similar to hash partitioning. Hash partitioning uses user-defined expressions, and key partitioning uses hash functions that are provided by ApsaraDB for OceanBase servers. The column values on which key partitioning is implemented can be integers or values of other data types.

  Note Key partitioning in ApsaraDB for OceanBase uses MurmurHash functions.

5.3.2. Range partitioning

Syntax

```
...
 PARTITION BY RANGE {(expr) | COLUMNS(column_list)}
     (partition_definition [, partition_definition] ...)
 partition_definition:
     PARTITION partitionname
     VALUES {LESS THAN {(expr | value_list) | MAXVALUE}
```

For range partitioning, the specified ranges must be continuous and cannot overlap with each other. You must define the ranges by using the `VALUES LESS THAN` operator.

Examples

If you use range partitioning, each partition contains the rows in which the values fall in the specified range of continuous numeric values. The row values are returned by the expressions that you specify to implement range partitioning.

In the following examples, the employees table is used. The table stores employee records for 20 video stores that are numbered from 1 to 20.

The employees table is created by executing the following statement:

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
 store_id INT NOT NULL
 );
```

You can use multiple methods to implement range partitioning on the employees table based on your business requirements.

In one of these methods, you can use the store_id column. For example, you can split the table into four partitions by adding a `PARTITION BY RANGE` clause to the statement.

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
 store_id INT NOT NULL
 )
 PARTITION BY RANGE (store_id) (
     PARTITION p0 VALUES LESS THAN (6),
     PARTITION p1 VALUES LESS THAN (11),
     PARTITION p2 VALUES LESS THAN (16),
     PARTITION p3 VALUES LESS THAN (21)
 );
```

In this partitioning scheme, p0 stores the rows for the employees of stores 1 to 5, p1 stores the rows for the employees of stores 6 to 10, and so on.

Notice The sequence numbers of the stores are sorted in ascending order. Each partition is defined based on the specified range of store sequence numbers. This meets the requirements of the `PARTITION BY RANGE` syntax. The requirements are similar to the syntax requirements of C or Java `switch … case` statements.

A new row that contains the data such as `72, 'Michael', 'Widenius', '1998-06-25', NULL, 13` can be inserted into the p2 partition as expected. However, if you need to insert rows for store 21, errors are reported. This is because no rules cover the rows whose store_id values are greater than 20 and ApsaraDB for OceanBase servers cannot determine the partitions that store the rows.

To prevent errors of this type, you can execute a `CREATE TABLE` statement that includes the `catchall` `VALUES LESS THAN` clause. You can use the clause to specify the partition in which you store the values that are greater than the explicitly specified maximum value.

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
 store_id INT NOT NULL
 )
 PARTITION BY RANGE (store_id) (
     PARTITION p0 VALUES LESS THAN (6),
     PARTITION p1 VALUES LESS THAN (11),
     PARTITION p2 VALUES LESS THAN (16),
     PARTITION p3 VALUES LESS THAN MAXVALUE
 );
```

MAXVALUE specifies the maximum allowed integer. All the rows whose store_id column values are greater than or equal 16 are stored in the p3 partition. In this example, the number 16 is the maximum value that is specified in an explicit way. When the number of stores increases to 25, 30, or another larger value, you can execute the `ALTER TABLE` statement to add partitions for the added stores. For example, you can add a partition for stores 21 to 25 and add another partition for stores 26 to 30.

The table in the following example has the same schema as that in the preceding example. You can partition the table based on the continuous ranges of job_code column values. The job_code column values specify the job codes of employees. In this example, two-digit job codes represent regular in-store workers, three-digit job codes represent office and support personnel, and four-digit codes represent management personnel. You can execute the following statement to create the partitioned table:

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT NOT NULL,
 store_id INT NOT NULL
 )
 PARTITION BY RANGE (job_code) (
     PARTITION p0 VALUES LESS THAN (100),
     PARTITION p1 VALUES LESS THAN (1000),
     PARTITION p2 VALUES LESS THAN (10000)
 );
```

In this example, the rows for the in-store workers are stored in the p0 partition. The rows for the office and support personnel are stored in the p1 partition, and the rows for the management personnel are stored in the p2 partition.

You can also use an expression in the `VALUES LESS THAN` clause. Note that the returned values of the expression must be able to be used for `LESS THAN` (<) comparison to determine the ranges into which the returned values fall. Therefore, the returned values cannot be NULL. Due to the same reason, the values in the hired, separated, job_code, and store_id columns of the employees table are defined as non-NULL values.

In the first example, the employees table is partitioned based on the sequence numbers of stores. You can also partition the table by using an expression based on onboarding dates or offboarding dates. For example, you want to partition the table based on the YEAR (separated) values. The values specify the years when employees left the company. In this example, the following `CREATE TABLE` statement is executed to implement the partitioning scheme:

```
CREATE TABLE employees (
 id INT NOT NULL,
 fname VARCHAR(30),
 lname VARCHAR(30),
 hired DATE NOT NULL DEFAULT '1970-01-01',
 separated DATE NOT NULL DEFAULT '9999-12-31',
 job_code INT,
 store_id INT
 )
 PARTITION BY RANGE (YEAR(separated)) (
     PARTITION p0 VALUES LESS THAN (1991),
     PARTITION p1 VALUES LESS THAN (1996),
     PARTITION p2 VALUES LESS THAN (2001),
     PARTITION p3 VALUES LESS THAN MAXVALUE
 );
```

In this partitioning scheme, the p0 partition stores the rows for the employees who left the company before 1991. The p1 partition stores the rows for the employees who left the company between 1991 and 1995. The p2 partition stores the rows for the employees who left the company between 1996 and 2000. The p3 partition stores the rows for the employees who left the company after 2000.

Scenarios

- You need to delete the previous data.
- You need to use a column that stores date or time values that are sorted in ascending order.
- You need to frequently run the queries that are based on partition key columns.

5.3.3. Hash partitioning

Syntax

If you use hash partitioning, you must specify column values or expressions for the columns on which hashing is to be implemented. You must also specify the number of partitions for each partitioned table. ApsaraDB for OceanBase automatically implements hash partitioning based on the specified settings.

```
 ...
 PARTITION BY HASH (expr)
     PARTITIONS num
```

If you use hash partitioning to partition a table, you must add the following clause to the `CREATE TABLE` statement: `PARTITION BY HASH (expr)`. In the clause, expr specifies an expression that returns integers. You can also specify expr as the name of a column. The column must store integers in ApsaraDB for OceanBase. You may need to append a `PARTITIONS num` clause to the PARTITION BY HASH (expr) clause. In the PARTITIONS num clause, num specifies the number of partitions for the table. You must specify this parameter as a positive integer.

Examples

You can partition the table based on the years when employees were hired by executing the following statement:

```
CREATE TABLE employees (
id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT
)
PARTITION BY HASH(YEAR(hired))
PARTITIONS 4
```

Scenarios

Hash partitioning ensures that data in tables is evenly distributed across table partitions. The number of partitions for each table is specified when you create the table.

5.3.4. Key partitioning

Syntax

If you use key partitioning to partition tables, the tables are partitioned based on the hash functions that are provided by ApsaraDB for OceanBase. Key partitioning uses one column or a list of columns as the partition key.

```
...
PARTITION BY KEY(column_list)
 PARTITIONS num
```

Examples

Execute the following statement to create the k1 table and partition the table based on the id field:

```
mysql> create table k2(id int primary key, name varchar(20))
-> partition by key() partitions 2;
Query OK, 0 rows affected (0.29 sec)
```

In this example, id is the primary key field. `partition by key()` and `partition by key(id)` are equivalent.

5.3.5. Subpartitioning

Syntax

Subpartitioning is also known as composite partitioning. Subpartitioning divides each partition of a partitioned table into subpartitions.

```
...
PARTITION BY RANGE(expr)
SUBPARTITION BY [HASH|KEY](expr) SUBPARTITIONS N
(PARTITION P0 VALUES LESS THAN (V0), …)
PARTITION BY [HASH|KEY](expr)
SUBPARTITION BY RANGE(expr) SUBPARTITION TEMPLATE (SUBPARTITION P0
VALUES LESS THAN (V0), …)
PARTITIONS N
...
```

Examples

```
CREATE TABLE ts (id INT, purchased DATE,PRIMARY KEY (id, purchased))
PARTITION BY RANGE(YEAR(purchased))
SUBPARTITION BY HASH(TO_DAYS(purchased))
SUBPARTITIONS 2
(
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE
)
```

In this example, the ts table has three range partitions: p0, p1, and p2. Each partition is divided into two subpartitions. As a result, the entire table is divided into six partitions. The number of partitions is calculated based on the formula: `3 × 2 = 6` . Based on the settings in the `PARTITION BY RANGE` clause, the first two partitions store only the rows whose values in the purchased column are smaller than 1990. The preceding statement is equivalent to the following statement:

```
CREATE TABLE ts (id INT, purchased DATE, PRIMARY KEY (id, purchased))
PARTITION BY RANGE(YEAR(purchased))
SUBPARTITION BY HASH(TO_DAYS(purchased))
(
    PARTITION p0 VALUES LESS THAN (1990)
    (
        SUBPARTITION s0,
        SUBPARTITION s1
    ),
    PARTITION p1 VALUES LESS THAN (2000)
    (
        SUBPARTITION s2,
        SUBPARTITION s3
    ),
    PARTITION p2 VALUES LESS THAN MAXVALUE
    (
        SUBPARTITION s4,
        SUBPARTITION s5
    )
);
```

Notes

Pay attention to the following syntax considerations when you implement subpartitioning:

- Each partition must have the same number of subpartitions.
- The primary key must cover all the partition key columns.
- If subpartitions are created based on ranges, we recommend that you use subpartitioning templates to create subpartitions.

```
CREATE TABLE ts (id INT, purchased DATE,PRIMARY KEY (id, purchased))
PARTITION BY HASH(id)
SUBPARTITION BY RANGE(YEAR(purchased))
SUBPARTITION TEMPLATE
(
    SUBPARTITION p0 VALUES LESS THAN (1990),
    SUBPARTITION p1 VALUES LESS THAN (2000),
    SUBPARTITION p2 VALUES LESS THAN MAXVALUE
)
PARTITIONS 2
```

- If you need to use the `SUBPARTITION` clause to define subpartitions for a partition of a partitioned table in an explicit way, you must define all the subpartitions of the partitioned table. Otherwise, the following statement fails to be executed:

  ```
  CREATE TABLE ts (id INT, purchased DATE,PRIMARY KEY (id, purchased))
  PARTITION BY RANGE(YEAR(purchased))
  SUBPARTITION BY HASH(TO_DAYS(purchased))
  (
      PARTITION p0 VALUES LESS THAN (1990)
      (
          SUBPARTITION s0,
          SUBPARTITION s1
      ),
      PARTITION p1 VALUES LESS THAN (2000),
      PARTITION p2 VALUES LESS THAN MAXVALUE
      (
          SUBPARTITION s2,
          SUBPARTITION s3
      )
  )
  ```

  An execution failure occurs even if the statement includes the `SUBPARTITIONS 2` clause.

- Each SUBPARTITION clause must include at least the subpartition name. Otherwise, you may need to specify a desired option for the subpartition or allow the subpartition to use the default settings for the option.
- Subpartition names must be unique across each partition. Subpartition names do not need to be unique across the entire table.

  For example, the following CREATE TABLE statement is valid:

  ```
  CREATE TABLE ts (id INT, purchased DATE,PRIMARY KEY (id,
  purchased) )
   PARTITION BY RANGE(YEAR(purchased))
   SUBPARTITION BY HASH(TO_DAYS(purchased))
   (
       PARTITION p0 VALUES LESS THAN (1990)
       (
           SUBPARTITION s0,
           SUBPARTITION s1
       ),
       PARTITION p1 VALUES LESS THAN (2000)
       (
           SUBPARTITION s0,
           SUBPARTITION s1
       ),
       PARTITION p2 VALUES LESS THAN MAXVALUE
       (
           SUBPARTITION s0,
           SUBPARTITION s1
       )
   );
  ```

5.3.6. Handle NULL values

Handle NULL as zero

If a column value or the value of a user-defined expression is NULL, NULL must be handled to implement partitioning. In most cases, NULL is handled as zero.

If you do not want to use this method of handling NULL, you can implement the NOT NULL constraint when you create tables. To implement the constraint, we recommend that you declare `NOT NULL` for the columns when you create the tables.

Note The NULL value indicates that the calculation result of an expression is NULL. NULL is handled as zero.

Examples

Assume that you insert a row that contains NULL values into a table that is partitioned by range. In the row, NULL is the column value that determines the partition. In this scenario, the system handles NULL as zero and assigns the row to the partition whose value range contains zero.

The tnrange and tnhash tables are used in the examples in this topic. In the following example, the tnrange table is created and a row is inserted into the table:

```
mysql> CREATE TABLE tnrange (
     -> id INT,
     -> name VARCHAR(5)
     -> )
     -> PARTITION BY RANGE(id) (
     -> PARTITION p1 VALUES LESS THAN (1),
     -> PARTITION p2 VALUES LESS THAN MAXVALUE
     -> );
 mysql> INSERT INTO tnrange VALUES (NULL, 'jim');
 mysql> SELECT * FROM tnrange;
 +------+------+
 | id | name |
 +------+------+
 | NULL | jim |
 +------+------+
 1 row in set (0.00 sec)
```

In the tnrange table, the `NOT NULL` constraint is not declared for the id column. Therefore, the id column can contain NULL values. In the following example, the ALTER TABLE statement is executed to delete the partition that contains NULL values. The SELECT statement is executed to check whether the rows that contain NULL values are stored in the p1 partition of the table.

```
mysql> ALTER TABLE tnrange DROP PARTITION p1;
Query OK, 0 rows affected (0.16 sec)
mysql> SELECT * FROM tnrange;
Empty set (0.00 sec)
```

If you use hash partitioning, an expression that returns NULL is handled as an expression that returns zero. To verify this rule, you can create a table that is partitioned based on hashing and insert a row that contains NULL into the table. Then, you can check whether the row is inserted into the table as expected. For example, execute the following statement to create the tnhash table in the test database:

```
CREATE TABLE tnhash (
 id INT,
 name VARCHAR(5)
 )
 PARTITION BY HASH(id)
     PARTITIONS 2;
```

Execute the INSERT INTO statement to insert a row into the tnhash table. The value in the id column for this row is NULL. Then, execute the SELECT statement to check whether the row is inserted as expected.

```
mysql> INSERT INTO tnhash VALUES (NULL, 'sam');
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM tnhash;
+------+------+
| id | name |
+------+------+
| NULL | sam |
+------+------+
1 row in set (0.01 sec)
```

For an integer that is represented by N, the value of `NULL MOD N` is always NULL. The system handles NULL as zero when the data is assigned to partitions. This example assumes that the Bourne Again Shell (Bash) is used to check whether the row that contains NULL is inserted to the partition as expected. Therefore, you can go back to the system shell to perform the check. In the Bash shell, execute the SELECT statement to query the data that is stored in the p0 partition of the tnhash table. This allows you to check whether the row is inserted into the first partition whose default name is p0.

```
mysql> SELECT * FROM tnhash partition(p0);
+------+------+
| id | name |
+------+------+
| NULL | sam |
+------+------+
1 row in set (0.00 sec)
```

5.4. Data types

5.4.1. Overview

ApsaraDB for OceanBase supports three categories of data types: numeric types, string types, and date and time types. describes the data types that ApsaraDB for OceanBase supports.

Supported data types

Category	Data type
Numeric types	• TINYINT • BOOL • BOOLEAN • SMALLINT • MEDIUMINT • INT • INTEGER • BIGINT • FLOAT • DOUBLE • DECIMAL **Note** The precision for this data type is inconsistent with that for the DECIMAL data type in MySQL. • DEC • NUMERIC • BIT
String types	• CHAR • VARCHAR • BINARY • VARBINARY • ENUM • SET
Date and time types	• DATE • DATETIME • TIMESTAMP • TIME • YEAR

5.4.2. Numeric types

The numeric types are divided into the following groups:

- Integers

  describes the integer data types.

  #### Integer data types

Data type	Storage size (bytes)	Range (signed or unsigned)	Minimum value (signed or unsigned)	Maximum value (signed or unsigned)
TINYINT	1	2^7-1	-128	127
		2^8-1	0	255
SMALLINT	2	2^{15}-1	-32768	32767
		2^{16}-1	0	65535
MEDIUMINT	3	2^{23}-1	-8388608	8388607
		2^{24}-1	0	16777215
INT	4	2^{31}-1	-2147483648	2147483647
		2^{32}-1	0	4294967295
BIGINT	8	2^{63}-1	-9223372036854775808	9223372036854775807
		2^{64}-1	0	18446744073709551615

- Floating-point numbers or decimal numbers

  describes the floating-point data types.

Floating-point data types

Data type	Storage size (bytes)	Description
FLOAT	4	This data type stores single-precision floating-point numbers.
FLOAT(*p*)	If `0 ≤ p ≤ 24`, the storage size is 4 bytes. If `25 ≤ p ≤ 53`, the storage size is 8 bytes.	If the precision ranges from 0 to 24 digits, the column type is FLOAT and each value in the FLOAT column is a single-precision floating-point number that occupies 4 bytes. If the precision ranges from 25 to 53 digits, the column type is DOUBLE and each value in the DOUBLE column is a double-precision floating-point number that occupies 8 bytes.
DOUBLE [PRECISION]	8	This data type stores double-precision floating-point numbers.
DECIMAL (M,D) NUMERIC (M,D)	Variable length	This data type stores fixed-point numbers.

Other types

Data type	Storage size (bytes)	Description
BIT(M)	The occupied bytes are calculated based on the formula: Number of occupied bytes = (M + 7)/8. The default value of M is 1 and the valid value range for M is 1 to 64.	This data type stores binary values.

Note

- You can specify the display width for the values of TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT data types. The format is data type(M), such as INT(20). In the format, M represents the maximum display width. The maximum valid display width is 255 bytes.

  The display width is irrelevant to the storage size or the value range of each data type.
- The UNSIGNED modifier indicates that only positive values can be stored in the specified field.
- The ZEROFILL modifier indicates that output values are padded with 0 instead of spaces. If you specify the ZEROFILL attribute for a numeric column, the system automatically adds the UNSIGNED attribute to the column.
- SERIAL is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE` .

  In the definition of integer columns, `SERIAL DEFAULT VALUE` is an alias for `NOT NULL AUTO_INCREMENT UNIQUE` .

TINYINT

```
TINYINT[(M)] [UNSIGNED] [ZEROFILL]
```

The TINYINT data type stores small one-byte integers.

The range of signed numbers for the data type is -128 to 127. The range of unsigned numbers for the data type is 0 to 255.

BOOL or BOOLEAN

BOOL or BOOLEAN is a synonym of TINYINT(1).

Zero values are handled as false, and non-zero values are handled as true.

SMALLINT

```
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]
```

The SMALLINT data type stores small two-byte integers.

The range of signed numbers for the data type is -32768 to 32767. The range of unsigned numbers for the data type is 0 to 65535.

MEDIUMINT

```
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]
```

The MEDIUMINT data type stores medium-sized integers.

The range of signed numbers for the data type is -8388608 to 8388607. The range of unsigned numbers for the data type is 0 to 16777215.

INT

```
INT[(M] [UNSIGNED] [ZEROFILL]
```

The INT data type stores normal-size integers.

The range of signed numbers for the data type is -2147483648 to 2147483647. The range of unsigned numbers for the data type is 0 to 4294967295.

Before you insert an invalid integer value into a table, the system automatically converts the value to zero.

INTEGER

```
INTEGER[(M)] [UNSIGNED] [ZEROFILL]
```

INTEGER is a synonym for INT.

BIGINT

```
BIGINT[(M)] [UNSIGNED] [ZEROFILL]
```

The BIGINT data type stores large integers.

The range of signed numbers for the data type is -9223372036854775808 to 9223372036854775807. The range of unsigned numbers for the data type is 0 to 18446744073709551615.

Signed BIGINT or DOUBLE values are used for all the arithmetic operations. Therefore, you cannot use the unsigned large integers that are larger than 9223372036854775807 (63 bits) in arithmetic operations except bit functions. If you use the unsigned large integers that are larger than 9223372036854775807, the last few digits in the result may be inaccurate. This is because rounding is implemented when you convert the values of the BIGINT data type into the values of the DOUBLE data type.

The system handles the values of the BIGINT data type in the following scenarios:

- A column of the BIGINT data type stores large unsigned integers.
- In MIN(col_name) or MAX(col_name), col_name specifies a BIGINT column.
- Operators such as plus signs (+), minus signs (-), and asterisks (*) are used and both operands are integers.
- Strings can be used to store exact integer values in columns of the BIGINT type. In this scenario, strings are converted into numbers and no intermediate double-precision numbers are generated during the conversion.
- If both operands of each plus sign (+), minus sign (-), or asterisk (*) operator are integers, the corresponding operator performs BIGINT operations. In this scenario, if you multiply two large integers that are returned by functions and the product is greater than 9223372036854775807, unexpected results are returned.

BIT

`BIT[(M)]`

The BIT data type stores binary values. *M* specifies the number of bits that can be stored. The valid value ranges from 1 to 64.

For example, if you specify M as 4, the binary values that can be stored range from 0000 to 1111. You can insert data to a column of the BIT data type by using `b'value' or 0bvalue` , such as `b'1001'` or `0b1001` .

Note In the contexts of numeric values, the BIT data type stores `numeric values` . In the contexts of strings, the BIT data type stores `string values` . The `numeric values` are the numbers that represent the binary values. The `string values` are the strings that represent the binary values. When you assign values of a BIT column to variables, the assigned values are `numeric values` .

FLOAT

```
FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]
```

The FLOAT data type stores small single-precision floating-point numbers.

The valid value range is -2^{128} to $+2^{128}$. The valid value range can also be expressed as -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38. This valid range specifies the theoretical limits based on the IEEE standard. The actual range may be smaller than the valid range because different hardware devices or operating systems are used.

M is the total number of digits and *D* is the number of digits that follow the decimal point. If *M* and *D* are not specified, values are stored based on the hardware limits. Each single-precision floating-point number is accurate to about seven decimal places.

Negative values are disallowed if the UNSIGNED attribute is specified.

Note If you use the values of the FLOAT data type for calculations, unexpected errors may occur. To prevent these errors, the values of the FLOAT data type are converted into the values of the DOUBLE data type before calculations are performed.

DOUBLE

```
DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]
```

The DOUBLE data type stores normal-sized and double-precision floating-point numbers.

The valid value range is -2^{1024} to $+2^{1024}$. The valid value range can also be expressed as -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308. This valid range specifies the theoretical limits based on the IEEE standard. The actual range may be smaller than the valid range because different hardware devices or operating systems are used.

M is the total number of digits and *D* is the number of digits that follow the decimal point. If *M* and *D* are not specified, values are stored based on the hardware limits. Each double-precision floating-point number is accurate to about 15 decimal places.

Negative values are disallowed if the UNSIGNED attribute is specified.

DOUBLE PRECISION

```
DOUBLE PRECISION [(M,D)] [UNSIGNED] [ZEROFILL], REAL[(M,D)] [UNSIGNED] [ZEROFILL]
```

DOUBLE PRECISION is a synonym of DOUBLE.

FLOAT(*p*)

```
FLOAT(p) [UNSIGNED] [ZEROFILL]
```

The FLOAT(p) data type stores floating-point numbers. *p* specifies the precision of a numeric value. The precision is represented by the number of digits in the numeric value. The value of the p parameter is used to only determine whether the data type of an output column is FLOAT or DOUBLE.

If the value of p ranges from 0 to 24, the data type is FLOAT and *M* or *D* values do not need to specified. If the value of p ranges from 25 to 53, the data type is DOUBLE and *M* or *D* values do not need to be specified. The value range of the output column is the same as the value range for the FLOAT or DOUBLE data type that is described in this topic. The FLOAT data type stores single-precision floating-point numbers. The DOUBLE data type stores double-precision floating-point numbers.

DECIMAL (different from the DECIMAL type in MySQL)

```
DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]
```

The DECIMAL data type stores packed exact fixed-point numbers. *M* is the total number of digits and specifies the precision of a numeric value. *D* is the number of digits that follow the decimal point and specifies the scale of the numeric value. The number of decimal points and negative number signs (-) is excluded from the *M* value. If the value of *D* is 0, values does not have decimal points or fractional parts. For the DECIMAL data type in ApsaraDB for OceanBase, the maximum number of digits that is specified by *M* is 38. In MySQL, the maximum number of digits is 65. *D* specifies the number of digits that follow the decimal point. The maximum number of digits that follow the decimal point for decimal numeric values is 30. If *D* is not specified, the default value 0 is used. If *M* is not specified, the default value 10 is used.

Negative values are disallowed if the UNSIGNED attribute is specified.

For DECIMAL columns in ApsaraDB for OceanBase, basic operations are performed based on a precision of 38 digits. The basic operations are addition (+), subtraction (-), multiplication (*), and division (/). In MySQL, the basic operations are performed based on a precision of 65 digits.

`DEC[(M[,D])] [UNSIGNED] [ZEROFILL], NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]` are synonyms of DECIMAL.

NUMERIC

`NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]` is a synonym of DECIMAL.

5.4.3. String types

describes the string types that ApsaraDB for OceanBase supports.

String types

String type	Number of bytes	Description
CHAR	0-255	The length of a CHAR column is the fixed length that you specify when you create your table. The value length ranges from 0 to 255 bytes. When CHAR values are stored, spaces are padded to the right of each value until the specified length is reached. When CHAR values are retrieved, the trailing spaces of the values are removed. When CHAR values are stored or retrieved, uppercase and lowercase conversion is not performed.
VARCHAR	0-262143	The maximum length of a VARCHAR value is determined by the maximum length for the row values of the VARCHAR data type and the character set that you use. In ApsaraDB for OceanBase, the maximum valid length is 256 KB. In MySQL, the maximum value length is 64 KB.

String type	Number of bytes	Description
BINARY	0-255	This data type is similar to the CHAR data type. The difference is that the columns of the BINARY data type store binary strings instead of non-binary strings.
VARBINARY	0-262143	This data type is similar to the VARCHAR data type. The difference is that the columns of the VARBINARY data type store binary strings instead of non-binary strings.

CHAR

```
CHAR[(M)] [CHARACTER SET charsetname] [COLLATE collationname]
```

CHAR is short for CHARACTER.

The length of a CHAR column is the fixed length that you specify when you create your table. The value length ranges from 0 to 255 bytes. When CHAR values are stored, spaces are padded to the right of each value until the specified length is reached.

VARCHAR

```
VARCHAR(M) [CHARACTER SET charsetname] [COLLATE collationname]
```

This data type stores variable-length strings.

M specifies the maximum value length of a column. The valid value range of *M* is 0 to 256 KB. The actual length that is required to store each value of a VARCHAR field is determined by the length of the real-time field value and the specific character set. For example, you can specify the maximum valid length as 256 KB characters.

If the system uses the UTF8MB4 character set, each character occupies 4 bytes and the maximum valid length is calculated based on the formula: 256 KB × 4 = 1 MB.

VARCHAR is short for CHAR VARYING.

When VARCHAR values are stored, the values are not padded with characters.

If the length of a value that is assigned to a CHAR or VARCHAR column exceeds the maximum length for the column, the value is truncated to meet the length requirement. If spaces are truncated, a warning message is returned. If non-space characters are truncated, an error message instead of a warning message is reported. You can enable the strict SQL mode to prevent the corresponding values from being inserted into the CHAR or the VARCHAR column.

BINARY, VARBINARY

BINARY and VARBINARY data types are similar to CHAR and VARCHAR data types. The difference is that the columns of BINARY and VARBINARY types store binary strings instead of non-binary strings. This means that the columns of the BINARY and VARBINARY types store byte strings instead of character strings. The columns of BINARY and VARBINARY types do not use character sets. The values in these columns are compared and sorted based on the numeric values that are converted from the byte values in the columns.

The maximum value length for a column of the BINARY or VARBINARY data type is the same as that for a column of the CHAR or VARCHAR data type. The maximum value length for a column of the BINARY or VARBINARY type is measured in bytes instead of characters.

5.4.4. Date and time data types

describes the date and time data types that are supported by ApsaraDB for OceanBase SQL.

Date and time data types

Date and time data type	Format	Value range	Size (bytes)
DATE	YYYY-MM-DD	'1000-01-01' to '9999-12-31'	3
DATETIME	YYYY-MM-DD HH:MM:SS	'1000-01-01 00:00:00' to '9999-12-31 23:59:59'	8
TIMESTAMP	YYYY-MM-DD HH:MM:SS	'1970-01-01 00:00:00' to '2037-12-31 23:59:59'	8
TIME	HH:MM:SS	'-838:59:59' to '838:59:59'	3
YEAR	YYYY (default format)	'1901 to 2155' and '0000'	1
	YY	'70 to 69'. This value range represents years 1970 to 2069.	

By default, the precision for the values of DATETIME, TIMESTAMP, and TIME data types is seconds in date and time functions. You can set the fsp parameter to specify the precision for the fractional second part.

You can specify the precision based on your storage requirements. The fractional second part occupies 0 to 3 bytes based on the specified precision. The most fine-grained time granularity for the fractional second part is microseconds. If you use the microsecond precision, the fractional second part occupies 3 bytes.

The syntax is `type_name(fsp)` .

The `type_name` parameter specifies the data type, such as DATETIME, TIMESTAMP, and TIME.

The *fsp* parameter specifies the precision of the fractional second part. The values of this parameter range from 0 to 6. The default value is 0. The largest value is 6. The value 6 indicates that the precision of the fractional second part is microseconds.

DATE

The DATE data type stores values that consist of only the date part. The supported value range is `1000-01-01` to `9999-12-31` .

ApsaraDB for OceanBase uses the `YYYY-MM-DD` format for DATE values. ApsaraDB for OceanBase allows you to use strings or numbers to assign values to DATE columns.

DATETIME

```
DATETIME[(fsp)]
```

The DATETIME data type stores values that consist of a date part and a time part. The supported value range is `1000-01-01 00:00:00.000000` to `9999-12-31 23:59:59.000000` . ApsaraDB for OceanBase uses the `YYYY-MM-DD HH:MM:SS[.fraction]` format for DATETIME values. ApsaraDB for OceanBase allows you to use strings or numbers to assign values to DATETIME columns.

The *fsp* parameter specifies the precision of the fractional second part. The values of this parameter range from 0 to 6. The default value is 0. The largest value is 6. The value 6 indicates that the precision of the fractional second part is microseconds.

DATETIME and TIMESTAMP data types support `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses.

TIMESTAMP

```
TIMESTAMP[(fsp)]
```

The TIMESTAMP data type in ApsaraDB for OceanBase is inconsistent with that in MySQL. This is because the format of the `TIMESTAMP` data type in ApsaraDB for OceanBase complies with strict mode requirements. Invalid values such as `0000-00-00 00:00:00` are disallowed in ApsaraDB for OceanBase.

The following string values are invalid:

```
  '2012^12^32'
  '20070523'
  '070523'
  '071332'
```

The following integer values are invalid:

```
  19830905
  830905
```

`DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses

If you use `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses, the TIMESTAMP or DATETIME column uses the current timestamp as the default value. The timestamp in the column is automatically updated to the current timestamp.

If you specify the value in the first `TIMESTAMP` column in a table as the `default value`, the DEFAULT clause cannot be ignored. You can specify the `current timestamp` or a constant date and time value as the default value.

- For the first `TIMESTAMP` column in a table, the `DEFAULT NULL` clause is the same as the `DEFAULT CURRENT_TIMESTAMP` clause. For the other `TIMESTAMP` columns in the table, the `DEFAULT NULL` clause is considered as `DEFAULT 0`.
- In the `CREATE TABLE` statement, you can use the following methods to declare the first `TIMESTAMP` column:
  - If you use `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses, the column uses the current timestamp as the default value. The timestamp in the column is automatically updated to the current timestamp.
  - If you use the `DEFAULT CURRENT_TIMESTAMP` clause and do not use the `ON UPDATE` clause, the column uses the current timestamp as the default value. In this scenario, the timestamp in the column is not automatically updated to the current timestamp.
  - If you do not use the `DEFAULT` clause and use the `ON UPDATE CURRENT_TIMESTAMP` clause, the default value of the column is 0. The timestamp in the column is automatically updated to the current timestamp.
  - If you specify a constant in the `DEFAULT` clause, the column uses the specified constant as the default value. If the `ON UPDATE CURRENT_TIMESTAMP` clause is specified for the column, the timestamp of the column is automatically updated to the current timestamp. Otherwise, automatic updates are not performed.

    In other words, you can use the current timestamp as either, neither, or both of the default value and the auto-update value.

    For example, you can use `ON UPDATE` to enable automatic updates of timestamps and avoid automatic initialization for the column.
  - In `DEFAULT` and `ON UPDATE` clauses, you can use `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP()`, or `NOW()`. These functions are equivalent.

    The order of the DEFAULT and ON UPDATE clauses in the statement does not affect the returned results. If the `DEFAULT clause` and the `ON UPDATE` clause are specified for a TIMESTAMP column at the same time, either of the clauses can occur first.

    In the following examples, the statements are equivalent:

    ```
    CREATE TABLE t (ts TIMESTAMP);
    CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);
    CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
    ```
  - Assume that you need to specify a default value or enable automatic updates of timestamps for all the `TIMESTAMP` columns instead of only the first TIMESTAMP column. In this case, you must specify a constant in the `DEFAULT` clause for the first `TIMESTAMP` column to disable automatic initialization and updates. For example, you can specify `DEFAULT 0` or `DEFAULT'2003-01-01 00:00:00'`. The rules for the other `TIMESTAMP` columns are the same as those for the first `TIMESTAMP` column, except that `DEFAULT` and `ON UPDATE` clauses cannot be ignored for the other columns. If the DEFAULT and ON clauses are ignored, automatic initialization and updates cannot apply to the timestamps.

    In the following examples, the statements are equivalent:

    ```
    CREATE TABLE t (
        ts1 TIMESTAMP DEFAULT 0,
        ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);
    CREATE TABLE t (
        ts1 TIMESTAMP DEFAULT 0,
    ts2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
    ```

NULL values in TIMESTAMP columns

You can specify the NULL attribute for TIMESTAMP columns so that the columns can store NULL values.

Examples

```
CREATE TABLE tstest
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
INSERT INTO tstest values(NULL, NULL, NULL);
SELECT * FROM tstest;
//Result
mysql> SELECT * FROM tstest;
+------+------+------+
| ts1  | ts2  | ts3  |
+------+------+------+
| NULL | NULL | NULL |
+------+------+------+
1 row in set (0.00 sec)
```

TIME

```
TIME[(fsp)]
```

The TIME data type stores values that consist of only the time part. The supported value range is -838:59:59.000000 to 838:59:59.000000. ApsaraDB for OceanBase uses the `HH:MM:SS[.fraction]` format for TIME values. ApsaraDB for OceanBase allows you to use strings or numbers to assign values to TIME columns.

The *fsp* parameter specifies the precision of the fractional second part. The values of this parameter range from 0 to 6. The default value is 0. The largest value is 6. The value 6 indicates that the precision of the fractional second part is microseconds.

`D TIME values can be strings in the D HH:MM:SS[.fraction]` format. You can use the following non-strict SQL syntax: `H:MM:SS[.fraction]` , `HH:MM:SS` , `HH:MM` , `D HH:MM:SS` , `D HH:MM` , `D HH` , or `SS` . *D* represents days. The valid values range from 0 to 34.

TIME values can be strings in the `HHMMSS` format. In the format, no delimiters are used. The system assumes that the strings specify valid time. For example, `'101112'` is valid and represents `'10:11:12'` . However, `'109712'` is invalid because the minute part 97 is invalid. The system changes the invalid string to `'00:00:00'` .

TIME values can be numeric values in the `HHMMSS` format. The system assumes that the numeric values specify valid time. For example, 101112 represents `'10:11:12'` . The system also supports numeric values in the following formats: SS, MMSS, HHMMSS, and HHMMSS.fraction.

TIME values can be the results that are returned by functions, such as `CURRENT_TIME` . The results must comply with the TIME value requirements.

Assume that a TIME value is a string where the hour, minute, and second parts are separated with colons (:). If the value for one of the parts is smaller than 10, you do not need to specify a two-digit value for the part. For example, `' 8:3:2'` and `'08:03:02'` are equivalent.

When you assign an abbreviated value to a TIME column, you must pay attention to the impacts of colons (:). If a TIME value does not contain colons (:), the system identifies the two rightmost digits in the TIME value as the second part. In this case, the system does not interpret the TIME value as the time on the current day. For example, you may interpret `'1112'` and 1112 as `11:12:00` : 12 minutes past 11. However, the system interprets '1112' and 1112 as `00:11:12` : 11 minutes and 12 seconds. Similarly, the system interprets `'12'` and 12 as `00:00:12` . If a TIME value contains colons (:), the system interprets the TIME value as the time on the current day. For example, the system interprets `'11:12'` as `'11:12:00'` instead of `'00:11:12'` .

If valid values fall out of the TIME value range, the values are converted into the closest boundaries of the value range. For example, `'-850:00:00'` is converted into `'-838:59:59'` and `'850:00:00'` is converted into `'838:59:59'` .

Invalid TIME values are converted into `'00:00:00'` . Note that `'00:00:00'` is a valid TIME value. Based on only the stored values in the table, you cannot distinguish the original `'00:00:00'` values from the `'00:00:00'` values that are converted from invalid values.

YEAR

The YEAR data type stores two-digit or four-digit values that represent years. By default, YEAR values use the four-digit format. In the four-digit format, valid YEAR values consist of 0000 and values that range from 1901 to 2155. In the two-digit format, valid YEAR values range from 70 to 69. The values represent years 1970 to 2069. YEAR values are displayed in the YYYY format. You can use strings or numbers to assign values to YEAR columns.

You can specify YEAR values in the following formats:

- Four-digit strings: The value range is `'1901'` to `'2155'`.
- Four-digit numbers: The value range is 1901 to 2155.
- Two-digit strings: The value range is `'00'` to `'99'`. The values in the range of `'00'` to `'69'` are converted into years 2000 to 2069. The values in the range of `'70'` to `'99'` are converted into years 1970 to 1999.
- Two-digit integers: The value range is 1 to 99. The values in the range of 1 to 69 are converted into years 2001 to 2069. The values in the range of 70 to 99 are converted into years 1970 to 1999. The difference between the range of two-digit integers and that of two-digit strings is that the range of two-digit integers does not include 0. This is because 0 cannot be specified as a number or interpreted as 2000. To use 0 to represent the year 2000, you must specify 0 as the string `'0'` or `'00'` or enable the system to interpret 0 as 0000.
- Results that are returned by functions such as NOW(): The returned results must comply with the YEAR value requirements.

  Invalid YEAR values are converted into 0000.

For DATETIME, DATE, TIMESTAMP, and YEAR data types, ApsaraDB for OceanBase use the following rules to interpret the dates that have ambiguous YEAR values:

- YEAR values in the range of 00 to 69 are converted into years 2000 to 2069.
- YEAR values in the range of 70 to 99 are converted into years 1970 to 1999.

You can use the `ORDER BY` clause to sort the two-digit YEAR values or the `TIMESTAMP` values that contain two-digit YEAR values.

You can use some functions such as MIN() and MAX() to convert `TIMESTAMP` values or YEAR values into numbers. Two-digit YEAR values are not applicable to these functions. In this case, you can convert the year parts of the `TIMESTAMP` values or the YEAR values into four-digit values. You can also use `MIN(DATE_ADD(TIMESTAMP,INTERVAL 0 DAYS))`.

5.5. Character sets

A character set is a set of symbols and encodings. A collation is a set of rules that are used to compare characters in a character set.

Supported character sets

ApsaraDB for OceanBase supports the UTF8MB4 character set. UTF8MB4 is a superset of UTF8 and uses a maximum of four bytes for each character.

Compared with UTF8, UTF8MB4 supports new characters in the iOS operating system, such as emojis. Characters that are supported by UTF8 are known as Basic Multilingual Plane (BMP) characters. The new characters that are supported by UTF8MB4 are known as supplementary characters.

You can specify character sets at different levels: tenant, database, table, field, and session. ApsaraDB for OceanBase supports only the UTF8MB4 character set. By default, the UTF8MB4 character set is used. In most cases, you do not need to specify the character set.

Note In ApsaraDB for OceanBase, the UTF8MB4 character set allows you to use UTF8. UTF8 is an alias for UTF8MB3.

Collation

In ApsaraDB for OceanBase, the supported collations for the UTF8MB4 character set are utf8mb4_bin and utf8mb4_general_ci. The default collation is utf8mb4_general_ci.

One of the main differences of the two collations is that they have different impacts on sorting orders and string comparisons. The utf8mb4_bin collation is case-sensitive and the utf8mb4_general_ci collation is not case-sensitive.

5.6. Auto-increment fields

5.6.1. Overview

This topic describes the AUTO_INCREMENT attribute of ApsaraDB for OceanBase.

ApsaraDB for OceanBase allows you to specify the AUTO_INCREMENT attribute for an integer field in a table. In this aspect, ApsaraDB for OceanBase is the same as MySQL. ApsaraDB for OceanBase can automatically generate unique values for auto-increment fields.

The following list describes the notes on the AUTO_INCREMENT attribute in ApsaraDB for OceanBase:

- You can specify the AUTO_INCREMENT attribute for fields of various integer data types, such as TINYINT, SMALLINT, MEDIUMINT, INT, INTEGER, BIGINT, FLOAT, and DOUBLE. You cannot specify the AUTO_INCREMENT attribute for fields of the other data types.

  Note In MySQL, you can specify the AUTO_INCREMENT attribute for fields of FLOAT, DOUBLE, and BOOLEAN data types. You cannot specify this attribute for fields of DECIMAL and BIT data types.

- You can specify the AUTO_INCREMENT attribute for only one field in a table.

```
mysql> create table t1 (id int auto_increment);
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
#In MySQL, auto-increment fields must have indexes that can be primary key indexes or general indexes. Otherwise, the p
receding error occurs. Indexes are not required for auto-increment fields in ApsaraDB for OceanBase. This is the differ
ence between ApsaraDB for OceanBase and MySQL.
```

```
mysql> create table t1 (id int auto_increment,name VARCHAR(20) primary key,key(id));
Query OK, 0 rows affected (0.01 sec)
#The AUTO_INCREMENT attribute is specified for the id field. The id field is a general field that has an index. Indexes
are not required for auto-increment fields in ApsaraDB for OceanBase. This is the difference between ApsaraDB for Ocean
Base and MySQL.
```

- In other cases, the following rules take effect:
  i. If you insert a NULL value into an `AUTO_INCREMENT` column, MySQL automatically generates the next sequence number for the column.
  ii. When you insert a row and do not explicitly specify a value for the `AUTO_INCREMENT` column, the system considers the value of the column as NULL. MySQL automatically generates the next sequence number for the column.
  iii. In all the modes except `NO_AUTO_VALUE_ON_ZERO` , if you insert 0 into an `AUTO_INCREMENT` column, the system considers the value of the column as NULL. MySQL automatically generates the next sequence number for this column.

```
mysql> insert into t1 (id,name) values (null,'test');
Query OK, 1 row affected (0.00 sec)
mysql> select * from t1;
+----+------+
| id | name |
+----+------+
|  1 | test |
+----+------+
1 row in set (0.00 sec)
```

The preceding statements are equivalent to the following statements:

```
mysql> insert into t1 (name) values ('test');
## Create a table.
mysql> CREATE TABLE t1 (id int AUTO_INCREMENT,PRIMARY KEY (id));
## Insert data into the table.
mysql> insert into t1 values (null),(null),(null);
Query OK, 3 rows affected (0.00 sec)
## Insert the value 7 into the table.
mysql> insert into t1 values (7);
Query OK, 1 row affected (0.00 sec)
## The value of the auto-increment column changes to 8.
mysql> show create table t1\G;
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t2` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8
```

5.6.2. System variables for auto-increment columns

```
mysql> show variables like '%auto_increment%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
```

You can specify global or session values for the preceding two system variables. The auto_increment_increment variable specifies the interval between two successive values in an auto-increment column. The auto_increment_offset variable specifies the start value of an auto-increment column.

The default value of the `auto_increment_increment` variable is 1. The values of this variable range from 1 to 65535.

The default value of the `auto_increment_offset` variable is 1. The values of this variable range from 1 to 65535.

Examples

- Example 1

```
auto_increment_increment=2
auto_increment_offset=1
mysql>create table t1(id int auto_increment primary key);
Query OK, 0 rows affected (0.00 sec)
mysql> set session auto_increment_increment=2;
Query OK, 0 rows affected (0.00 sec)
mysql> set session auto_increment_offset=1;
Query OK, 0 rows affected (0.00 sec)
mysql> show session variables like '%auto_incre%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)
mysql>  insert into t1 values (null),(null),(null),(null),(null),(null);
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> select * from t1;
+----+
| id |
+----+
|  1 |
|  3 |
|  5 |
|  7 |
|  9 |
| 11 |
+----+
6 rows in set (0.00 sec)
```

- Example 2

```
auto_increment_increment=2
auto_increment_offset=2
mysql> truncate t1;
Query OK, 0 rows affected (0.00 sec)
mysql>
mysql> set session auto_increment_increment=2;
Query OK, 0 rows affected (0.00 sec)
mysql> set session auto_increment_offset=2;
Query OK, 0 rows affected (0.00 sec)
mysql> show session variables like '%auto_incre%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 2     |
| auto_increment_offset    | 2     |
+--------------------------+-------+
2 rows in set (0.00 sec)
mysql>  insert into t1 values (null),(null),(null),(null),(null),(null);
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> select * from t1;
+----+
| id |
+----+
|  2 |
|  4 |
|  6 |
|  8 |
| 10 |
| 12 |
+----+
6 rows in set (0.00 sec)
```

- Example 3

```
auto_increment_increment=10
auto_increment_offset=5
mysql> truncate t1;
Query OK, 0 rows affected (0.00 sec)
mysql> set session auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)
mysql>  set session auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)
mysql> show session variables like '%auto_incre%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)
mysql> insert into t1 values (null),(null),(null),(null),(null),(null);
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
mysql> select * from t1;
+----+
| id |
+----+
|  5 |
| 15 |
| 25 |
| 35 |
| 45 |
| 55 |
+----+
6 rows in set (0.00 sec)
```

5.6.3. Change start values of auto-increment columns

You can specify the start value of an auto-increment column when you execute `CREATE TABLE` to create a table. You can also execute `ALTER TABLE table_name AUTO_INCREMENT=n` to change the start value of an AUTO_INCREMENT column. The specified value n may be smaller than the current value of the `AUTO_INCREMENT` column. If this occurs, the system does not report an error for the executed statement, but the value n does not take effect.

Examples

```
Mysql >create table t1(id int auto_increment primary key,name VARCHAR(10)) auto_increment=100;
Query OK, 0 rows affected (0.00 sec)
#Set the start value of the auto_increment field to 100 when you create the table.
mysql >alter table t1 auto_increment=50;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
#Execute the ALTER TABLE statement to change the start value of the auto_increment field in the t1 table to 50.
mysql >select * from t1;
+----+------+
| id | name |
+----+------+
| 50 | a    |
| 51 | a    |
| 52 | a    |
| 53 | a    |
| 54 | a    |
+----+------+
5 rows in set (0.00 sec)
mysql >alter table t1 auto_increment=10;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
#Execute the ALTER TABLE statement to change the start value of the auto-increment field in the t1 table to 10. The value 10 does not take effect because 10 is smaller than the current value of the auto_increment field.
mysql>show create table t1;
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Table | Create Table
|
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| t1    | CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=55 DEFAULT CHARSET=utf8 |
+-------+------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

5.6.4. LAST_INSERT_ID() function

This function returns the auto-increment field value that is latest inserted in the current session. If you insert multiple rows into the table in the latest operation, the LAST_INSERT_ID() function returns the auto-increment field value of the first row.

Examples

```
mysql>select LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                5 |
+------------------+
1 row in set (0.00 sec)
```

5.6.5. Limits

The following list describes the limits of auto-increment fields:

- The values of auto-increment fields must be unique. The values of auto-increment fields may not be continuous. For example, if you execute the `INSERT ...SELECT...` statement to insert data in batches, the values that are generated for an auto-increment field may not be continuous.
- You can execute a single INSERT statement on only one partition. If you execute a single INSERT statement across partitions, the system reports an error.

5.7. Functions

5.7.1. Overview

ApsaraDB for OceanBase supports date and time functions, string functions, type conversion functions, aggregate functions, flow control functions, mathematical functions, comparison functions, information functions, and other functions.

Expressions can be used in SQL statements and clauses, such as ORDER BY and HAVING clauses in SELECT statements, WHERE clauses in SELECT, DELETE, and UPDATE statements, and SET statements. You can define expressions by using literal values, column values, NULL values, functions, and operators.

In most cases, an expression that contains a NULL value returns a NULL value.

describes the functions that ApsaraDB for OceanBase supports.

Supported functions

Function type	Function description	Function name
Date and time functions	Return date and time information.	• CURRENT_TIME • CURTIME • CURRENT_TIMESTAMP • CURRENT_DATE • CURDATE • DATE_ADD • DATE_FORMAT • DATE_SUB • EXTRACT • NOW • STR_TO_DATE • TIME_TO_USEC • USEC_TO_TIME • UNIX_TIMESTAMP • DATEDIFF • TIMEDIFF • TIMESTAMPDIFF • PERIOD_DIFF • TO_DAYS • FROM_DAYS
String functions	Manipulate n-base numbers, strings, and expressions. For example, you can use a function of this type to retrieve the start position of a string or the number of returned strings.	• CONCAT • SUBSTRING • SUBSTR • TRIM • LENGTH • UPPER • LOWER • HEX • UNHEX • INT2IP • IP2INT • LIKE • REGEXP • REPEAT • SUBSTRING_INDEX • LOCATE • INSTR • REPLACE • FIELD • ELT
Type conversion functions	Convert values from a data type to another data type.	CAST

Function type	Function description	Function name
Aggregate functions	Perform calculations on a set of values and return a single value. In most cases, aggregate functions are used in conjunction with `GROUP BY` clauses of SELECT statements. If an aggregate function is used in conjunction with a `GROUP BY` clause, the aggregate functions return a single value for each group instead of the entire table.	• AVG • COUNT • MAX • MIN • SUM • GROUP_CONCAT
Mathematical functions	Perform mathematical calculations based on numeric expressions.	• ROUND • CEIL • FLOOR • ABS • NEG • SIGN • CONV • MOD • POW • POWER
Comparison functions	Compare input values.	• GREATEST • LEAST • ISNULL
Flow control functions	Control flows.	• CASE • IF • IFNULL • NULLIF
Information functions	Retrieve dynamic database information.	• FOUND_ROWS • LAST_INSERT_ID
Other functions	The other functions.	• COALESCE • NVL • DECODE

5.7.2. Date and time functions

The functions of this type return date and time information.

CURRENT_TIME() and CURRENT_TIMESTAMP()

Notice You can pass a number that ranges from 0 to 6 into a CURRENT_TIME() or a CURRENT_TIMESTAMP() function. The number specifies the precision of the fractional second part.

The following list describes the notes on CURRENT_TIME() and CURRENT_TIMESTAMP() functions:

- `CURRENT_TIME()` and `CURRENT_TIMESTAMP()` functions return the current time of the system. The two functions return the time in different formats.
- `CURRENT_TIME()` returns the current time in the `HH:MI:SS` format. The returned value excludes the date part.
- `CURRENT_TIMESTAMP()` returns the current date and time in the `YYYY-MM-DD HH:MI:SS` format.

Examples

```
root@test 04:09:10>SELECT CURRENT_TIME(), CURRENT_TIMESTAMP();
+----------------+---------------------+
| CURRENT_TIME() | CURRENT_TIMESTAMP() |
+----------------+---------------------+
| 16:24:09       | 2014-10-31 16:24:09 |
+----------------+---------------------+
1 row in set (0.00 sec)
root@(none) 09:59:30>select CURRENT_TIME(1);
#You can specify the precision of the fractional second part.
+-----------------+
| CURRENT_TIME(1) |
+-----------------+
| 09:59:32.4      |
+-----------------+
1 row in set (0.00 sec)
root@(none) 09:59:32>select CURRENT_TIME(7);
ERROR 1426 (42000): Too big precision 7 specified for column 'curtime'. Maximum is 6.
#The value that you can specify for the precision ranges from 0 to 6. If the value that you specify is not in this range,
the system reports the 1426 (42000) error.
```

Note If CURRENT_TIME() or CURRENT_TIMESTAMP() is run for different transactions in the same session, the returned results may not be incremental in chronological order. This is because the time of one server is different from that of another server.

CURTIME()

`CURTIME()` is a synonym for `CURRENT_TIME()` or `CURRENT_TIME` .

```
mysql> select curtime(), current_time(), current_time;
+-----------+----------------+--------------+
| curtime() | current_time() | current_time |
+-----------+----------------+--------------+
| 14:45:37  | 14:45:37       | 14:45:37     |
+-----------+----------------+--------------+
1 row in set (0.00 sec)
```

CURRENT_DATE()

`CURRENT_DATE()` returns the current date in the `'YYYY-MM-DD'` or `YYYYMMDD` format.

The returned date format varies based on the date format that is specified in the function. If the date format in the function is a string, the returned date is a string. If the date format in the function is a numeric value, the returned date is a numeric value.

Examples

```
mysql> select current_date, current_date+5;
+--------------+----------------+
| current_date | current_date+5 |
+--------------+----------------+
| 2015-08-27   |       20150832 |
+--------------+----------------+
1 row in set (0.00 sec)
```

CURDATE()

`CURDATE()` is a synonym for `CURRENT_DATE()` and `CURRENT_DATE` .

```
mysql> select curdate(), current_date(), current_date;
+------------+----------------+--------------+
| curdate()  | current_date() | current_date |
+------------+----------------+--------------+
| 2015-08-27 | 2015-08-27     | 2015-08-27   |
+------------+----------------+--------------+
1 row in set (0.00 sec)
```

DATE_ADD(*date*, INTERVAL *exprunit*)

This function adds a specified interval to a date. The *date* parameter specifies the start date to which an interval is added. The *expr* parameter specifies the interval to be added. The values of *expr* can be negative. Based on the internal configurations and the time zone, the operating system determines whether to use daylight saving time (DST) for the `DATE_ADD()` function.

The following list describes the notes on the DATE_ADD() function:

- The data types of *date* values must be string or date and time data types, such as DATETIME and TIMESTAMP. The other data types are not supported. If you use string data types, the strings must represent time values.
- The valid format of the *date* parameter is `YYYY-MM-DD HH:MM:SS.SSSSSS`.

  MySQL supports non-strict syntax for parsing date strings. If a string contains digits and non-digit characters, MySQL parses only the digits as time values and assigns the values to the year, month, and day parts in sequence. For example, the `Ywwe1990d07 09,12:45-08&900` string represents the same time as `1990-07-09 12:45:08.900`. Strict syntax is applied in ApsaraDB for OceanBase. The system reports errors for invalid date types. For example, if you set the date parameter to the invalid date `ABC`, the system reports an error.
- You must specify the date part in a *date* string. The time part in a date string is optional. If you do not specify the time part, the default values are used. For example, `1990-07-09` is valid. By default, the time values that follow 1990-07-09 are set to 0 values: `1990-07-09 00:00:00.000000`. `1990-07` and `1990` are invalid date values.
- The DATE_ADD() function cannot parse TIMESTAMP strings, such as `990309`.
- In ApsaraDB for OceanBase, you can use the results of invoking other system functions as the values of *date*.
- ApsaraDB for OceanBase does not support fuzzy match for two-digit years. For example, MySQL interprets the 12 year as the 2012 year, but ApsaraDB for OceanBase interprets the 12 year as the year 12.
- The values of *expr* can be negative. If you specify a negative value for the expr parameter, the function subtracts the corresponding interval from the start date. You can use the results of invoking system functions as expr values. All the results are processed as strings.
- The *unit* parameter specifies the unit of the interval that is to be added or subtracted. The valid values of the parameter are MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, SECOND_MICROSECOND, MINUTE_MICROSECOND, MINUTE_SECOND, HOUR_MICROSECOND, HOUR_SECOND, HOUR_MINUTE, DAY_MICROSECOND, DAY_SECOND, DAY_MINUTE, DAY_HOUR, and YEAR_MONTH. QUARTER represents quarters.
- If the *unit* value is a compound unit, you must enclose *expr* values in single quotation marks (').

Note In MySQL command-line clients, if a single line is excessively long and compromises read experience, you can add `\G` to the end of the SELECT statement. \G is used to align the query results in a vertical way.

Examples

```
mysql> SELECT DATE_ADD(now(), INTERVAL 5 DAY),
->     DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL 5 SECOND),
->     DATE_ADD('2014-01-10', INTERVAL 5 MINUTE),
->     DATE_ADD('2014-01-10', INTERVAL 5 HOUR),
->     DATE_ADD('2014-01-10', INTERVAL 5 DAY),
->     DATE_ADD('2014-01-10', INTERVAL 5 WEEK),
->     DATE_ADD('2014-01-10', INTERVAL 5 MONTH),
->     DATE_ADD('2014-01-10', INTERVAL 5 QUARTER),
->     DATE_ADD('2014-01-10', INTERVAL 5 YEAR),
->     DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND),
->     DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE),
->     DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND),
->     DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND),
->     DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE),
->     DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR),
->     DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH) \G
*************************** 1. row ***************************
                                      DATE_ADD(now(), INTERVAL 5 DAY): 2016-03-19 13:56:45
                       DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND): 2014-01-10 00:00:00.000005
                            DATE_ADD('2014-01-10', INTERVAL 5 SECOND): 2014-01-10 00:00:05
                            DATE_ADD('2014-01-10', INTERVAL 5 MINUTE): 2014-01-10 00:05:00
                              DATE_ADD('2014-01-10', INTERVAL 5 HOUR): 2014-01-10 05:00:00
                               DATE_ADD('2014-01-10', INTERVAL 5 DAY): 2014-01-15
                              DATE_ADD('2014-01-10', INTERVAL 5 WEEK): 2014-02-14 00:00:00
                             DATE_ADD('2014-01-10', INTERVAL 5 MONTH): 2014-06-10
                           DATE_ADD('2014-01-10', INTERVAL 5 QUARTER): 2015-04-10 00:00:00
                              DATE_ADD('2014-01-10', INTERVAL 5 YEAR): 2019-01-10
       DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND): 2014-01-10 00:00:05.000005
   DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND): 2014-01-10 00:05:05.000005
               DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND): 2014-01-10 00:05:05
  DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND): 2014-01-10 05:05:05.000005
              DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND): 2014-01-10 05:05:05
                 DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE): 2014-01-10 05:05:00
DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND): 2014-01-11 05:05:05.000005
            DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND): 2014-01-11 05:05:05
               DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE): 2014-01-11 05:05:00
                    DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR): 2014-01-11 05:00:00
                   DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH): 2015-02-10
1 row in set (0.01 sec)
```

In the DATE_ADD() function, you can place addition (+) or subtraction operators (-) before INTERVARL, as shown in the following examples:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

For addition operations, `INTERVAL expr unit` can be placed on the right of the *date* parameter. The value of the date parameter can be a date value or a date and time value. For subtraction operations, `INTERVAL expr unit` can be placed only on the right of the *date* parameter. If INTERVAL expr unit is placed on the left of the date parameter, a date value is subtracted from an interval. This computing process does not return a valid result.

Examples

```
mysql> SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND ,'2005-01-01' - INTERVAL 1 SECOND\G;
*************************** 1. row ***************************
'2008-12-31 23:59:59' + INTERVAL 1 SECOND: 2009-01-01 00:00:00
     '2005-01-01' - INTERVAL 1 SECOND: 2004-12-31 23:59:59
1 row in set (0.00 sec)
```

DATE_FORMAT(*date*, *format*)

`DATE_FORMAT()` is an inverse function of `STR_TO_DATE()`. `DATE_FORMAT()` converts a specified value of the *date* parameter into a time string based on the specified *format*.

The *date* parameter specifies the time value to be converted. The *date* values can be date and time values of the STRING data type. For more information, see the description of the date parameter in `DATE_ADD()` .

describes the *format* values that can be used in a format string.

Values of the format parameter in DATE_FORMAT()

Format	Description	Returned value range or format
%a	The abbreviation of the day of the week.	Sun..Sat
%b	The abbreviation of the month.	Jan to Dec.
%c	The month of the numeric type.	1 to 12.
%D	The day that ends with an ordinal indicator: st, nd, rd, or th.	1st to 31st.
%d	The day of the numeric type in the month.	01 to 31.
%e	The day of the numeric type in the month.	1 to 31.
%f	The microsecond.	000000 to 999999.
%H	The hour.	00 to 23.
%h	The hour.	01 to 12.
%I	The hour.	01 to 12.
%i	The minute.	00 to 59.
%j	The sequence number of the day in the year.	000 to 366.
%k	The hour.	0 to 23.
%l	The hour.	01 to 12.
%M	The name of the month.	January to December.
%m	The month of the numeric type.	01 to 12.
%p	The morning or the afternoon.	AM and PM.
%r	The time in the 12-hour clock.	hh:mm:ss AM/PM
%S	The seconds.	00 to 59.
%s	The seconds.	00 to 59.
%T	The time in the 24-hour clock.	hh:mm:ss
%U	The sequence number of the week in the year when Sunday is the first day of each week.	00 to 53.
%u	The sequence number of the week in the year when Monday is the first day of each week.	00 to 53.
%V	The sequence number of the week in the year when Sunday is the first day of each week. This option is used in conjunction with %X. Notes: A date may fall within the first week of a year or the last week of the previous year. If this occurs, the sequence number of the week varies based on the format values. Wednesday, January 01, 2014 is used as an example to explain this rule. If the format value is %U or %u, the date falls within week 00 in 2014. If the format value is %V or %v, the date falls within week 52 in 2013.	01 to 53.

Format	Description	Returned value range or format
%v	The sequence number of the week in the year when Monday is the first day of each week. This option is used in conjunction with %x.	01 to 53.
%W	The day of the week.	Sunday to Saturday.
%w	The sequence number of the day in the week.	`0 to 6. The value 0 represents Sunday. The value 6 represents Saturday. Similar rules apply to values 1 to 5.`
%X	The year of the week when Sunday is the first day of each week. The year is represented by a number that consists of four digits. %X is used in conjunction with %V.	-
%x	The year of the week when Monday is the first day of each week. The year is represented by a number that consists of four digits. %x is used in conjunction with %v.	-
%Y	The year in the four-digit format.	-
%y	The year in the two-digit format.	-
%%	The literal character %.	-

Examples

```
mysql> SELECT DATE_FORMAT('2014-01-01', '%Y-%M-%d'),
DATE_FORMAT('2014-01-01', '%X-%V'),DATE_FORMAT('2014-01-01', '%U') \G
*************************** 1. row ***************************
DATE_FORMAT('2014-01-01', '%Y-%M-%d'): 2014-January-01
DATE_FORMAT('2014-01-01', '%X-%V'): 2013-52
DATE_FORMAT('2014-01-01', '%U'): 00
1 row in set (0.00 sec)
```

DATE_SUB(*date*, INTERVAL *expr unit*)

This function subtracts a specified interval from a date. The *date* parameter specifies the start date from which an interval is subtracted. The *expr* parameter specifies the interval to be subtracted from the start date. The values of *expr* can be negative. If you specify a negative value for the expr parameter, the function adds the corresponding interval to the start date.

For more information about the parameter description, see `DATE_ADD()` .

Examples

```
mysql> SELECT DATE_SUB('2014-01-10', INTERVAL 5 HOUR),
DATE_SUB('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND)\G;
*************************** 1. row ***************************
DATE_SUB('2014-01-10', INTERVAL 5 HOUR): 2014-01-09 19:00:00
DATE_SUB('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND): 2014-01-09 18:54:54.999995
1 row in set (0.00 sec)
```

EXTRACT(*unit* FROM *date*)

This function extracts parts from a date based on the specified units. The *date* parameter specifies the date. The *unit* parameter specifies the units.

For more information about the parameter description, see `DATE_ADD()` .

- The data type of the results that the EXTRACT function returns is BIGINT.
- For a single unit, such as microseconds to years, the EXTRACT function returns the corresponding integer.
- If the *unit* value is WEEK, the function returns the sequence number of the week in the year of the date that you specify in the *date* expression. ApsaraDB for OceanBase considers the first Sunday of the year as the beginning of the first week in the year. If the first Sunday of a year is not January 1, January 1 and the days between the first Sunday and January 1 are in week 0. For example, the first Sunday in 2013 is January 6. Therefore, `SELECT EXTRACT(WEEK FROM '2013-01-01')` returns 0 and `SELECT EXTRACT(WEEK FROM '2013-01-06')` returns 1.

- For compound units such as SECOND_MICROSECOND, ApsaraDB for OceanBase combines the values for the specified units and returns a single value. For example, `SELECT EXTRACT(YEAR_MONTH FROM '2012-03-09')` returns `201203`.

```
mysql> SELECT EXTRACT(WEEK FROM '2013-01-01'),
EXTRACT(WEEK FROM '2013-01-06'),
EXTRACT(YEAR_MONTH FROM '2012-03-09'),
EXTRACT(DAY FROM NOW())\G;
*************************** 1. row ***************************
EXTRACT(WEEK FROM '2013-01-01'): 0
EXTRACT(WEEK FROM '2013-01-06'): 1
EXTRACT(YEAR_MONTH FROM '2012-03-09'): 201203
EXTRACT(DAY FROM NOW()): 18
1 row in set (0.00 sec)
```

NOW()

This function returns the current system time that is accurate to seconds. The time format is `YYYY-MM-DD HH:MI:SS`.

Notice You can pass a number that ranges from 0 to 6 into a NOW() function. The number specifies the precision of the fractional second part. By default, `NOW()` is equivalent to `NOW(0)`.

Examples

```
mysql> SELECT NOW();
+-----------------------------+
| NOW()                       |
+-----------------------------+
| 2014-02-17 11:46:15|
+-----------------------------+
1 row in set (0.00 sec)
root@(none) 09:52:46>select now(0);
+----------------------+
| now(0)               |
+----------------------+
| 2014-11-03 09:55:01 |
+----------------------+
1 row in set (0.00 sec)
root@(none) 09:55:01>select now(1);
+------------------------+
| now(1)                 |
+------------------------+
| 2014-11-03 09:55:04.2 |
+------------------------+
1 row in set (0.00 sec)
root@(none) 09:55:04>select now(2);
+-------------------------+
| now(2)                  |
+-------------------------+
| 2014-11-03 09:55:06.57 |
+-------------------------+
1 row in set (0.00 sec)
root@(none) 09:55:06>select now(3);
+--------------------------+
| now(3)                   |
+--------------------------+
| 2014-11-03 09:55:09.576 |
+--------------------------+
1 row in set (0.00 sec)
root@(none) 09:55:09>select now(7);
ERROR 1426 (42000): Too big precision 7 specified for column 'now'. Maximum is 6.
```

STR_TO_DATE(*str*,*format*)

This function converts a string that is specified by *str* into a value of the DATEITME, DATE, or TIME type based on the *format* string. If the format string contains date and time parts, `STR_TO_DATE()` returns a DATETIME value. If the format string contains only the date part, this function returns a DATE value. If the format string contains only the time part, this function returns a TIME value.

The format of the DATE, TIME, or DATETIME value in *str* must be specified in *format*. If the *str* value contains an invalid DATE, TIME, or DATETIME value, STR_TO_DATE() returns NULL. In this case, the system returns a warning message for the invalid value.

describes the *format* values that can be used in a format string.

Values of the format parameter in STR_TO_DATE()

Format	Description	Returned value range or format
%b	The abbreviation of the month.	Jan to Dec
%c	The month of the numeric type.	1 to 12
%D	The day that ends with an ordinal indicator: st, nd, rd, or th.	1st to 31st
%d	The day of the numeric type in the month.	01 to 31
%e	The day of the numeric type in the month.	1 to 31
%f	The microsecond.	000000 to 999999
%H	The hour.	00 to 23
%h	The hour.	01 to 12
%I	The hour.	01 to 12
%i	The minute.	00 to 59
%k	The hour.	0 to 23
%l	Hours.	01 to 12
%M	The name of the month.	January to December
%m	The month of the numeric type.	01 to 12
%p	The morning or the afternoon.	AM and PM
%r	The time in the 12-hour clock.	hh:mm:ss AM/PM
%S	The seconds.	00 to 59
%s	The seconds.	00 to 59
%T	The time in the 24-hour clock.	hh:mm:ss
%Y	The year in the four-digit format.	-

Examples

```
mysql> SELECT STR_TO_DATE('2014-Jan-1st 5:5:5', '%Y-%b-%D');
+-----------------------------------------------------+
| STR_TO_DATE('2014-Jan-1st 5:5:5 pm', '%Y-%b-%D') |
+-----------------------------------------------------+
| 2014-01-01 05:05:05                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

TIME_TO_USEC(*date*)

This function converts the internal time of ApsaraDB for OceanBase into the number of microseconds. The returned result of this function indicates the number of microseconds from `1970-01-01 00:00:00` to the time that is specified by *date*. The returned result is the UTC time and uses the UTC+0 time zone.

The following list describes the notes on the TIME_TO_USEC(date) function:

- The *date* parameter specifies the date that is to be converted into the number of microseconds. The date uses the time zone that is specified in the system. The value of the date parameter is a string of the TIMESTAMP or TIME data type.
- The `TIME_TO_USEC` function can use the results of invoking other functions as the values of the date parameter. Note that the results must be strings of the TIMESTAMP or TIME data type.
- The returned values of the TIME_TO_USEC function are measured in microseconds. The data type of the returned values is INT.

Examples

```
mysql> SELECT TIME_TO_USEC('2014-03-25'), TIME_TO_USEC(now(6));
+----------------------------+---------------------+
| TIME_TO_USEC('2014-03-25') | TIME_TO_USEC(now(6)) |
+----------------------------+---------------------+
|           1395676800000000 |    1395735415207794 |
+----------------------------+---------------------+
1 row in set (0.00 sec)
```

USEC_TO_TIME(*usec*)

This function is an inverse function of TIME_TO_USEC (*date*). The USEC_TO_TIME(usec) function adds the value of *usec* to `1970-01-01 00:00:00` and returns a result that uses the required time zone. For example, if you invoke the `USEC_TO_TIME(1)` function in the UTC+8 time zone, the function returns `1970-01-01 08:00:01`.

The following list describes the notes on the USEC_TO_TIME(usec) function:

- The *usec* parameter specifies the number of microseconds.
- This function returns a value of the TIMESTAMP data type.

Examples

```
mysql> SELECT USEC_TO_TIME(1);
+----------------------------+
| USEC_TO_TIME(1)            |
+----------------------------+
| 1970-01-01 08:00:00.000001 |
+----------------------------+
1 row in set (0.00 sec)
```

UNIX_TIMESTAMP(),UNIX_TIMESTAMP(*date*)

If you do not pass arguments into the UNIX_TIMESTAMP() function, the function returns a `UNIX timestamp`. A UNIX timestamp is the number of seconds that have elapsed since `'1970-01-01 00:00:00' GMT`. The returned result is an unsigned integer.

If you pass a *date* value into the UNIX_TIMESTAMP() function, the function returns the value as the number of seconds that have elapsed since `'1970-01-01 00:00:00' GMT`. The values of the *date* parameter can be DATE strings, DATETIME strings, TIMESTAMP strings, or numbers in the YYMMDD or YYYYMMDD format. Note that the numbers represent the local time.

Examples

```
mysql> SELECT UNIX_TIMESTAMP();
+------------------+
| UNIX_TIMESTAMP() |
+------------------+
|       1427176668 |
+------------------+
1 row in set (0.00 sec)
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00')
+---------------------------------------+
| UNIX_TIMESTAMP('1997-10-04 22:23:00') |
+---------------------------------------+
|                             875974980 |
+---------------------------------------+
1 row in set (0.00 sec)
```

If the values in the `TIMESTAMP` column are used as the values of the date parameter for the `UNIX_TIMESTAMP` function, the function returns internal timestamps.

If you pass a date that does not fall within the basic date ranges into the UNIX_TIMESTAMP() function, the function returns 0. The function checks the specified date values against only the basic date ranges. In the basic date ranges, year values range from 1970 to 2037, month values range from 01 to 12, and day values range from 01 to 31.

DATEDIFF(*expr1*,*expr2*)

The DATEDIFF() function returns the number of days between the start date that is specified by *expr1* and the end date that is specified by *expr2*. *expr1* and *expr2* are date or date and time expressions. Only the date parts in the expressions are used to calculate the returned result.

You must specify two arguments for this function. If the number of the specified arguments is not two, the system reports an error.

Examples

```
mysql> select datediff('2015-06-19','1994-12-17'), datediff('2015-06-19','1998-06-27 10:10:10'), datediff(now(), '2014-01
-02')\G;
*************************** 1. row ***************************
         datediff('2015-06-19','1994-12-17'): 7489
datediff('2015-06-19','1998-06-27 10:10:10'): 6201
               datediff(now(), '2014-01-02'): 533
1 row in set (0.00 sec)
```

TIMEDIFF(*expr1*,*expr2*)

The `TIMEDIFF()` function returns the time interval between the start time that is specified by *expr1* and the end time that is specified by *expr2*. *expr1* and *expr2* are time or date and time expressions. The data types of expr1 and expr2 values must be the same.

The result that is returned by the `TIMEDIFF()` function must fall within the valid range of time values. You can also use the `TIMESTAMPDIFF()` and `UNIX_TIMESTAMP()` functions. These functions return integer values.

Examples

```
mysql>  select timediff(now(), '2017-06-06 11:11:22'), timediff('2015-06-06 12:12:12', '2014-06-05 11:11:11')\G;
*************************** 1. row ***************************
                timediff(now(), '2017-06-06 11:11:22'): 315:00:15
timediff('2015-06-06 12:12:12', '2014-06-05 11:11:11'): 838:59:59
1 row in set, 1 warning (0.00 sec)
```

TIMESTAMPDIFF(*unit*,*datetime_expr1*,*datetime_expr2*)

This function returns the difference between the value that is specified by *datetime_expr1* and the value that is specified by *datetime_expr2*. The returned result is an integer. The *unit* parameter specifies the unit of the returned result.

Valid values of the *unit* parameter are `MICROSECOND (microsecond)` , SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, and YEAR.

Examples

```
mysql> select timestampdiff(second,now(), '2011-01-01 11:11:11'),  timestampdiff(second, '2011-01-01 11:11:11', now())\G;
*************************** 1. row ***************************
 timestampdiff(second,now(), '2011-01-01 11:11:11'): -140843995
timestampdiff(second, '2011-01-01 11:11:11', now()): 140843995
1 row in set (0.00 sec)
```

PERIOD_DIFF(*p1*,*p2*)

This function returns the number of months between the *p1* period and the *p2* period. *p1* and *p2* values must use the YYMM or YYYYMM format.

Notice *p1* and *p2* values are not date values.

Examples

```
mysql> select period_diff(20150702, 20790503), period_diff(150702, 790503);
+--------------------------------+----------------------------+
| period_diff(20150702, 20790503) | period_diff(150702, 790503) |
+--------------------------------+----------------------------+
|                         -76777 |                     -76777 |
+--------------------------------+----------------------------+
1 row in set (0.00 sec)
```

TO_DAYS(date)

This function returns the number of days between the zero year and the date that is specified by the date parameter.

Examples

```
 mysql> SELECT TO_DAYS('2015-11-04'), TO_DAYS('20151104');
+-----------------------+---------------------+
| TO_DAYS('2015-11-04') | TO_DAYS('20151104') |
+-----------------------+---------------------+
|                736271 |              736271 |
+-----------------------+---------------------+
1 row in set (0.00 sec)
```

TO_DAYS() is not used for the dates that are earlier than the year 1582 when the Gregorian calendar was first introduced. This is because some days were lost when the Julian calendar was switched to the Gregorian calendar. The lost days are not considered in this function.

Two-digit year values in dates are converted into four-digit year values. For example, `'2015-11-04'` and `'15-11-04'` represent the same date.

Examples

```
mysql> SELECT TO_DAYS('2015-11-04'), TO_DAYS('151104');
+-----------------------+-------------------+
| TO_DAYS('2015-11-04') | TO_DAYS('151104') |
+-----------------------+-------------------+
|                736271 |            736271 |
+-----------------------+-------------------+
1 row in set (0.00 sec)
```

The system considers the zero date `'0000-00-00'` as an invalid date. The TO_DAYS() function returns the following result for the zero date:

```
mysql> SELECT TO_DAYS('0000-00-00');
+-----------------------+
| TO_DAYS('0000-00-00') |
+-----------------------+
|                  NULL |
+-----------------------+
1 row in set, 1 warning (0.01 sec)
mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)
```

FROM_DAYS(N)

The function returns a DATE value based on the number of days that you specify by N.

Examples

```
mysql> SELECT FROM_DAYS(736271), FROM_DAYS(700000);
+-------------------+-------------------+
| FROM_DAYS(736271) | FROM_DAYS(700000) |
+-------------------+-------------------+
| 2015-11-04        | 1916-07-15        |
+-------------------+-------------------+
1 row in set (0.00 sec)
```

Note Use caution when you use the FROM_DAYS() function to process ancient dates. This function is not used to process the dates that are earlier than the 1582 year when the Gregorian calendar was first introduced.

FROM_UNIXTIME(unix_timestamp[,*format*])

```
FROM_UNIXTIME(unix_timestamp) , FROM_UNIXTIME(unix_timestamp,format)
```

This function converts a unix_timestamp value into a value in the `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS` format. The format depends on whether the function uses strings or numbers as arguments.

If you set the *format* parameter to a format string, the function returns the result based on the specified format. The format values in this function support those for the format input parameter in the DATE_FORMAT() function.

describes the *format* values that can be used in a format string.

Values of the format parameter in FROM_UNIXTIME()

Format	Description
%a	The abbreviation of the day of the week. The returned values range from Sun to Sat.
%b	The abbreviation of the day of the week. The returned values range from Jan to Dec.
%c	The month of the numeric data type. The returned values range from 0 to 12.
%D	The day that ends with an ordinal indicator: st, nd, rd, or th. The returned values range from 0th to 31st.
%d	The day of the numeric data type in the month. The returned values range from 00 to 31.
%e	The day of the numeric data type in the month. The returned values range from 0 to 31.
%f	The microsecond. The returned values range from 000000 to 999999.
%H	The hour. The returned values range from 00 to 23.
%h	The hour. The returned values range from 01 to 12.
%I	The hour. The returned values range from 01 to 12.
%i	The minute of the numeric data type. The returned values range from 00 to 59.
%j	The number of days in the year. The returned values range from 001 to 366.
%k	The hour. The returned values range from 0 to 23.
%l	The hour. The returned values range from 1 to 12.
%M	The name of the month. The returned values range from January to December.
%m	The month of the numeric data type. The returned values range from 00 to 12.
%p	The morning or the afternoon. The returned value for the morning is AM. The returned value for the afternoon is PM.
%r	The time in the 12-hour clock. The returned value uses the hh:mm:ss format and ends with AM or PM. hh represents hours, mm represents minutes, and ss represents seconds.
%S	The seconds. The returned values range from 00 to 59.
%s	The seconds. The returned values range from 00 to 59.
%T	The time in the 24-hour clock. The returned value uses the hh:mm:ss format.
%U	The sequence number of the week in the year when Sunday is the first day of each week. The returned values range from 00 to 53.
%u	The sequence number of the week in the year when Monday is the first day of each week. The returned values range from 00 to 53.

Format	Description
%V	The sequence number of the week in the year when Sunday is the first day of each week. This option is used in conjunction with %X. The returned values range from 01 to 53.
%v	The sequence number of the week in the year when Monday is the first day of each week. This option is used in conjunction with %x. The returned values range from 01 to 53.
%W	The day of the week. The returned values range from Sunday to Saturday.
%w	The day of the week. The returned values range from 0 to 6. The value 0 represents Sunday. The value 6 represents Saturday. Similar rules apply to numbers 1 to 5.
%X	The year of the week when Sunday is the first day of each week. The year is represented by a number that consists of four digits. This option is used in conjunction with %V.
%x	The year of the week when Monday is the first day of each week. The year is represented by a number that consists of four digits. This option is used in conjunction with %v.
%Y	The year of the numeric data type. The year is represented by a number that consists of four digits.
%y	The year of the numeric data type. The year is represented by a number that consists of two digits.
%%	The literal character %.

All the other characters are copied into the returned result and do not need to be interpreted.

Note The % characters must be placed before the format values.

Examples

```
mysql> SELECT FROM_UNIXTIME(875996580);
+--------------------------+
| FROM_UNIXTIME(875996580) |
+--------------------------+
| 1997-10-05 04:23:00      |
+--------------------------+
1 row in set (0.00 sec)
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
+------------------------------+
| FROM_UNIXTIME(875996580) + 0 |
+------------------------------+
|               19971005042300 |
+------------------------------+
1 row in set (0.00 sec)
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x');
+---------------------------------------------------------+
| FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x') |
+---------------------------------------------------------+
| 2016 6th January 10:18:22 2016                          |
+---------------------------------------------------------+
1 row in set (0.01 sec)
```

5.7.3. String functions

Note ApsaraDB for OceanBase supports only the UTF8MB4 character set.

CONCAT(*str1,..., strN*)

This function concatenates one or more strings into a single string. Each input value in CONCAT functions must be a string or a NULL value. Otherwise, the system reports an error. If the concentration is successful, the function returns the concatenated string. If an argument is NULL, the function returns NULL.

Note If the data types of the str parameter are numeric data types, the system converts numeric values into strings in an implicit way.

Examples

```
mysql> select concat('test'), concat('test','OceanBase'), concat('test', 'OceanBase', '1.0'), concat('test','OceanBase','
1.0', NULL)\G;
*************************** 1. row ***************************
                        concat('test'): test
            concat('test','OceanBase'): testOceanBase
    concat('test', 'OceanBase', '1.0'): testOceanBase1.0
concat('test','OceanBase','1.0', NULL): NULL
1 row in set (0.00 sec)
```

Errors

- If the syntax of the function is invalid, the system reports the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .
- If the format of a data type is invalid, the system reports the following error: `ERROR 1054 (42S22): Unknown column` . For example, if you do not enclose a string in double quotations marks ("), the system reports the error.

SUBSTRING

```
SUBSTRING(str,pos)
SUBSTRING(str FROM pos)
SUBSTRING(str,pos,len)
SUBSTRING(str FROM pos FOR len)
```

This function has the same semantics as the SUBSTR function.

SUBSTR

```
SUBSTR(str,pos,len)
SUBSTR(str,pos)
SUBSTR(str FROM pos)
SUBSTR (str FROM pos FOR len)
```

This function extracts a substring from a specified string. The *pos* parameter specifies the start position of the returned substring. The *len* parameter specifies the length of the returned substring. To use the standard SQL syntax to specify the start position, specify the pos parameter in the FROM clause.

- The values of *str* must be strings. The values of *pos* and *len* must be integers. If an argument is NULL, the function returns NULL.
- The Chinese characters that you specify in the values of *str* are identified as byte streams.
- If you do not specify the *len* parameter in the function, the returned substring starts from the position *pos* and ends with the last character of the source string.
- If you specify the *pos* parameter as a negative value, the function determines the start position of the returned substring based on the right-of-left order. If you specify the *pos* parameter as 0, the system considers 0 as 1. This means that the returned substring starts with the first character of the source string.
- If the *len* value is smaller than or equal to 0 or no character exists at the *pos* position, the returned result is an empty string.

```
mysql> SELECT SUBSTR('abcdefg',3), SUBSTR('abcdefg',3,2), SUBSTR('abcdefg',-3), SUBSTR('abcdefg',3,-2), SUBSTR('abcdefg
' from -4 for 2)\G;
*************************** 1. row ***************************
SUBSTR('abcdefg',3): cdefg
SUBSTR('abcdefg',3,2): cd
SUBSTR('abcdefg',-3): efg
SUBSTR('abcdefg',3,-2):
SUBSTR('abcdefg' from -4 for 2): de
1 row in set (0.00 sec)
```

Errors

If the syntax is invalid, the system reports the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .

TRIM([[{BOTH | LEADING | TRAILING}] [remstr] FROM] str)

This function can remove leading and trailing characters from a specified string at a time. This function can also remove only leading or trailing characters from a sting at a time.

- The values of *remstr* and *str* parameters must be strings or NULL. If an argument is NULL, the function returns NULL.
- If you do not specify BOTH, LEADIN, or TRAILING, BOTH is used by default.
- The *remstr* parameter is optional. If you do not specify this parameter, the function removes spaces from a specified string.

```
mysql> SELECT TRIM(' bar '),
TRIM(LEADING 'x' FROM 'xxxbarxxx'),
TRIM(BOTH 'x' FROM 'xxxbarxxx'),
TRIM(TRAILING 'x' FROM 'xxxbarxxx')\G;
*************************** 1. row ***************************
TRIM(' bar '): bar
TRIM(LEADING 'x' FROM 'xxxbarxxx'): barxxx
TRIM(BOTH 'x' FROM 'xxxbarxxx'): bar
TRIM(TRAILING 'x' FROM 'xxxbarxxx'): xxxbar
1 row in set (0.00 sec)
```

LENGTH(*str*)

This function returns the length of a specified string. The returned length is measured in bytes. The values of the input parameter must be strings or NULL. Otherwise, the system reports an error. If the operation is successful, the function returns an integer of the INT data type. The returned integer indicates the string length. If an argument is NULL, the function returns NULL.

If the data types of the *str* parameter are numeric data types, the system converts numeric values into strings in an implicit way.

```
mysql> SELECT LENGTH('text');
+----------------+
| LENGTH('text') |
+----------------+
|              4 |
+----------------+
1 row in set (0.00 sec)
mysql> select length(-1.23);
+---------------+
| length(-1.23) |
+---------------+
|             5 |
+---------------+
1 row in set (0.00 sec)
mysql> select length(1233e);
mysql> select length(1233e);
ERROR 1054 (42S22): Unknown column '1233e' in 'field list'
```

Errors

If the data type of the str parameter is invalid, the system reports the following error: `ERROR 1054 (42S22): Unknown column 'XXXX' in 'field list'` .

UPPER(*str*)

This function converts a specified string into uppercase characters. The values of the str parameter must be strings. If the str value is NULL, the function returns NULL.

If the data types of the *str* parameter are numeric data types, the system converts numeric values into strings in an implicit way.

The byte range of the Chinese character set does not overlap with that of the ASCII character set. Therefore, you can specify Chinese characters in the input values of the UPPER function.

```
mysql> SELECT UPPER ('Hello, OceanBase!') ;
+-----------------------------+
| UPPER('Hello, OceanBase!')    |
+-----------------------------+
| HELLO, OCEANBASE!             |
+-----------------------------+
1 row in set (0.00 sec)
mysql> select upper(e);
ERROR 1054 (42S22): Unknown column 'e' in 'field list'
mysql> select upper(1.235.) ;
ERROR 1064 (42000): You have an error in your SQL syntax;
```

Errors

- If the data type of the str parameter is invalid, the system reports the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .
- If you do not use double quotation marks (`''`) to enclose a string, the system reports the following error: `ERROR 1054 (42S22): Unknown column 'XX' in 'field list'` . This error is different from the following error: `ERROR 1166 (42703): Unknown column Name` that ApsaraDB for OceanBase reports.

LOWER(*str*)

This function converts a specified string into lowercase characters. The values of the str parameter must be strings. If the str value is NULL, the function returns NULL.

If the data types of the *str* parameter are numeric data types, the system converts numeric values into strings in an implicit way.

The byte range of the Chinese character set does not overlap with that of the ASCII character set. Therefore, you can specify Chinese characters in the input values of the LOWER function.

Examples

```
mysql> SELECT LOWER ('Hello, OceanBase!') ;
+-----------------------------+
| LOWER('Hello, OceanBase!')    |
+-----------------------------+
| hello, oceanbase!             |
+-----------------------------+
1 row in set (0.00 sec)
mysql> select lower(1.23) ;
+-------------+
| lower(1.23) |
+-------------+
| 1.23        |
+-------------+
1 row in set (0.00 sec)
mysql> select lower(1.23h);
ERROR 1583 (42000): Incorrect parameters in the call to native function 'lower'
mysql> select lower(1.23e);
ERROR 1582 (42000): Incorrect parameter count in the call to native function 'lower'
```

Errors

- If the syntax is invalid, the system reports the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .
- If the specified value of an input parameter is invalid, the system reports the following errors: `ERROR 1582 (42000): Incorrect parameter count in the call to native function 'lower'` or `ERROR 1583 (42000): Incorrect parameters in the call to native function 'lower'` .

HEX(*str*)

This function converts a specified string into a hexadecimal string. If the str value is NULL, the function returns NULL.

If the *str* value is a numeric value, this function converts the integer part of the numeric value into a hexadecimal string.

If the *str* value is a string, the function returns a hexadecimal string for the *str* value. Each character in the *str* value is converted into two hexadecimal digits.

Examples

```
mysql> SELECT HEX(255);
        -> 'FF'
mysql> SELECT HEX('abc');
        -> 616263
mysql> SELECT HEX('OceanBase'),
HEX(123),
HEX(0x0123);
+--------------------+----------+-------------+
| HEX('OceanBase')   | HEX(123) | HEX(0x0123) |
+--------------------+----------+-------------+
| 4F6365616E42617365 | 7B       | 0123        |
+--------------------+----------+-------------+
1 row in set (0.00 sec)
mysql> select hex(0x012);
+------------+
| hex(0x012) |
+------------+
| 0012       |
+------------+
1 row in set (0.00 sec)
```

Errors

If the syntax is invalid, the system reports the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .

UNHEX(*str*)

This function is an inverse function of HEX(*str*). The UNHEX(str) function interprets each pair of hexadecimal digits in the str value as a number and converts this number into a character. The returned value is a binary string.

The values of *str* must be strings or NULL. If the *str* value is a valid hexadecimal string, the UNHEX(str) function uses an algorithm to convert hexadecimal values into byte streams. If the *str* value is not a hexadecimal string, the function returns NULL. If the *str* value is NULL, the function returns NULL.

Examples

```
mysql> SELECT HEX('OceanBase'),
UNHEX('4f6365616e42617365'),
UNHEX(HEX('OceanBase')),
UNHEX(NULL)\G;
*************************** 1. row ***************************
HEX('OceanBase'): 4F6365616E42617365
UNHEX('4f6365616e42617365'): OceanBase
UNHEX(HEX('OceanBase')): OceanBase
UNHEX(NULL): NULL
1 row in set (0.00 sec)
mysql> select unhex(abc);
ERROR 1054 (42S22): Unknown column 'abc' in 'field list';
```

Errors

If the data type of the str parameter is invalid, the system reports the following error: `ERROR 1054 (42S22): Unknown column 'abc' in 'field list'` .

INT2IP(*int_value*)

Note The `INT2IP` function is available only in ApsaraDB for OceanBase.

This function converts a specified integer into an IP address.

The data type of the int_value parameter must be INT. If the value of the int_value parameter is NULL, the function returns NULL. If the specified integer is greater than MAX_INT32 or smaller than 0, the function returns NULL. MAX_INT32 is the maximum signed 32-bit integer.

Examples

```
mysql> SELECT INT2IP(16777216),
    -> HEX(16777216),
    -> INT2IP(1);
+-------------------+---------------+-----------+
| INT2IP(16777216)  | HEX(16777216) | INT2IP(1) |
+-------------------+---------------+-----------+
| 1.0.0.0           | 1000000       | 0.0.0.1   |
+-------------------+---------------+-----------+
1 row in set (0.00 sec)
```

IP2INT('*ip_addr*')

Note The `INT2INT` function is available only in ApsaraDB for OceanBase.

This function converts an IP address of the STRING data type into an integer.

Pay attention to the following considerations when you use the function:

- The values of the ip_addr parameter must be strings.

  If the value of the ip_addr parameter is NULL, the function returns NULL. If the specified IP address is invalid, the function returns NULL. For example, the IP address contains non-digit characters, or the number in each segment of an IP address is larger than 256.

- This function supports only IPv4 addresses.

Examples

```
mysql> SELECT IP2INT('0.0.0.1'),
HEX(IP2INT('0.0.0.1')),
HEX(IP2INT('1.0.0.0')),
IP2INT('1.0.0.257')\G;
*************************** 1. row ***************************
IP2INT('0.0.0.1'): 1
HEX(IP2INT('0.0.0.1')): 1
HEX(IP2INT('1.0.0.0')): 1000000
IP2INT('1.0.0.257'): NULL
1 row in set (0.01 sec)
```

[NOT] LIKE *str2* [ESCAPE *str3*]

This function compares strings based on wildcards. The arguments on the left of the LIKE or NOT LIKE keyword and those on the right of the keyword must be strings or NULL. Otherwise, the system reports an error. If the left argument matches the right argument, the LIKE or NOT LIKE function returns TRUE. If the left argument does not match the right argument, the LIKE or NOT LIKE function returns FALSE. If an argument is NULL, the function returns NULL.

The function supports the following wildcards: percent signs (%) and underscores (_).

- A percent sign (%) matches a string of zero or more characters.
- An underscore (_) matches a single character. The matched character must exist.

If you need to search for `a_c` instead of `abc`, you can use `a\\_c`. ApsaraDB for OceanBase uses double backslashes (\\) as escape characters. In a\\_c, the double backslashes (\\) are used to escape the underscore (_).

You can use the ESCAPE clause to specify an escape character. If the *str2* value contains the *str3* value, the str2 characters that follow the *str3* value are processed as general characters. For example, in `LIKE 'abc%' ESCAPE 'c'`, `c` is an escape character. In this case, the percent sign (%) is a general character instead of an escape character. The matched string for this SQL statement is `ab%`.

```
mysql> SELECT 'ab%' LIKE 'abc%' ESCAPE 'c';
+------------------------------+
| 'ab%' LIKE 'abc%' ESCAPE 'c' |
+------------------------------+
|                            1 |
+------------------------------+
1 row in set (0.00 sec)
```

Note When you use the ESCAPE clause to specify an escape character, the escape character must be a one-character string. In the ESCAPE clause, percent signs (%) or underscores (_) cannot be specified as escape characters.

Examples

```
mysql>  select 'a_c' like 'a\\_c';
+---------------------+
| 'a_c' like 'a\\_c' |
+---------------------+
|                  1 |
+---------------------+
1 row in set (0.00 sec)
mysql> select 'abc_' like 'abcdd_' escape 'dd';
ERROR 1064 (42000): Incorrect arguments to ESCAPE
```

expr [NOT] REGEXP | RLIKE *pat*

This function compares a specified string expression with a pattern. The *expr* parameter specifies the string. The *pat* parameter specifies the pattern.

If the *expr* value matches the *pat* value, the function returns 1. Otherwise, the function returns 0. If the *expr* value or the *pat* value is NULL, the function returns NULL.

RLIKE is a synonym for REGEXP.

The values of *expr* and *pat* must be strings or NULL. If the values of the two parameters are numeric values, the system converts the values into strings in an implicit way. If the data type of the expr or pat parameter is invalid, the system reports an error.

The specified pattern must be a valid regular expression. Otherwise, the system reports an error.

```
mysql> select 1234 regexp 1;
+----------------+
| 1234 regexp 1 |
+----------------+
|              1 |
+----------------+
1 row in set (0.00 sec)
mysql>  select 'hello'  rlike 'h%';
+----------------------+
| 'hello'  rlike 'h%' |
+----------------------+
|                    0 |
+----------------------+
1 row in set (0.00 sec)
mysql> select 1234 regexp ^y;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual thatcorresponds to your MySQL server version f
or the right syntax to use near '^y' at line 1 mysql> select yunzhi regexp '^y';
ERROR 1054 (42S22): Unknown column 'yunzhi' in 'field list'
mysql> select 'hello' not  rlike '*h*';
ERROR 1139 (42000): Got error 'repetition-operator operand invalid' from regexp
mysql> select 'hello' not  rlike '*h*';
ERROR 1139 (42000): Got error 'repetition-operator operand invalid' from regexp
```

Errors

- If the syntax of is invalid, the system reports the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .
- If the data type of the expr or pat argument is invalid, the system reports the following error: `ERROR 1054 (42S22): Unknown column 'yunzhi' in 'field list'` .
- If the system cannot identify the specified regular expression, the system reports the following error: `ERROR 1139 (42000): Got error 'repetition-operator operand invalid' from regexp` .

REPEAT(*str*, *count*)

This function repeats a string for a specified number of times. The *str* parameter specifies the string to be repeated. The *count* parameter specifies the number of times. If the specified `count value is smaller than or equal to 0` , the function returns an empty string.

If the value of *str* or *count* is NULL, the function returns NULL.

If the data types of the *str* parameter are numeric data types, the system converts numeric values into strings in an implicit way.

The system can convert the data type of the *count* parameter into a numeric data type in an implicit way. If the conversion fails, the systems considers the *count* value as 0.

Examples

```
mysql> select repeat('1',-1),  repeat(null,null),repeat('test',4);
+----------------+-------------------+------------------+
| repeat('1',-1) | repeat(null,null) | repeat('test',4) |
+----------------+-------------------+------------------+
|                  | NULL              | testtesttesttest |
+----------------+-------------------+------------------+
1 row in set (0.00 sec)
mysql> select repeat(11111,'2');
+-------------------+
| repeat(11111,'2') |
+-------------------+
| 1111111111        |
+-------------------+
1 row in set (0.00 sec)
```

SUBSTRING_INDEX(*str*, *delim*, *count*)

This function returns a specified substring of a string. The returned substring is located before the position where a specified delimiter last occurs in the specified string. In this function, the *delim* parameter specifies the delimiter. The *count* parameter specifies the number of occurrences for the delimiter. The *str* parameter specifies the string from which you want to extract the substring.

If the *count* value is positive, the function determines the last delimiter in the string based on the left-to-right order. In this case, the function returns the characters that are located on the left of the last delimiter. If the *count* value is negative, the function determines the last delimiter in the string based on the right-to-left order. In this case, the function returns the characters that are located on the right of the last delimiter.

If an argument is NULL, the function returns NULL. If the value of the *str* or *delim* parameter is an empty string, the function returns an empty string. If the *count* value is 0, the function returns an empty string.

If the data types of the *str, delim, and count* parameters are numeric data types, the system converts numeric values into strings in an implicit way.

Examples

```
mysql> select substring_index('abcdabc', 'abc', 0), substring_index('abcdabc', 'abc', 1), substring_index('abcdabc', 'abc
', 2), substring_index('abcdabc', 'abc', 3), substring_index('abcdabc', 'abc', -1), substring_index('abcdabc', 'abc', -2)
, substring_index('abcdabc', 'abc', -3)\G;
*************************** 1. row ***************************
 substring_index('abcdabc', 'abc', 0):
 substring_index('abcdabc', 'abc', 1):
 substring_index('abcdabc', 'abc', 2): abcd
 substring_index('abcdabc', 'abc', 3): abcdabc
substring_index('abcdabc', 'abc', -1):
substring_index('abcdabc', 'abc', -2): dabc
substring_index('abcdabc', 'abc', -3): abcdabc
1 row in set (0.00 sec)
```

LOCATE(*substr,str*) , LOCATE(*substr,str,pos*)

The LOCATE(substr,str) function returns the position of the first occurrence for a specified substring in a string. The *substr* parameter specifies the substring and the *str* parameter specifies the source string. The LOCATE(substr,str,pos) function also returns the position of the first occurrence for a specified string in a string. However, this function adds the following input parameter to the LOCATE(substr,str) function: pos. The *pos* parameter specifies the position where the function starts to search for the substring. The *substr* parameter specifies the substring and the *str* parameter specifies the source string. If the *substr* substring is not included in the *str* source string, the function returns 0.

Examples

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar',5);
        -> 7
```

INSTR(*str,substr*)

This function returns the position of the first occurrence for a specified substring in a string that is specified by *str*. This function is the same as the LOCATE(substr,str) function, except that the order of the arguments is reversed.

Examples

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

REPLACE(*str, from_str, to_str*)

This function replaces each specified substring in a string with a new substring. The *from_str* parameter specifies the substring to be replaced. The *to_str* parameter specifies the substring that replaces the specified substring. The *str* parameter specifies the source string.

Examples

```
mysql> SELECT REPLACE('abc.efg.gpg.nowdew.abc.dabc.e', 'abc.', 'www');
+---------------------------------------------------------+
| REPLACE('abc.efg.gpg.nowdew.abc.dabc.e', 'abc.', 'www') |
+---------------------------------------------------------+
| wwwefg.gpg.nowdew.wwwdwwwe                              |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

FIELD(*str,str1,str2,str3,...*)

This function returns the index position of a specified string in a list of strings. In this function, *str* specifies the string and *str1,str2,str3,...* specifies the list of strings. The sequence numbers for index positions start from 1. If the *str* value is not found, the function returns 0.

If all the input values in the FIELD() function are strings, the function compares the input values based on the strings. If all the input values are numbers, the function compares the input values based on the numbers. In other scenarios, the function compares the input values based on the DOUBLE numbers.

If the *str* value is NULL, the function returns 0. This is because NULL cannot be compared with the other values. FIELD() is a supplement to ELT().

Examples

```
mysql> select field('abc','abc1','abc2','abc','abc4','abc'), field(NULL, 'null1', NULL);
+-----------------------------------------------+----------------------------+
| field('abc','abc1','abc2','abc','abc4','abc') | field(NULL, 'null1', NULL) |
+-----------------------------------------------+----------------------------+
|                   3 |                          0 |
+-----------------------------------------------+----------------------------+
1 row in set (0.00 sec)
```

ELT(N, *str1, str2, str3,...*)

This function returns a specified string in a list of strings. The returned string is specified by the index number N. For example, if you set `N to 1` , the function returns the *str1* value. If you set N to 2, the function returns the *str2* value. If N is smaller than 1 or greater than the number of strings in the function, the function returns NULL. ELT() is a supplement to FIELD().

```
mysql> select elt(3, 'abc1', 'abc2', 'abc', 'abc4', 'abc'), elt(0, 'null1', NULL);
+----------------------------------------------+-----------------------+
| elt(3, 'abc1', 'abc2', 'abc', 'abc4', 'abc') | elt(0, 'null1', NULL) |
+----------------------------------------------+-----------------------+
| abc                                          | NULL                  |
+----------------------------------------------+-----------------------+
1 row in set (0.00 sec)
```

INSERT (*str1,pos,len,str2*)

This function replaces a specified substring of a string with a new substring. The *str1* parameter specifies the source string and the *str2* parameter specifies the substring that replaces the original substring. The *pos* parameter specifies the start position of the original substring and the *len* parameter specifies the length of the original substring. If the *pos* value is greater than the length of the source string, the function returns the source string. If the *len* value is greater than the length of the str1 or str2 string, the function replaces the original substring that starts at the *pos* position. If an argument is NULL, the function returns NULL. This function supports multibyte characters.

- The values of *str1* and *str2* must be strings. The values of *pos* and *len* must be integers. If an argument is NULL, the function returns NULL.

- The text characters in *str1* and *str2* are identified as byte streams.
- If the *pos* value is negative or greater than the length of the *str1* value, the function returns the str1 value.
- If the *len* value is smaller than 0 or greater than the length of the *str1* value, the function returns a string that consists of the str2 value and a substring of the str1 string. The substring starts from the first character of the str1 value and ends at the position that is specified by the pos parameter.

Examples

```
mysql> select insert('Quadratic',-2,100,'What'), insert('Quadratic',7,3,'What'), insert('Quadratic',1,3,'What'), insert('
Quadratic',10,3,'What'), insert('Quadratic',5,-1,''), insert('Quadratic',7,-1,'What')\G;
*************************** 1. row ***************************
insert('Quadratic',-2,100,'What'): Quadratic
   insert('Quadratic',7,3,'What'): QuadraWhat
  insert('Quadratic',1,3,'What'): QWhatratic
  insert('Quadratic',10,3,'What'): Quadratic
      insert('Quadratic',5,-1,''): Quad
  insert('Quadratic',7,-1,'What'): QuadraWhat
1 row in set (0.00 sec)
```

5.7.4. Type conversion functions

CAST(*expr* AS *type*)

This function converts a value of *expr* to a value of a data type that is specified by *type*. For more information about data types, see Data types.

Parameter description

`expr` specifies the valid SQL expression.

`AS` separates the two parameters. The parameter before AS specifies the data to be processed. The parameter after AS specifies the destination data type.

`type` specifies the destination data type. This function converts the expr value into a value of the specified data type. You can use one of the following data types as the destination data type:

- CHAR[(N)]. If you use CHAR[N] as the destination data type in the CAST function, the value of the CHAR data type cannot exceed N characters in length.
- DATE
- DATETIME
- DECIMAL
- SIGNED [INTEGER]
- TIME
- UNSIGNED [INTEGER]

The CAST function is applicable when one of the following conditions is met:

- The data types of the two expressions are the same.
- The data types of the two expressions can be converted in an implicit way.
- The data types must be converted in an explicit way.

If an attempt is made to perform an invalid conversion, ApsaraDB for OceanBase returns an error message.

If the length of a data type is not specified, the system uses the maximum length that is supported for the data type in ApsaraDB for OceanBase. For example, the maximum length for the VARCHAR data type is 262,143 bytes. The maximum length for a numeric data type is 65 bits for floating-point numbers.

You can use the CAST function to convert signed and unsigned 64-bit values. If you use a numeric operator such as a plus sign (+) and one of the operands is an unsigned integer, the function returns an unsigned value. To override the numeric operator, you can use the SIGNED or `UNSIGNED cast` operator. The SINGED operator converts a value into a signed 64-bit integer and the UNSIGNED cast operator converts a value into an unsigned 64-bit integers.

If an operand is a floating-point value, the result is a floating-point value.

Examples

```
mysql> select cast(1-2 as unsigned), cast(cast(1-2 as unsigned) as signed);
+-----------------------+--------------------------------------+
| cast(1-2 as unsigned) | cast(cast(1-2 as unsigned) as signed) |
+-----------------------+--------------------------------------+
|  18446744073709551615 |                                    -1 |
+-----------------------+--------------------------------------+
1 row in set (0.00 sec)
mysql>  SELECT CAST(1 AS UNSIGNED) - 2.0;
+---------------------------+
| CAST(1 AS UNSIGNED) - 2.0 |
+---------------------------+
|                      -1.0 |
+---------------------------+
1 row in set (0.00 sec)
mysql> select cast(0 as date);
+-----------------+
| cast(0 as date) |
+-----------------+
| 0000-00-00      |
+-----------------+
1 row in set (0.00 sec)
```

5.7.5. Aggregate functions

Aggregate functions perform calculations on a set of values and return a single value. Aggregate functions ignore NULL values. In most cases, aggregate functions are used in conjunction with GROUP BY clauses of SELECT statements.

All the aggregate functions are deterministic. If aggregate functions are invoked by using the same set of input values, the aggregate functions return the same value.

In ApsaraDB for OceanBase, you can specify only one argument for each aggregate function. For example, `COUNT(c1, c2)` is not supported and `COUNT(c1)` is supported.

AVG(([DISTINCT] *expr*)

This function returns the average for a specified data set. This function ignores the NULL values in the specified data set. The DISTINCT keyword is used to return the average of distinct expr values. If no matched rows are found, the AVG() function returns NULL.

Examples

```
mysql> select * from oceanbasetest;
+----+------+------+
| id | ip   | ip2  |
+----+------+------+
|  1 |    4 | NULL |
|  3 |    3 | NULL |
|  4 |    3 | NULL |
+----+------+------+
3 rows in set (0.01 sec)
mysql> select avg(ip2), avg(ip), avg(distinct(ip)) from oceanbasetest;
+----------+---------+-------------------+
| avg(ip2) | avg(ip) | avg(distinct(ip)) |
+----------+---------+-------------------+
|     NULL |  3.3333 |            3.5000 |
+----------+---------+-------------------+
1 row in set (0.00 sec)
mysql> select avg(distinct(ip)),avg(ip),avg(ip2) from oceanbasetest;
+-------------------+---------+----------+
| avg(distinct(ip)) | avg(ip) | avg(ip2) |
+-------------------+---------+----------+
|            3.5000 |  3.3333 |     NULL |
+-------------------+---------+----------+
1 row in set (0.00 sec)
```

COUNT([DISTINCT] *expr*)

COUNT([DISTINCT] *expr*) returns the number of non-NULL values in the rows that are retrieved by a specified SELECT statement. If no matched rows are found, COUNT() returns 0. The DISTINCT keyword is used to return the number of distinct *expr* values.

COUNT(*) returns the number of retrieved rows. The retrieved rows may contain NULL values.

Examples

```
mysql> select * from oceanbasetest;
+----+------+------+
| id | ip   | ip2  |
+----+------+------+
|  1 |    4 | NULL |
|  3 |    3 | NULL |
|  4 |    3 | NULL |
+----+------+------+
3 rows in set (0.00 sec)
mysql> select count(ip2), count(ip), count(distinct(ip)), count(*) from oceanbasetest;
+------------+-----------+---------------------+----------+
| count(ip2) | count(ip) | count(distinct(ip)) | count(*) |
+------------+-----------+---------------------+----------+
|          0 |         3 |                   2 |        3 |
+------------+-----------+---------------------+----------+
1 row in set (0.00 sec)
```

MAX([DISTINCT] *expr*)

This function returns the maximum value among a specified data set.

You can pass strings into the MAX() function as arguments. If this occurs, the strings are sorted in alphabetical order and the maximum string is returned. You can use the DISTINCT keyword in the function to obtain the maximum value among distinct *expr* value. The function returns the same maximum value if you do not use the DISTINCT keyword.

For example, the a table has three rows of data: `id=1,num=10, id=2,num=20, and id=3,num=30` .

```
mysql> SELECT MAX(num) FROM a;
+-----------------+
| MAX(num)        |
+-----------------+
|              30 |
+-----------------+
1 row in set (0.00 sec)
```

MIN([DISTINCT] *expr*)

This function returns the minimum value among a specified data set.

You can pass strings into the MIN() function as arguments. If this occurs, the strings are sorted in alphabetical order and the minimum string is returned. You can use the DISTINCT keyword in the function to obtain the minimum value of distinct *expr* values. The function returns the same minimum value if you do not use the DISTINCT keyword.

For example, the a table has three rows of data: `id=1,num=10, id=2,num=20, and id=3,num=30` .

```
mysql> SELECT MIN(num) FROM a;
+----------------+
| MIN(num)       |
+----------------+
|             10 |
+----------------+
1 row in set (0.00 sec)
```

SUM([DISTINCT] *expr*)

This function returns the sum of *expr* values. If no rows are found for expr, the SUM() function returns NULL. You can use the DISTINCT keyword to obtain the sum of distinct *expr* values.

If no matched rows are found, the SUM() function returns NULL.

Examples

```
mysql> select * from oceanbasetest;
+------+------+------+
| id   | ip   | ip2  |
+------+------+------+
|    1 |    4 | NULL |
|    3 |    3 | NULL |
|    4 |    3 | NULL |
+------+------+------+
3 rows in set (0.00 sec)
mysql> select sum(ip2),sum(ip),sum(distinct(ip)) from oceanbasetest;
+----------+---------+-------------------+
| sum(ip2) | sum(ip) | sum(distinct(ip)) |
+----------+---------+-------------------+
|     NULL |      10 |                 7 |
+----------+---------+-------------------+
1 row in set (0.00 sec)
```

GROUP_CONCAT([DISTINCT] *expr*)

This function concatenates non-NULL strings from a specified group into a single string.

- Syntax

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
[ORDER BY {unsigned_integer | col_name | expr}
[ASC | DESC] [,col_name ...]]
[SEPARATOR str_val])
```

- Examples

```
mysql> select * from book;     //The table named book (book number, book title, publisher) +--------+------------------
--------------+----------------------------+
| bookid | bookname                        | publishname                  |
+--------+---------------------------------+----------------------------+
|      1 | git help                        | alibaba group publisher     |
|      2 | MySQL Optimization                   | Zhejiang University Press          |
|      3 | Java Programming Guide                    | Machinery Industry Press              |
|      3 | Java Programming Guide                    | Machinery Industry Press              |
|     4 | Large-Scale Distributed Storage System           | Machinery Industry Press               |
+--------+---------------------------------+----------------------------+
5 rows in set (0.00 sec)
//Retrieve book titles.
mysql> select group_concat(bookname) from book group by bookname;
+-----------------------------------+
| group_concat(bookname)            |
+-----------------------------------+
| git help                          |
| Java Programming Guide, Java Programming Guide         |
| MySQL Optimization                     |
| Large-Scale Distributed Storage System              |
+-----------------------------------+
4 rows in set (0.00 sec)
//Retrieve distinct book titles.
mysql> select group_concat(distinct(bookname)) from book group by bookname;
+----------------------------------+
| group_concat(distinct(bookname)) |
+----------------------------------+
| git help                         |
| Java Programming Guide                    |
| MySQL Optimization                     |
| Large-Scale Distributed Storage System             |
+----------------------------------+
4 rows in set (0.01 sec)
mysql> select bookname, group_concat(publishname order by publishname desc separator  ';' ) from book group by bookname
;
+---------------------------------+---------------------------------------------------------------------+
| bookname                        | group_concat(publishname order by publishname desc separator  ';' ) |
+---------------------------------+---------------------------------------------------------------------+
| git help                        | alibaba group publisher                                             |
+---------------------------------+---------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

5.7.6. Mathematical functions

The functions of this type perform mathematical calculations based on numeric expressions.

ROUND(X), ROUND(X,D)

The ROUND() function returns a number that is rounded to the specified length or precision.

This function rounds the X argument to the nearest integer. If you specify two arguments for the function, the function rounds X to D decimal places. The Dth digit is obtained by rounding. To keep D digits of the X value to the left of the decimal point, set D to a negative value.

The data type of the returned value is the same as that of the first argument. You can assume that the value is an integer, double-precision floating-point number, or decimal value. This means that the returned result for an integer argument is also an integer. No fractional part is included in the returned result.

- For exact-value numbers, the ROUND() function rounds values to the nearest integer.

  If the fractional part of a positive value is `.5` or greater than `.5`, the function rounds the positive value up to the next integer. If the fractional part of a negative value is greater than or equal to .5, the function rounds the negative value down to the next integer. In other words, the value is rounded based on the distance between zero and the specified value along the X-axis.

  If the fractional part of a positive value is smaller than `.5`, the function rounds the value down to the next integer. If the fractional part of a negative value is smaller than .5, the function rounds the negative value down to the next integer.

- For approximate-value numbers, the ROUND() function follows the bankers rounding rule. If a value has a fractional part, the function rounds the value to the nearest even integer based on the rule.

Examples

```
mysql> select round(2.15,2);
+---------------+
| round(2.15,2) |
+---------------+
|          2.15 |
+---------------+
1 row in set (0.00 sec)
mysql> select round(2555e-2,1);
+------------------+
| round(2555e-2,1) |
+------------------+
|             25.6 |
mysql> select round(25e-1), round(25.3e-1),round(35e-1);
+--------------+----------------+--------------+
| round(25e-1) | round(25.3e-1) | round(35e-1) |
+--------------+----------------+--------------+
|            3 |              3 |            4 |
+--------------+----------------+--------------+
1 row in set (0.00 sec)
```

CEIL(*expr*)

This function rounds a specified expression value up to the next largest integer.

The function supports comparison operations. The comparison result is a BOOLEAN value. The BOOLEAN value is converted into a numeric value: 1 for TRUE or 0 for FALSE.

If the value of expr is NULL, the function returns NULL.

If you specify a string of numbers, the system converts the string into a numeric value in an implicit way.

The returned value is converted into a BIGINT number.

Examples

```
mysql> select ceil(1.2), ceil(-1.2), ceil(1+1.5), ceil(1=1),ceil(1<1),ceil(null);
+-----------+------------+-------------+-----------+-----------+------------+
| ceil(1.2) | ceil(-1.2) | ceil(1+1.5) | ceil(1=1) | ceil(1<1) | ceil(null) |
+-----------+------------+-------------+-----------+-----------+------------+
|         2 |         -1  |           3  |         1 |         0 |       NULL |
+-----------+------------+-------------+-----------+-----------+------------+
1 row in set (0.00 sec)
mysql> select ceil(name);
ERROR 1166 (42703): Unkown column name 'name'
mysql> select ceil('2');
+-----------+
| ceil('2') |
+-----------+
|         2 |
+-----------+
1 row in set (0.00 sec)
```

FLOOR(*expr*)

This function is similar to the CEIL(*expr*) function. This function rounds a specified expression value down to the next smallest integer.

The function supports comparison operations. The comparison result is a BOOLEAN value. The BOOLEAN value is converted into a numeric value: 1 for TRUE or 0 for FALSE.

If the value of expr is NULL, the function returns NULL.

If you specify a string of numbers, the system converts the string into a numeric value in an implicit way.

The returned value is converted into a BIGINT number.

Examples

```
mysql> select floor(1.2), floor(-1.2), floor(1+1.5), floor(1=1),floor(1<1),floor(null);
+------------+-------------+--------------+------------+------------+-------------+
| floor(1.2) | floor(-1.2) | floor(1+1.5) | floor(1=1) | floor(1<1) | floor(null) |
+------------+-------------+--------------+------------+------------+-------------+
|          1 |          -2 |            2 |          1 |          0 |        NULL |
+------------+-------------+--------------+------------+------------+-------------+
1 row in set (0.00 sec)
mysql> select floor(name);
ERROR 1166 (42703): Unkown column name 'name'
mysql> select floor('2');
+------------+
| floor('2') |
+------------+
|          2 |
+------------+
1 row in set (0.00 sec)
```

ABS(*expr*)

This function returns the absolute value for a specified numeric expression. The data type of the returned value is the same as that of the expression value.

The function supports comparison operations. The comparison result is a BOOLEAN value. The BOOLEAN value is converted into a numeric value: 1 for TRUE or 0 for FALSE.

If the value of expr is NULL, the function returns NULL.

If you specify a string of numbers, the system converts the string into a numeric value in an implicit way.

The returned value is converted into a BIGINT number.

Examples

```
mysql> select abs(5),abs(-5.777),abs(0),abs(1/2),abs(1-5);
+--------+-------------+--------+----------+----------+
| abs(5) | abs(-5.777) | abs(0) | abs(1/2) | abs(1-5) |
+--------+-------------+--------+----------+----------+
|      5 |       5.777 |      0 |      0.5 |        4 |
+--------+-------------+--------+----------+----------+
1 row in set (0.00 sec)
```

NEG(*expr*)

This function is a negation function that subtracts a specified operand from zero and returns the final result.

The function supports comparison operations. The comparison result is a BOOLEAN value. The NEG function performs the NOT operation. The returned result is 1 or 0: 1 for TRUE and 0 for FALSE.

Examples

```
mysql> select neg(1),neg(1+1),neg(2*3),neg(1=1),neg(5<1);
+--------+----------+----------+----------+----------+
| neg(1) | neg(1+1) | neg(2*3) | neg(1=1) | neg(5<1) |
+--------+----------+----------+----------+----------+
|     -1 |       -2 |       -6 |       -1 |        0 |
+--------+----------+----------+----------+----------+
1 row in set (0.00 sec)
```

SIGN(X)

This function returns the sign of a specified number. The function returns -1, 0, or 1 based on whether the specified X value is negative, zero, or positive.

The function supports comparison operations. The comparison result is a BOOLEAN value. The BOOLEAN value is converted into a numeric value: 1 for TRUE and 0 for FALSE.

If the value of expr is NULL, the function returns NULL.

The function supports floating-point numbers and hexadecimal numbers.

Examples

```
mysql> SELECT SIGN(-32);
        -> -1
mysql> SELECT SIGN(0);
        -> 0
mysql> SELECT SIGN(234);
        -> 1
mysql> select sign(null),sign(false),sign(0x01);
+------------+-------------+------------+
| sign(null) | sign(false) | sign(0x01) |
+------------+-------------+------------+
|       NULL |           0 |          1 |
+------------+-------------+------------+
1 row in set (0.00 sec)
```

CONV(N, *from_base, to_base*)

This function converts a number from one base to another base.

The function converts a number from the *from_base* based to the *to_base* base. The returned result is a string. The value of the N input parameter can be an integer or a string. The minimum base is 2 and the maximum base is 36. If the *to_base* value is a negative number, N is processed as a signed number. Otherwise, N is processed as an unsigned number. If the *from_base* value is a negative number, the value is processed as an integer and the value sign is ignored. The data types of the N argument must be INT or STRING.

The values of *from_base* and *to_base* parameters must be decimal INT values. The value range is the union of [-36,-2] and [2,36].

Invalid input values result in errors. In the following scenarios, the input values are invalid:

- The values of the *from_base* or *to_base* parameter are not valid decimal INT values.
- The values of the *from_base* or *to_base* parameter do not fall within the valid value range: union of `[-36,-2] and [2,36]` .
- N is an invalid numeric value. For example, N falls out of the following character range: 0 to 9, a to z, or A to Z.
- N falls out of the range of *from_base* values. For example, the *from_base* value is 2, but N is 3.
- N falls out of the valid range of 64-bit integers. The valid range is [-9223372036854775807, 9223372036854775807].

Examples

```
mysql> select conv(9223372036854775807,10,2);
+-----------------------------------------------------------------+
| conv(9223372036854775807,10,2)                                  |
+-----------------------------------------------------------------+
| 111111111111111111111111111111111111111111111111111111111111111 |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)
mysql> select conv('-acc',21,-7);
+--------------------+
| conv('-acc',21,-7) |
+--------------------+
| -16425             |
+--------------------+
1 row in set (0.00 sec)
```

MOD(*N*,*M*)

This function returns the remainder after a number is divided by another number. MOD(*N*,*M*), *N* % *M*, and *N* MOD *M* are equivalent.

You can use the MOD() function to perform calculations on numbers that have fractional parts. This function returns the exact remainder after a number is divided by another number.

If N or M is NULL, NULL is retuned. If M is 0, NULL is returned.

Examples

```
mysql> select mod(29,19), 29 mod 19, 29 % 19;
+------------+-----------+---------+
| mod(29,19) | 29 mod 19 | 29 % 19 |
+------------+-----------+---------+
|         10 |        10 |      10 |
+------------+-----------+---------+
1 row in set (0.00 sec)
mysql> select mod(19.5, 29);
+---------------+
| mod(19.5, 29) |
+---------------+
|          19.5 |
+---------------+
1 row in set (0.00 sec)
mysql> select mod(29, null);
+---------------+
| mod(29, null) |
+---------------+
|          NULL |
+---------------+
1 row in set (0.00 sec)
mysql> select mod(100,0);
+------------+
| mod(100,0) |
+------------+
|       NULL |
+------------+
1 row in set (0.00 sec)
```

POW(*X*, *Y*)

This function raises X to the power of Y.

Examples

```
mysql> select pow(4,2), pow(4,-2), pow(1,null);
+----------+-----------+-------------+
| pow(4,2) | pow(4,-2) | pow(1,null) |
+----------+-----------+-------------+
|       16 |    0.0625 |        NULL |
+----------+-----------+-------------+
1 row in set (0.00 sec)
```

POWER(*X*, *Y*)

POWER(*X*,*Y*) and POW(*X*,*Y*) are equivalent.

RAND([N])

The `RAND([N])` function accepts zero or one argument N and returns a random floating-point number that falls within the range of [0,1.0). N is called a random seed. If you need to retrieve a random integer that falls within the range of [i, j), you can use the expression FLOOR(I + RAND() * (j - i)).

If you do not specify N, a random seed is generated during initialization. The RAND() function generates a random number based on this random seed. Therefore, the RAND() function generates a different random number each time the function is invoked.

Examples

```
mysql> select a, b, rand() from t3;
+------+------+---------------------+
| a    | b    | rand()              |
+------+------+---------------------+
|    1 |    1 |   0.641815407799385 |
|    2 |    2 | 0.16825051248841966 |
|    3 |    3 |  0.9158063697775886 |
+------+------+---------------------+
3 rows in set (0.00 sec)
mysql> select a, b, rand() from t3;
+------+------+---------------------+
| a    | b    | rand()              |
+------+------+---------------------+
|    1 |    1 | 0.07428034215632857 |
|    2 |    2 |  0.6239826321825224 |
|    3 |    3 |   0.897072165177271 |
+------+------+---------------------+
3 rows in set (0.00 sec)
```

If you specify N, N is used as the random seed to generate random numbers. The function generates random numbers based on whether N is a constant:

- If N is a constant, N is used as the random seed during initialization. Then, the RAND(N) function generates a random number based on the initialized value. If the values of N are the same, the generated random number sequences are the same.

```
mysql> select a, b, rand(3) from t3;
+------+------+---------------------+
| a    | b    | rand(3)             |
+------+------+---------------------+
|    1 |    1 |  0.9057697559760601 |
|    2 |    2 | 0.37307905813034536 |
|    3 |    3 | 0.14808605345719125 |
+------+------+---------------------+
3 rows in set (0.00 sec)
mysql> select a, b, rand(3) from t3;
+------+------+---------------------+
| a    | b    | rand(3)             |
+------+------+---------------------+
|    1 |    1 |  0.9057697559760601 |
|    2 |    2 | 0.37307905813034536 |
|    3 |    3 | 0.14808605345719125 |
+------+------+---------------------+
3 rows in set (0.00 sec)
```

- If N is a variable, such as a column name, N is used as the random seed to generate a random number in each function call. If the values of N are the same, the generated random numbers are the same.

```
mysql> select a, b, rand(a), rand(b) from t3;
+------+------+---------------------+---------------------+
| a    | b    | rand(a)             | rand(b)             |
+------+------+---------------------+---------------------+
|    1 |    1 | 0.40540353712197724 | 0.40540353712197724 |
|    2 |    2 |  0.6555866465490187 |  0.6555866465490187 |
|    3 |    3 |  0.9057697559760601 |  0.9057697559760601 |
+------+------+---------------------+---------------------+
3 rows in set (0.00 sec)
```

The `RAND([N])` function can be used in SELECT statements and the following clauses: `WHERE` , `ORDER BY` , and `GROUP BY` . This function in the clauses runs based on the preceding rules.

For example, if you need to sort a table in a random way, you can execute the following SQL statement: `select * from t1 order by rand()` . If you need to sample 100 rows of a table in a random way, you can execute the following SQL statement: `select * from t1 order by rand() limit 100` .

5.7.7. Comparison functions

Comparison functions compare input values of parameters. The values that can be compared include numbers, characters, and dates.

GREATEST(*value1*, ...)

This function returns the maximum value among the specified input values. This function performs the opposite operation of the `LEAST()` function.

You must specify at least two arguments. If an argument is NULL, the function returns NULL.

If the specified arguments contain numeric values and strings, the system converts the strings to numeric values in an implicit way. If the conversion fails, the system reports errors.

Examples

```
mysql> select greatest('2',1,0), greatest('a','b','c'), greatest('a', NULL, 'c'), greatest('2014-05-15','2014-06-01')\G;
*************************** 1. row ***************************
                  greatest('2',1,0): 2
              greatest('a','b','c'): c
           greatest('a', NULL, 'c'): NULL
greatest('2014-05-15','2014-06-01'): 2014-06-01
1 row in set (0.00 sec)
```

LEAST(*value1*, ...)

This function returns the minimum value among the specified arguments. This function performs the opposite operation of the `GREATEST()` function.

You must specify at least two arguments. If an argument is NULL, the function returns NULL.

If the specified arguments contain numeric values and strings, the system converts the strings to numeric values in an implicit way. If the conversion fails, the system reports errors.

Examples

```
mysql> select least('2',4,9), least('a','b','c'), least('a',NULL,'c'), least('2014-05-15','2014-06-01')\G;
*************************** 1. row ***************************
                  least('2',4,9): 2
              least('a','b','c'): a
             least('a',NULL,'c'): NULL
least('2014-05-15','2014-06-01'): 2014-05-15
1 row in set (0.00 sec)
```

ISNULL(*expr*)

This function checks whether a specified value of an expression is NULL. If the *expr* value is NULL, the `ISNULL()` function returns 1. Otherwise, the function returns 0.

Examples

```
mysql> SELECT ISNULL(null), ISNULL('test'), ISNULL(123.456), ISNULL('10:00');
+--------------+----------------+-----------------+-----------------+
| ISNULL(null) | ISNULL('test') | ISNULL(123.456) | ISNULL('10:00') |
+--------------+----------------+-----------------+-----------------+
|            1 |              0 |               0 |               0 |
+--------------+----------------+-----------------+-----------------+
1 row in set (0.01 sec)
mysql> SELECT ISNULL(null+1);
+----------------+
| ISNULL(null+1) |
+----------------+
|              1 |
+----------------+
1 row in set (0.00 sec)
```

The ISNULL() function can be used as an alternative to the equal sign (=) comparison. You can use an equal sign (=) operator to check whether an expression value is NULL.

Note If the equal sign (=) operator is used to check whether an expression value is NULL, the returned result is invalid in most cases.

The ISNULL() function provides some features that are the same as the IS NULL operator.

5.7.8. Flow control functions

CASE

You can use the following syntax:

```
 CASE value WHEN [compare-value] THEN result [WHEN [compare-value] THEN result ...] [ELSE result] END
```

You can also use the following syntax:

```
CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END
```

For the former syntax, the function returns `value=compare-value`.

For the latter syntax, the function returns the result if the first condition is met. If no matched results are available, the function returns the result of the ELSE part. If the ELSE part is unavailable, the function returns NULL.

Examples

```
 mysql> select CASE 'b' when 'a' then 1 when 'b' then 2 END;
 +----------------------------------------------+
 | CASE 'b' when 'a' then 1 when 'b' then 2 END |
 +----------------------------------------------+
 |                                            2 |
 +----------------------------------------------+
 1 row in set (0.00 sec)
mysql> select CASE concat('a','b') when concat('ab') then 'a' when 'b' then 'b' end; +-----------------------------------
---------------------------------
| CASE concat('a','b') when concat('ab') then 'a' when 'b' then 'b' end| +-----------------------------------------------
---------------------
| a                                                                     +---------------------------------------------------
------------------
 1 row in set (0.00 sec)
mysql> select case when 1>0 then 'true' else 'false' end;
+---------------------------------------------+
| case when 1>0 then 'true' else 'false' end |
+---------------------------------------------+
| true                                        |
+---------------------------------------------+
1 row in set (0.00 sec)
```

IF(*expr1*,*expr2*,*expr3*)

If the *expr1* condition is met, such as *expr1*<>0 and *expr1*<>NULL, the function returns the *expr2* value. Otherwise, the function returns the *expr3* value.

The `IF()` function returns numeric values or strings based on the data types of expr2 and expr3 values.

Examples

```
mysql> select if(5>6, 'T','F'), if (5>6, 1, 0), if(null, 'True', 'False'), if(0, 'True', 'False')\G;
*************************** 1. row ***************************
         if(5>6, 'T','F'): F
           if (5>6, 1, 0): 0
if(null, 'True', 'False'): False
   if(0, 'True', 'False'): False
1 row in set (0.00 sec)
```

If one of *expr2* and *expr3* values is NULL, the data type of the `IF()` function result is the same as that of the non-null expression value.

describes the default data types of the values that are returned by the `IF()` function.

Default data types of returned values

Expression value	Default data type of the returned values
expr2 or *expr3* returns a string.	String
expr2 or *expr3* returns a floating-point number.	Float-pointing number
expr2 or *expr3* returns an integer.	Integer

If *expr2* and *expr3* values are strings and one of the strings is case-sensitive, the returned value is case-sensitive.

IFNULL(*expr1*, *expr2*)

If the *expr1* value is not NULL, the `IFNULL()` function returns the *expr1* value. Otherwise, the function returns the *expr2* value.

The `IFNULL()` function returns numeric values or strings based on the data types of arguments.

Examples

```
mysql> SELECT IFNULL('abc', null), IFNULL(NULL+1, NULL+2), IFNULL(1/0, 0/1);
+---------------------+------------------------+------------------+
| IFNULL('abc', null) | IFNULL(NULL+1, NULL+2) | IFNULL(1/0, 0/1) |
+---------------------+------------------------+------------------+
| abc                 |                   NULL |           0.0000 |
+---------------------+------------------------+------------------+
1 row in set (0.01 sec)
```

NULLIF(*expr1*, *expr2*)

For `expr1=expr2`, the function returns NULL. Otherwise, the function returns the *expr1* value. This rule is the same as that for `CASE WHEN expr2=expr2 THEN NULL ELSE expr1 END`.

Examples

```
mysql> SELECT NULLIF('ABC', 123), NULLIF('123',123), NULLIF(NULL, 'abc');
+--------------------+-------------------+---------------------+
| NULLIF('ABC', 123) | NULLIF('123',123) | NULLIF(NULL, 'abc') |
+--------------------+-------------------+---------------------+
| ABC                | NULL              | NULL                |
+--------------------+-------------------+---------------------+
1 row in set, 1 warning (0.01 sec)
```

Note

5.7.9. Information functions

FOUND_ROWS()

You may use a LIMIT clause in a SELECT statement to limit the number of rows that are returned from the database server to the client.

In some cases, you need to retrieve the actual number of rows that the statement returns if the statement does not have the LIMIT clause. To retrieve the actual number of rows, you can execute the statement that does not have the LIMIT clause again. If you do not want to execute the statement again, you can use `SQL_CALC_FOUND_ROWS` in the SELECT statement. This way, you can invoke the `FOUND_ROW()` function to retrieve the actual number of rows that are returned by the SELECT statement that does not have the LIMIT clause.

Examples

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

The second SELECT statement returns a number that may be different from that for the first SELECT statement. The number indicates how many rows are returned by the first SELECT statement if the first statement does not have the LIMIT clause.

Note Assume that `SQL_CALC_FOUND_ROWS` is not used in the first SELECT statement. When you use the LIMIT clause in the statement, the `FOUND_ROWS()` function returns a value. The returned value may be different from the value that is returned when you do not use the LIMIT clause.

For the SELECT SQL_CALC_FOUND_ROWS statement in the preceding example, the returned value of the `FOUND_ROWS()` function is valid only for a short period. The returned value becomes invalid when the statement that follows the `SELECT SQL_CALC_FOUND_ROWS` statement is executed. If you need to use the returned number of rows later, you must save the number.

Examples

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

If you use `SQL_CALC_FOUND_ROWS` in a query, the system calculates the number of rows in the full result set. This process requires less time than the process of running another query that does not use the LIMIT clause. This is because the result set in the former process does not need to be sent to the client.

Assume that you want to limit the number of rows that a query returns and do not want to run another query to retrieve the number of rows in the full result set. In this case, you can use `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()`. For example, you can use a web script for a paged display. The displayed information contains the links to pages for the other parts of the query results. You can use the `FOUND_ROWS()` function to determine the number of additional pages that are required to show the remaining results.

The implementation of `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` in UNION queries is more complex than that in simple SELECT statements. This is because a UNION query may contain more than one LIMIT clause. For example, you may use LIMIT clauses in the SELECT statements of a UNION query, or you may use the clauses to limit the UNION results.

If `SQL_CALC_FOUND_ROWS` is used for a UNION query, the expected returned result is the row count that is not limited by the global LIMIT clause.

If you need to use `SQL_CALC_FOUND_ROWS` in a UNION query, make sure the following requirements are met:

- The `SQL_CALC_FOUND_ROWS` keyword must appear in the first SELECT statement of the UNION query.
- The UNION ALL syntax is used. The returned value of the `FOUND_ROWS()` function is accurate only if `UNION ALL` is used.
- If the UNION query does not contain LIMIT clauses, `SQL_CALC_FOUND_ROWS` is ignored. In this case, the query returns the number of rows in the temporary table that is created to process the UNION query.

LAST_INSERT_ID()

This function returns the auto-increment field value that is latest inserted in the current session. If you insert multiple rows into the table in the latest operation, the `LAST_INSERT_ID()` function returns the auto-increment field value of the first row.

Examples

```
mysql>select LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                5 |
+------------------+
1 row in set (0.00 sec)
```

5.7.10. Other functions

COALESCE(*expr, expr, expr,...*)

The function evaluates the argument expressions in sequence until the function finds a non-NULL value. Then, the function returns the non-NULL value. If the values of all the expressions are NULL, the function returns a NULL value.

The data types of all the expressions must be the same. If the data types are different, the system converts the data types into the same type in an implicit way.

```
mysql> SELECT COALESCE(NULL,NULL,3,4,5), COALESCE(NULL,NULL,NULL);
  +---------------------------+--------------------------+
  | COALESCE(NULL,NULL,3,4,5) | COALESCE(NULL,NULL,NULL) |
  +---------------------------+--------------------------+
  |                         3 |                     NULL |
  +---------------------------+--------------------------+
1 row in set (0.00 sec)
```

NVL(*str1,replace_with*)

If the *str1* value is NULL, this function replaces the value with the *replace_with* value.

If you set str1 to NULL, the function returns a value that you specify for the replace_with parameter. This helps you retrieve complete outputs. In most cases, the *str1* value is a column name. No limits are placed on the values of `replace_with`. For example, you can specify hard-coded values, references to other columns, or expressions for the replace_with parameter.

Examples

```
mysql> SELECT NVL(NULL, 0), NVL(NULL, 'a');
+--------------+----------------+
| NVL(NULL, 0) | NVL(NULL, 'a') |
+--------------+----------------+
|            0 | a              |
+--------------+----------------+
1 row in set (0.00 sec)
```

ORA_DECODE()

The `ORA_DECODE()` function runs in the same way as the Oracle `DECODE()` function.

Note ApsaraDB for OceanBase is compatible with MySQL and supports some functions of Oracle. The names of the Oracle functions that are used in ApsaraDB for OceanBase start with `ORA_`.

`ora_decode (condition, value 1, returned value 1, value 2, returned value 2, ... value n, returned value n, default value)`

The following section describes the meaning of this function:

```
IF condition = value 1 THEN
....RETURN (returned value 1)
ELSIF condition = value 2 THEN
....RETURN (returned value 2)
       ......
ELSIF condition = value n THEN
....RETURN (returned value n)
ELSE
....RETURN (default value)
END IF
```

SLEEP(duration)

This function suspends an SQL query for a specified duration that is measured in seconds. The function returns 0 after the suspension ends.

If only the SLEEP function is run in a query that is not interrupted, the function returns 0, as shown in the following example:

```
mysql> SELECT SLEEP(1000);
+------------------+
| SLEEP(1000) |
+------------------+
|          0 |
+------------------+
```

If only the SLEEP function is run in a query that is interrupted, the function returns 1 and does not return error codes, as shown in the following example:

```
mysql> SELECT SLEEP(1000);
+------------------+
| SLEEP(1000) |
+------------------+
|          1 |
+------------------+
```

If the SLEEP function is part of a query and the query is interrupted due to the suspension, the function returns the error code 1317, as shown in the following example:

```
mysql> SELECT 1 FROM t1 WHERE SLEEP(1000);
ERROR 1317 (70100): Query execution was interrupted
```

5.7.11. Full-text search functions

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
search_modifier:
  {
       IN NATURAL LANGUAGE MODE
     | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
     | IN BOOLEAN MODE
     | WITH QUERY EXPANSION
  }
```

ApsaraDB for OceanBase allows you to use full-text search functions to search for full-text indexes.

When you use the full-text search functions, make sure that the following requirements are met:

- Full-text indexes are created on the columns that are specified in the `MATCH(col1,col2,...)` function. The function supports only `FULLTEXT CTXCAT` indexes.
- For `FULLTEXT CTXCAT` indexes, the columns in the `MATCH(col1,col2,...)` function must include the columns that are specified for the CTXCAT indexes. For example, you can create the `FULLTEXT INDEX(c1, c2, c3) CTXCAT(c2, c3)` index. In this scenario, the retrieved data can be matched only if the function is `MATCH(c2,c3)` .
- You can use search modifiers that are included in the preceding statement to specify the search mode for full-text search functions. The default search mode is NATURAL LANGUAGE MODE.

Note ApsaraDB for OceanBase supports only two search modes: `NATURAL LANGUAGE MODE` and `BOOLEAN MODE` .

Full-text search: NATURAL LANGUAGE MODE

If you use the default mode or specify the `IN NATURAL LANGUAGE MODE` search modifier, the `MATCH...AGAINST` function uses the `NATURAL LANGUAGE` mode to run a full-text search. In the `NATURAL LANGUAGE` mode, specify the AGAINST argument as a search string. In the index, the function searches for the string by comparing strings based on the character set. For each row in the table, the MATCH function returns a relevance value. The value indicates the relevance between the search string and the row data. To be more specific, the value indicates the similarity between the text of the search string and the text in the data table.

By default, string columns in ApsaraDB for OceanBase are not case-sensitive. Therefore, the keywords for full-text searches are not case-sensitive. If you need to run case-sensitive full-text searches, you can specify case-sensitive data types for the columns on which full-text indexes is created. For example, you can specify the data type of the columns as UTF8MB4_BIN.

If the `MATCH...AGAINST` function is used in the WHERE clause, MATCH is used to filter data that is irrelevant to the keywords in the function. ApsaraDB for OceanBase supports only `MATCH...AGAINST=0` and `MATCH...AGAINST>0` . `MATCH...AGAINST=0` indicates that no data matches the keywords and MATCH...AGAINST>0 indicates that at least one keyword is matched.

You can specify multiple keywords for the AGAINST parameter. You must separate the keywords with spaces and enclose the keywords in a single quotation mark ('). This means that the OR logical operator is applied on the specified keywords. If the text in the table matches one of the keywords, a match occurs.

Full-text search: BOOLEAN MODE

In ApsaraDB for OceanBase, you can run boolean full-text searches by using the `IN BOOLEAN MODE` search modifier. In this mode, some special operators at the beginning of the keywords in search strings have special meanings.

Examples:

```
SELECT * FROM t1 WHERE MATCH (a, b)
    AGAINST ('Chrysanthemum Jasmine' IN BOOLEAN MODE);
+----+------------+------------+
| id | a          | b          |
+----+------------+------------+
| 1  | Alipay     | Chrysanthemum tea      |
| 2  | Taobao       | Jasmine       |
+----+------------+------------+
```

```
SELECT * FROM t1 WHERE MATCH (a, b)
    AGAINST ('+Chrysanthemum -Jasmine tea' IN BOOLEAN MODE);
+----+------------+------------+
| id | a          | b          |
+----+------------+------------+
| 1  | Alipay     | Chrysanthemum tea      |
+----+------------+------------+
```

In ApsaraDB for OceanBase, boolean full-text searches support the following operators:

- The plus sign (+) represents the AND operator. The AND operator indicates that the search results must include the keyword that is preceded by the plus sign (+).
- The minus sign (-) represents the NOT operator. The NOT operator indicates that the search results must exclude the keyword that is preceded by the minus sign (-).
- If *no operator* is specified, the OR operator is applied on the specified keywords. The OR operator indicates that the search results must include at least one of the specified keywords.

When you run boolean full-text searches, make sure the following requirements are met:

- All the operators must be placed at the beginning of keywords. The operators at the end of keywords are ignored. For example, the plus sign (+) in `+Chrysanthemums` is valid and takes effect. The plus sign (+) in `Chrysanthemums+` is invalid and ignored.
- Operators and keywords cannot be separated with other characters. If operators and keywords are separated with other characters, the operators are ignored. For example, the plus sign (+) in `+ Chrysanthemums` is ignored.

5.8. Operators and precedences

5.8.1. Overview

ApsaraDB for OceanBase supports a wide range of operators: arithmetic operators, comparison operators, vector comparison operators, logical operators, and bitwise operators. The other topics in the "Operators and precedences" chapter describe each of the listed operators and their precedences.

5.8.2. Logical operators

In ApsaraDB for OceanBase, the logical operators convert left and right operands into the values of the BOOLEAN data type before the operators perform calculations. If Error is returned for logical operations, calculation errors occur.

In ApsaraDB for OceanBase, you must comply with the following rules when you convert the specified values into the values of the BOOLEAN data type:

- Only the following strings can be converted into the values of the BOOLEAN data type: True, False, 1, and 0. The strings True and 1 are converted into True Boolean values. The strings False and 0 are converted into False Boolean values.
- If a numeric value of the INT, FLOAT, DOUBLE, or DECIMAL data type is converted into a value of the BOOLEAN data type and the numeric value is not zero, the result value is True. If the numeric value to be converted is zero, the result value is False.

NOT!

The logical NOT operator. If you need to use the operator to perform calculations, you must convert the data types of input values based on the specified rules. The following table describes the rules of converting the data types for the operator.

INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
True/False	True/False	True/False	Error	True/False/E	True/False	NULL

Example:

```
mysql> SELECT NOT 0, NOT 1, NOT NULL;
+-------+-------+----------+
| NOT 0 | NOT 1 | NOT NULL |
+-------+-------+----------+
|     1 |     0 |     NULL |
+-------+-------+----------+
1 row in set (0.00 sec)
mysql> select not current_timestamp;
+-----------------------+
| not current_timestamp |
+-----------------------+
|                     0 |
+-----------------------+
1 row in set (0.00 sec)
mysql> select not now();
+-----------+
| not now() |
+-----------+
|         0 |
+-----------+
1 row in set (0.00 sec)
```

AND &&

The logical AND operator. If you need to use the operator to perform calculations, you must convert the data types of input values based on the specified rules. The following table describes the rules of converting the data types for the operator.

- INT FLOAT DOUBLE TIMESTAMP VARCHAR BOOL NULL INT True/False True/False True/False Error True/False/Error True/False False/NULL FLOAT - True/False True/False Error True/False/Error True/False False/NULL DOUBLE - - True/False Error True/False/Error True/False False/NULL TIMESTAMP - - - Error Error True/False Error VARCHAR - - - - True/False/Error True/False/Error False/NULL BOOL - - - - - True/False False/NULL NULL - - - - - - NULL

Examples:

```
mysql> SELECT (0 AND 0), (0 AND 1), (1 AND 1), (1 AND NULL);
+-----------+-----------+-----------+--------------+
| (0 AND 0) | (0 AND 1) | (1 AND 1) | (1 AND NULL) |
+-----------+-----------+-----------+--------------+
|         0 |         0 |         1 |         NULL |
+-----------+-----------+-----------+--------------+
1 row in set (0.00 sec)
```

OR ||

The logical OR operator. If you need to use the operator to perform calculations, you must convert the data types of input values based on the specified rules. The following table describes the rules of converting the data types for the operator. - INT FLOAT DOUBLE TIMESTAMP VARCHAR BOOL NULL INT True/False True/False True/False Error True/False/Error True/False True/NULL FLOAT - True/False True/False Error True/False/Error True/False True/NULL DOUBLE - - True/False Error True/False/Error True/False True/NULL TIMESTAMP - - - Error Error Error Error VARCHAR - - - - True/False/Error True/False/Error True/NULL BOOL - - - - - True/False True/NULL NULL - - - - - - NULL

Example:

```
mysql> SELECT (0 OR 0), (0 OR 1), (1 OR 1), (1 AND NULL);
+----------+----------+----------+--------------+
| (0 OR 0) | (0 OR 1) | (1 OR 1) | (1 AND NULL) |
+----------+----------+----------+--------------+
|        0 |        1 |        1 |         NULL |
+----------+----------+----------+--------------+
1 row in set (0.01 sec)
```

5.8.3. Arithmetic operators

In ApsaraDB for OceanBase, you can perform arithmetic operations on only the values of the numeric data types or the VARCHAR data type. If you perform arithmetic operations on the values of other data types, errors are reported. If you perform arithmetic operations on strings and the strings cannot be converted to the values of the DOUBLE data type, errors are reported. For example, if you perform the `'3.4he' + 3` operation, an error is reported.

Strings can be converted into the values of the DOUBLE data type only in two scenarios. In one of the scenarios, each character in the strings is a digit. In the other scenario, the strings start with plus signs (+) or minus signs (-) and the characters that follow the plus signs (+) or the minus signs (-) are digits.

describes the arithmetic operators that ApsaraDB for OceanBase supports.

Supported arithmetic operators

Expression	Description	Example
+	The addition operator.	SELECT 2+3;
-	The minus operator.	SELECT 2-3;
*	The multiplication operator.	SELECT 2*3;
/	The division operator. The returned results are quotients. If the divisor is 0, `NULL` is returned.	SELECT 2/3;
% or MOD	The modulo operator. The returned results are remainders. If the divisor is 0, `NULL` is returned.	SELECT 2%3, 2 MOD 3;
^	Raises the specified number to the power of another number.	SELECT 2^2

If you need to use the addition (+), minus (-), and asterisk (*) operators to perform calculations, you must convert the data types of input values based on the specified rules. The following table describes the rules of converting the data types for these operators. - INT FLOAT DOUBLE TIMESTAMP VARCHAR BOOL INT INT DOUBLE DOUBLE Error DOUBLE/Error Error FLOAT - DOUBLE DOUBLE Error DOUBLE/Error Error DOUBLE - - DOUBLE Error DOUBLE/Error Error TIMESTAMP - - - Error Error Error VARCHAR - - - - DOUBLE/Error Error BOOL - - - - - Error

If you need to use the division (/) operator to perform calculations, you must convert the data types of input values based on the specified rules. The following table describes the rules of converting the data types for the operator. If the divisor is zero, an error is returned.

- INT FLOAT DOUBLE TIMESTAMP VARCHAR BOOL NULL INT DOUBLE/Error DOUBLE/Error DOUBLE/Error Error DOUBLE/Error Error NULL/Error FLOAT - DOUBLE/Error DOUBLE/Error Error DOUBLE/Error Error NULL/Error DOUBLE - - DOUBLE/Error Error DOUBLE/Error Error NULL/Error TIMESTAMP - - - Error Error Error Error VARCHAR - - - - DOUBLE/Error Error NULL/Error BOOL - - - - - Error Error NULL NULL

If you need to use the percent sign (%) and MOD operators to perform calculations, you must convert the data types of input values based on the specified rules. The following table describes the rules of converting the data types for these operators. - INT FLOAT DOUBLE TIMESTAMP VARCHAR BOOL NULL INT INT DOUBLE DOUBLE Error DOUBLE/Error Error NULL FLOAT - DOUBLE DOUBLE Error DOUBLE/Error Error NULL DOUBLE - - DOUBLE Error DOUBLE/Error Error NULL TIMESTAMP - - - Error Error Error Error VARCHAR - - - - DOUBLE/Error Error NULL/Error BOOL - - - - - Error Error NULL NULL

5.8.4. Comparison operators

In ApsaraDB for OceanBase, operands are compared after they are converted into the values of the same type. You can use comparison operators to compare the operands. All the comparison operators return NULL or the values of the BOOLEAN data type. If the comparison result is true, 1 is returned. If the comparison result is false, 0 is returned. If the comparison result is unknown, NULL is returned.

Compare numeric values

You can use a comparison operator to compare two numeric values.

describes the comparison operators that ApsaraDB for OceanBase supports.

Comparison operators

Expression	Description	Example
=	Equal to	`SELECT 1=0, 1=1, 1=NULL;`
>=	Greater than or equal to	`SELECT 1>=0, 1>=1, 1>=2, 1>=NULL;`
>	Greater than	`SELECT 1>0, 1>1, 1>2, 1>NULL;`
<=	Less than or equal to	`SELECT 1<=0, 1<=1, 1<=2, 1<=NULL;`
<	Less than	`SELECT 1<0, 1<1, 1<2, 1<NULL;`
! = or <>	Not equal to	`SELECT 1! =0, 1! =1, 1<>0, 1<>1, 1! =NULL, 1<>NULL;`

If you need to use the comparison operators to perform calculations, you must convert the data types of the input values based on the specified rules. The following table describes the rules of converting the data types for the comparison operators. - INT FLOAT DOUBLE TIMESTAMP VARCHAR BOOL NULL INT INT FLOAT DOUBLE Error ? INT Error NULL FLOAT - FLOAT DOUBLE Error ? FLOAT Error NULL DOUBLE - - DOUBLE Error ? DOUBLE Error NULL TIMESTAMP - - - TIMESTAMP ? TIMESTAMP Error NULL VARCHAR - - - - VARCHAR ? BOOL NULL BOOL - - - - - BOOL NULL NULL NULL

Note

- ? ApsaraDB for OceanBase attempts to convert the specified data types to the data types that start with question marks (?) before the comparisons are performed. If the type conversion fails, errors are reported.
- If you compare the values of the BOOLEAN data type, False is smaller than True.

[NOT] BETWEEN ... AND ...

Checks whether the specified value falls in or out of a range of values.

```
mysql> SELECT 2 BETWEEN 1 AND 2,
3 NOT BETWEEN 1 AND 2,
1 BETWEEN null AND 0,
1 NOT BETWEEN null AND 0\G;
*************************** 1. row ***************************
2 BETWEEN 1 AND 2: 1
3 NOT BETWEEN 1 AND 2: 1
1 BETWEEN null AND 0: 0
1 NOT BETWEEN null AND 0: 1
1 row in set (0.00 sec)
```

Example:

The following two tables are available: emp that stores the employee information and salgrade that stores the salary information. The employee information includes employee names and salaries. The salary information includes salary ranges and salary levels. In this example, you can execute the following statements to query the employee names, salaries, and salary levels.

```
 mysql> select * from emp;  //emp is the table that stores the employee information.
 +--------+----------+
 | ename  | sal      |
 +--------+----------+
 | Jerry  | 25000.00 |
 | Larry  | 40000.00 |
 | Maggie | 46000.00 |
 | Micky  | 15000.00 |
 +--------+----------+
 4 rows in set (0.00 sec)
 mysql> select * from salgrade; //salgrade is the table that stores the salary information.
 +-------+----------+----------+
 | grade | losal    | hisal    |
 +-------+----------+----------+
 |     1 | 10000.00 | 20000.00 |
 |     2 | 20001.00 | 30000.00 |
 |     3 | 30001.00 | 40000.00 |
 |     4 | 40001.00 | 50000.00 |
 |     5 | 50001.00 | 90000.00 |
 +-------+----------+----------+
 5 rows in set (0.00 sec)
 mysql> select a1.ename, a1.sal, a2.grade from emp a1, salgrade a2 where a1.sal between a2.losal and a2.hisal;  //The BET
WEEN...AND... operator specifies the query criteria.
 +--------+----------+-------+
 | ename  | sal      | grade |
 +--------+----------+-------+
 | Micky  | 15000.00 |     1 |
 | Jerry  | 25000.00 |     2 |
 | Larry  | 40000.00 |     3 |
 | Maggie | 46000.00 |     4 |
 +--------+----------+-------+
 4 rows in set (0.00 sec)
```

[NOT] IN

Checks whether the specified value falls within a specified range.

Example:

```
mysql> SELECT 2 IN (1, 2), 3 IN (1, 2)\G;
*************************** 1. row ***************************
2 IN (1, 2): 1
3 IN (1, 2): 0
1 row in set (0.00 sec)
```

IS [NOT] NULL | TRUE | FALSE | UNKNOWN

Checks whether a value is NULL, true, false, or unknown. If the operation is successful, TRUE or FALSE is returned. In this scenario, NULL is not returned.

Example:

```
mysql> SELECT 0 IS NULL,
NULL IS NULL, NULL IS TRUE,
(0>1) IS FALSE,
NULL IS UNKNOWN,
0 IS NOT NULL,
NULL IS NOT NULL,
NULL IS NOT TRUE,
(0>1) IS NOT FALSE,
NULL IS NOT UNKNOWN\G;
*************************** 1. row ***************************
          0 IS NULL: 0
       NULL IS NULL: 1
       NULL IS TRUE: 0
     (0>1) IS FALSE: 1
    NULL IS UNKNOWN: 1
      0 IS NOT NULL: 1
   NULL IS NOT NULL: 0
   NULL IS NOT TRUE: 1
 (0>1) IS NOT FALSE: 0
NULL IS NOT UNKNOWN: 0
1 row in set (0.00 sec)
```

Vector comparison operators

Vector comparison operators compare two vectors or rows. The supported operators are `<`, `>`, `=`, `<=`, `>=`, `! =`, `<=>`, `IN`, and `NOT IN`. All these operators are binary operators. Each two vectors to be compared must have the same number of dimensions.

The <=> operator represents `NULL-safe equal`. The <=> operator is similar to the equal to (=) operator and performs equality comparisons. However, if both operands are NULL, the <=> operator returns 1 instead of NULL. If only one operand is NULL, the <=> operator returns 0 instead of NULL.

The expressions (1,2) and ROW(1,2) are also known as row constructors. The expressions (1,2) and ROW(1,2) are equivalent. The two expressions are valid in other contexts. For example, the following two statements are equivalent. However, only the second statement can be optimized.

```
SELECT * FROM t1 WHERE (column1, column2) = (1, 1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

If comparison results are returned after the first i scalars of each two operands are compared, vector comparison operators do not continue to compare the other numeric values of the operands. This is the main difference between vector comparison operators and general operators.

When you use vector comparison operators, pay attention to the following considerations:

- To compare n-tuple vectors, you can omit the ROW keyword. Note that n must be larger than or equal to three. For example, `ROW(1, 2,3) < ROW(1,3,5)` is equivalent to `(1,2,3) < (1,3,5)`.
- The IN and NOT IN operators can be used only in vector operations. The IN operator indicates that the left parameter value is included in the right value set. The NOT IN operator indicates that the left parameter value is excluded from the right value set. Each right value set is enclosed in parentheses (). For example, you can specify `1 in (2, 3, 1)` for comparison.
- For `IN or NOT IN` operators, the scalar operands in the corresponding positions must be comparable. Otherwise, errors are reported. For example, `ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4))` is valid and `ROW(1,2) in (ROW(2,1), ROW(2,3), ROW(1,3,4))` is invalid.

  Examples:

```
mysql> SELECT ROW(1,2) < ROW(1, 3),
    -> ROW(1,2,10) < ROW(1, 3, 0),
    -> ROW(1,null) < ROW(1,0),
    -> ROW(null, 1) < ROW(null, 2),
    -> ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)),
    -> 1 in (1,2,3),
    -> 1 not in (2,3,4),
    -> ROW(1,2) not in (ROW(2,1),ROW(2,3), ROW(3,4)),
    -> NULL = NULL,
    -> NULL <=> NULL,
    -> NULL <=> 1,
    -> 1 <=> 0 \G;
*************************** 1. row ***************************
                                ROW(1,2) < ROW(1, 3): 1
                          ROW(1,2,10) < ROW(1, 3, 0): 1
                              ROW(1,null) < ROW(1,0): NULL
                         ROW(null, 1) < ROW(null, 2): NULL
ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)): 1
                                        1 in (1,2,3): 1
                                    1 not in (2,3,4): 1
       ROW(1,2) not in (ROW(2,1),ROW(2,3), ROW(3,4)): 1
                                         NULL = NULL: NULL
                                       NULL <=> NULL: 1
                                          NULL <=> 1: 0
                                             1 <=> 0: 0
1 row in set (0.00 sec)
```

5.8.5. Vector comparison operators

Vector comparison operators compare two vectors or rows. The supported operators are less than (<), greater than (>), equal to (=), less than or equal to (<=), greater than or equal to (>=), not equal to (!=), <=>, IN, and NOT IN. All these operators are binary operators. Each two vectors to be compared must have the same number of dimensions.

The <=> operator represents NULL-safe equal. The <=> operator is similar to the equal to the (=) operator and performs equality comparisons. However, if both operands are NULL, the <=> operator returns 1 instead of NULL. If only one operand is NULL, the <=> operator returns 0 instead of NULL.

The expressions (1,2) and ROW(1,2) are also known as row constructors. The expressions (1,2) and ROW(1,2) are equivalent. The two expressions are valid in other contexts.

For example, the following two statements are equivalent. However, only the second statement can be optimized.

```
SELECT * FROM t1 WHERE (column1, column2) = (1, 1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

Vector comparison operators differ from general operators in the following aspects:

- If comparison results are returned after the first i scalars of each two operands are compared, vector comparison operators do not continue to compare the other numeric values of the operands.
- When you use vector comparison operators, pay attention to the following considerations:
  - To compare n-tuple vectors, you can omit the ROW keyword. Note that n must be larger than or equal to three.

    For example, `ROW(1,2,3) < ROW(1,3,5)` is equivalent to `(1,2,3) < (1,3,5)` .
  - The `IN and NOT IN` operators can be used only in vector operations. The IN operator indicates that the left parameter value is included in the right value set. The NOT IN operator indicates that the left parameter value is excluded from the right value set. Each right value set is enclosed in parentheses ().

    For example, you can specify `1 in (2, 3, 1)` for comparison.
  - For `IN or NOT IN` operators, the scalar operands in the corresponding positions must be comparable. Otherwise, errors are reported.

    For example, `ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4))` is valid and `ROW(1,2) in (ROW(2,1), ROW(2,3), ROW(1,3,4))` is invalid.

Examples

```
mysql> SELECT ROW(1,2) < ROW(1, 3),
    -> ROW(1,2,10) < ROW(1, 3, 0),
    -> ROW(1,null) < ROW(1,0),
    -> ROW(null, 1) < ROW(null, 2),
    -> ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)),
    -> 1 in (1,2,3),
    -> 1 not in (2,3,4),
    -> ROW(1,2) not in (ROW(2,1),ROW(2,3), ROW(3,4)),
    -> NULL = NULL,
    -> NULL <=> NULL,
    -> NULL <=> 1,
    -> 1 <=> 0 \G;
*************************** 1. row ***************************
                                ROW(1,2) < ROW(1, 3): 1
                          ROW(1,2,10) < ROW(1, 3, 0): 1
                              ROW(1,null) < ROW(1,0): NULL
                         ROW(null, 1) < ROW(null, 2): NULL
ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)): 1
                                        1 in (1,2,3): 1
                                    1 not in (2,3,4): 1
       ROW(1,2) not in (ROW(2,1),ROW(2,3), ROW(3,4)): 1
                                         NULL = NULL: NULL
                                       NULL <=> NULL: 1
                                          NULL <=> 1: 0
                                             1 <=> 0: 0
1 row in set (0.00 sec)
```

5.8.6. Bitwise operators

In ApsaraDB for OceanBase, bit operations are performed on the values of the BIGINT data type. BIGINT is a data type that stores 64-bit integers. For bitwise operators, the maximum number of bits for each value is 64.

describes the bitwise operators that ApsaraDB for OceanBase supports.

Bitwise operators

Expression	Description	Example
BIT_COUNT(N)	Returns the number of bits that are specified by the parameter N.	`SELECT BIT_COUNT(29);-> 4`
&	The bitwise AND operator.	`SELECT 29 & 15;-> 13` The returned value is an unsigned 64-bit integer.
~	Inverts all the bits.	`SELECT 29 & ~15;-> 16` The returned value is an unsigned 64-bit integer.
|	The bitwise OR operator.	`SELECT 29 | ~15;-> 31` The returned value is an unsigned 64-bit integer.
^	The bitwise XOR operator.	`SELECT 1 ^ 1;-> 0` The returned value is an unsigned 64-bit integer.

Expression	Description	Example
<<	Shifts the specified value of the BIGINT data type to the left by two bits.	`SELECT 1 << 2;-> 4` The returned value is an unsigned 64-bit integer.
>>	Shifts the specified value of the BIGINT data type to the right by two bits.	`SELECT 4 << 2;-> 1` The returned value is an unsigned 64-bit integer.

5.8.7. Operator precedences

If you need to use ApsaraDB for OceanBase operators to perform mixed operations, you must familiarize yourself with the operator precedences.

describes the operators that ApsaraDB for OceanBase supports. The following table lists the operators based on precedences in descending order.

Operator precedences

Precedence	Operator
15	!
14	- (unary minus) and ~
13	^
12	*, /, %, and MOD
11	+, -
10	<<, >>
9	&
8	|
7	= (comparison operator: equal to), <=>, >, >=, <, <=, <>, ! =, IS, LIKE, REGEXP, and IN
6	BETWEEN
5	NOT
4	AND and &&
3	XOR
2	OR and ||
1	= (assignment operator)

Note You can use parentheses () to enclose the operations that you want to perform before the other operations. This also helps you identify the operations that have high precedences.

5.9. Escape characters

An escape character is a character sequence that is prefixed with a backslash (\) in a string and invokes an alternative interpretation on the subsequent characters.

Escape characters are case-sensitive. For example, `\b` represents the backspace and `\B` represents the B character.

describes the escape characters that can be recognized by ApsaraDB for OceanBase.

Escape characters

Escape character	Description
\b	The backspace.
\f	The form feed. Note MySQL does not support this escape character.
\n	The line feed.
\r	The carriage return.
\t	The tab character.
\\	The backslash (\).
\'	The single quotation mark (').
\"	The double quotation mark (").
_	The underscore (_).
\%	The percent sign (%).
\0	The NULL character.
\Z	The ASCII 26 character. The corresponding key combination is Ctrl+Z. In Windows, ASCII 26 is the end-of-file indicator. You can encode the ASCII 26 character as `\Z` so that ASCII 26 is not interpreted as the end-of-file indicator.

5.10. DDL statements

5.10.1. Overview

Data definition language (DDL) statements allow you to manage basic database components. For example, you can execute DDL statements to create, modify, and delete tables.

ApsaraDB for OceanBase supports a wide range of DDL statements. For example, the following statements are supported: `CREATE DATABASE` , `ALTER DATABASE` , `DROP DATABASE` , `CREATE TABLE` , `DROP TABLE` , `ALTER TABLE` , `CREATE INDEX` , `DROP INDEX` , `CREATE VIEW` , `DROP VIEW` , `ALTER VIEW` , `TRUNCATE TABLE` .

5.10.2. CREATE DATABASE

Syntax

```
CREATE DATABASE [IF NOT EXISTS] dbname
    [create_specification_list];
create_specification_list:
      create_specification [create_specification…]
create_specification:
      [DEFAULT] CHARACTER SET [=] charsetname
      | [DEFAULT] COLLATE [=] collationname
      | REPLICA_NUM [=] num
      | PRIMARY_ZONE [=] zone
| DEFAULT TABLEGROUP [=] {NULL | tablegroupname}
```

The `CREATE DATABASE` statement allows you to create a database. When you create the database, you can specify the default attributes for the database, such as the default character set and collation.

Notes:

- REPLICA_NUM specifies the number of replicas.
- PRIMARY_ZONE specifies the primary zone of the database. Before you specify this attribute, make sure that the primary zone is

included in the zone list of the tenant.
- DEFAULT TABLEGROUP specifies the default table group of the database. If you do not specify this parameter, the default value NULL is used.

Examples

```
root@(none) 01:36:27>create database test2 default CHARACTER SET UTF8;
Query OK, 1 row affected (0.00 sec)
root@(none) 01:36:44>create database test3 CHARACTER SET UTF8;
Query OK, 1 row affected (0.00 sec)
```

Errors

- If the database that you want to create already exists and you do not add `IF NOT EXITS` to the statement, the system returns the following error: `ERROR 1007 (HY000): Can't create database 'test3'; database exists`.
- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax`.
- If the length of the specified database name exceeds the maximum length, the system returns the following error: `ERROR 1059 (42000): Identifier name 'XXXX' is too long`.

5.10.3. ALTER_DATABASE

Syntax

```
ALTER DATABASE [dbname]
     alter_specification_list;
alter_specification_list:
      [SET] alter_specification [alter_specification…]
alter_specification:
      [DEFAULT] {CHARACTER SET | CHARSET} [=] charsetname
      | [DEFAULT] COLLATE [=] collationname
      | REPLICA_NUM [=] num
      | PRIMARY_ZONE [=] zonename
      | {READ ONLY | READ WRITE}
| DEFAULT TABLEGROUP [=] {NULL | tablegroupname}
```

The ALTER_DATABASE statement allows you to change the attributes of a specified database. For example, you can change the following attributes: character set, collation, the number of replicas, primary zone, database permissions, and default table group. REPLICA_NUM specifies the number of replicas. PRIMARY_ZONE specifies the primary zone. `READ ONLY| READ WRITE` specifies whether the database is read-only. READ ONLY indicates that users can have only read access to the database, and READ WRITE indicates that users can have read and write access to the database. `DEFAULT TABLEGROUP` specifies the default table group of the database. If the value is NULL, the default table group is deleted.

The database name is optional. If you do not specify the database name, the statement takes effect on the current database.

Examples

```
alter database test2 DEFAULT CHARACTER SET UTF8;
```

Errors

- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax`.
- If the specified database name is invalid or the database does not exist, the system returns the following error: `ERROR 1049 (42000): Unknown database`.

  ```
  mysql> alter database notest default character set utf8;
  ERROR 1049 (42000): Unknown database
  ```

5.10.4. DROP DATABASE

Syntax

```
DROP DATABASE [IF EXISTS] dbname;
```

The `DROP DATABASE` statement allows you to drop all the tables in a specified database and delete the database.

If you add `IF EXISTS` to the statement and the specified database does not exist, no errors are reported.

Examples

```
mysql> drop database notest;
ERROR 1008 (HY000): Can't drop database 'notest'; database doesn't exist
mysql> drop database if exists notest;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> show warnings;
+-------+------+-----------------------------------------------------+
| Level | Code | Message                                             |
+-------+------+-----------------------------------------------------+
| Note  | 1008 | Can't drop database 'notest'; database doesn't exist |
+-------+------+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Errors

- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax`.
- If the specified database does not exist, the system returns the following error: `ERROR 1008 (HY000): Can't drop database 'XXX'; database doesn't exist`.

5.10.5. CREATE TABLE

The CREATE TABLE statement allows you to create a table in a specified ApsaraDB for OceanBase database.

Syntax

```
CREATE TABLE [IF NOT EXIST] tblname
(create_definition,...)
[table_options]
[partition_options];
CREATE TABLE [IF NOT EXISTS] tblname
LIKE oldtblname
create_definition:
colname column_definition
| PRIMARY KEY (index_col_name [, index_col_name...]) [index_type] [index_options]...
      | {INDEX|KEY} [indexname] (index_col_name,...) [index_type] [index_options]...
      | UNIQUE [INDEX|KEY] [indexname] (index_col_name,...) [index_type]  [index_options]...
| FULLTEXT [INDEX|KEY] [indexname] (index_col_name,...) CTXCAT(index_col_name,...) [index_options]...
column_definition:
     data_type [NOT NULL | NULL] [DEFAULT defaultvalue]
      [AUTO_INCREMENT] [UNIQUE [KEY]] | [[PRIMARY] KEY]
      [COMMENT 'string']
   | [data_type] [GENERATED ALWAYS] AS (expression)
      [VIRTUAL | STORED] [UNIQUE [KEY]] [COMMENT comment]
      [NOT NULL | NULL] [[PRIMARY] KEY]
data_type:
  TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME[(fsp)]
  | TIMESTAMP[(fsp)]
  | DATETIME[(fsp)]
  | YEAR
  | CHAR[(length)]
      [CHARACTER SET charsetname] [COLLATE collationname]
  | VARCHAR(length)
      [CHARACTER SET charsetname] [COLLATE collationname]
  | BINARY[(length)]
  | VARBINARY(length)
index_col_name:
     colname [(length)] [ASC | DESC] ApsaraDB for OceanBase does not support prefix indexes.[(length)]
index_type:
      USING BTREE
index_options:
      index_option [index_option…]
index_option:
GLOBAL [LOCAL]
    |COMMENT 'string'
    |COMPRESSION [=] {NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}
    |BLOCK_SIZE [=] size
    |STORING(columname_list)
| VISIBLE [INVISIBLE]
columname_list:
    colname [, colname…]
table_options:
     table_option [table_option]...
table_option:
[DEFAULT] {CHARACTER SET| CHARSET} [=] charsetname
| [DEFAULT] COLLATE [=] collationname
| COMMENT [=] 'string'
| COMPRESSION [=] {NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}
| EXPIRE_INFO [=] expr
| REPLICA_NUM [=] num
| TABLE_ID [=] id
| BLOCK_SIZE [=] size
| USE_BLOOM_FILTER [=] {True | False}
| STEP_MERGE_NUM [=] num
     | TABLEGROUP [=] 'tablegroupname'
| PRIMARY_ZONE [=] zonelist
| AUTO_INCREMENT [=] num
| PCTFREE [=] integer
     | LOCALITY [=] locality
partition_options:
     PARTITION BY
   HASH(expr)
   |KEY(column_list)
   PARTITIONS num
   [partition_definition ...]
partition_definition:
 COMMENT [=] 'string'
```

Note B-tree indexes are created for the data that is stored in ApsaraDB for OceanBase. The data is sorted based on primary keys. In ApsaraDB for OceanBase, you do not need to specify primary keys because the system automatically generates the primary keys.

The `CREATE TABLE` statement supports the UNIQUE constraint. The CREATE TABLE statement does not support temporary tables or the CHECK constraint. When you execute the statement to create a table, you cannot import data from other tables to the table that you want to create.

Notes:

- If you add `IF NOT EXISTS` to the statement and the table that you want to create already exists, the system does not return an error. If you do not add IF NOT EXISTS to the statement, the system returns an error.
- For more information about the `data_type` parameter, see Data types.
- `NOT NULL`, `DEFAULT`, and `AUTO_INCREMENT` are used to implement integrity constraints on columns.
- describes the `table_option` parameters. The parameters are separated with commas (,).

Parameters of table_option

Parameter	Description	Example
CHARACTER SET	The character set that is used to encode the strings in the table. The character set is used to provide the metadata information. Set the value to utf8mb4.	`CHARACTER SET = 'utf8mb4'`
COMMENT	The description of the table.	`COMMENT='create by Bruce'`
TABLE_ID	The ID of the table. If the specified table ID is smaller than 50,000, turn on the `enable_sys_table_ddl` switch of the RootServer. If you are a general user, we recommend that you do not specify the table ID.	`TABLE_ID =4000`
BLOCK_SIZE	The micro-block size of a partition.	The default size is 16 KB.
USE_BLOOM_FILTER	Specifies whether to use a Bloom filter when the data in the table is read. Valid values: ◦ False: indicates that the Bloom filter is not used. False is the default value. ◦ True: indicates that the Bloom filter is used.	`USE_BLOOM_FILTER = False`
TABLEGROUP	The table group to which the table belongs.	-
REPLICA_NUM	The number of replicas for the partitions in the table. The default value is 3.	`REPLICA_NUM = 3`
ZONE_LIST	The list of clusters.	-
PRIMARY_ZONE	The primary zone.	-
AUTO_INCREMENT	The start value for an auto-increment field.	`AUTO_INCREMENT = 5`

Parameter	Description	Example

PCTFREE	The percentage of the idle space that is reserved in a database macro-block.	When you use the parameter, pay attention to the following syntax details:
		◦ In the Oracle syntax, the PCTFREE parameter specifies the similar setting. To ensure compatibility with the Oracle syntax, ApsaraDB for OceanBase uses the same parameter name.
		◦ Use the following syntax to create a table:
		`CREATE TABLE table_name (column_definition) PCTFREE [=] integer`
		◦ Use the following syntax to modify a table:
		`ALTER TABLE table_name PCTFREE [=] integer`
		Note
		◦ If you do not specify the PCTFREE parameter when you create a table, the default value 10 is used.
		◦ The specified value of the PCTFREE parameter must be an integer. The valid value range is [0,50).

- In `index_option`, you can use the GLOBAL keyword to specify global indexes and use the LOCAL keyword to specify local indexes. By default, `global indexes` are used. When you create a partitioned table, you must specify the LOCAL keyword to use local indexes. If you do not specify the LOCAL keyword, the system returns an error.

Examples

The following example is used to illustrate how to create a database table and view the table information.

1. Execute the following statement to create a database table:

   ```
   CREATE TABLE test (c1 int primary key, c2 VARCHAR(50)) REPLICA_NUM = 3, PRIMARY_ZONE = 'zone1';
   ```

   You can also execute the following statement:

   ```
   CREATE TABLE test (c1 int, c2 VARCHAR(50), primary key(c1)) REPLICA_NUM = 3, PRIMARY_ZONE = 'zone1';
   ```

2. Execute the following statement to view the table information:

   ```
   SHOW tables;
   DESCRIBE test;
   ```

Errors

- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax`.
- If the table name already exists, the system returns the following error: `ERROR 1050 (42S01): Table 'test' already exists`.
- If the length of the specified table name exceeds the maximum length, the system returns the following error: `ERROR 1059 (42000): Identifier name 'XXXX' is too long`.
- If duplicate partition names are found in the table to be created, the system returns the following error: `ERROR 1517 (HY000): Duplicate partition name` `XX`. The following example is provided to explain the error:

  ```
  mysql> create table employeestest(id int) partition by range(id) (partition dpname values less than (10), partition dPn
  ame values less than (20));
  ERROR 1517 (HY000): Duplicate partition name 'dPname'
  ```

5.10.6. ALTER TABLE

The ALTER TABLE statement allows you to update the schema of an existing table. For example, you can modify an existing table and table attributes, add columns, modify columns and column attributes, or delete columns.

Syntax

```
ALTER TABLE tblname
alter_specification [, alter_specification]...
alter_specification:
   ADD [COLUMN] colname column_definition
  | ADD [COLUMN] (colname column_definition,...)
  | ADD {INDEX | KEY} [indexname] (index_col_name,...) [index_tpye] [index_options]
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY (index_col_name,...) [index_type] [index_options] #ApsaraDB for OceanBase does not support the ADD PR
IMARY KEY clause.
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX | KEY] [indexname] (index_col_name,...) [index_type] [index_options]
  | ALTER [COLUMN] colname {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] oldcolname newcolname column_definition        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] colname column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] colname
  | DROP PRIMARY KEY#ApsaraDB for OceanBase does not support the DROP PRIMARY KEY clause. You cannot use the DROP PRIMARY
KEY clause to drop a primary key.
  | DROP {INDEX | KEY} indexname
  | RENAME [TO | AS] newtblname
  | ORDER BY colname
  | CONVERT TO CHARACTER SET charsetname [COLLATE collationname]
  | [DEFAULT] CHARACTER SET charsetname [COLLATE collationname]
  | table_options
  | partition_options
  | ADD PARTITION partition_definition
  | DROP PARTITION partition_names
  | COALESCE PARTITION number
  | REORGANIZE PARTITION partition_names INTO (partition_definitions)
  | ANALYZE PARTITION partition_names
  | CHECK PARTITION partition_names
  | OPTIMIZE PARTITION partition_names
  | REBUILD PARTITION partition_names
  | REPAIR PARTITION partition_names
  | DROP TABLEGROUP
  | AUTO_INCREMENT [=] num
column_definition:
     data_type [NOT NULL | NULL] [DEFAULT defaultvalue]
      [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
      [COMMENT 'string']
table_options:
     [SET] table_option [table_option]...
table_option:
[DEFAULT] {CHARACTER SET | CHARSET} [=] charsetname
| [DEFAULT] COLLATE [=] collationname
| COMMENT [=] 'string'
| COMPRESSION [=] {NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}
| EXPIRE_INFO [=] expr
| REPLICA_NUM [=] num
| TABLE_ID [=] id
| BLOCK_SIZE [=] size
| USE_BLOOM_FILTER [=] {True| False}
| STEP_MERGE_NUM [=] num
| TABLEGROUP [=] tablegroupname
| PRIMARY_ZONE [=] zonelist
| AUTO_INCREMENT [=] num
     | PCTFREE [=] integer
| {READ ONLY | READ WRITE}
     | LOCALITY [=] locality
partition_options:
     PARTITION BY
         HASH(expr)
         | KEY(column_list)
         [PARTITIONS num]
        [partition_definition ...]
partition_definition:
 COMMENT [=] 'commenttext'
```

ApsaraDB for OceanBase does not support the alter_specification clauses of the `ALTER TABLE` statement. In ApsaraDB for OceanBase, the table ID in table_option cannot be changed. However, if you attempt to change the table ID, the system does not return syntax errors. ApsaraDB for OceanBase supports the alter_specification clauses that are frequently used.

Add columns

```
ALTER TABLE tblname
       ADD [COLUMN] col_name column_definition;
```

For more information about the `data_type` parameter, see Data types.

ApsaraDB for OceanBase does not allow you to add primary key columns.

Modify column attributes

```
ALTER TABLE tblname
   ALTER [COLUMN] colname
   [SET DEFAULT literal| DROP DEFAULT];
```

Drop columns

```
ALTER TABLE tblname
    DROP [COLUMN] colname;
```

ApsaraDB for OceanBase does not allow you to drop the primary key columns or the columns that have indexes.

Rename tables

```
ALTER TABLE tblname
RENAME [TO] newtblname;
```

Rename columns

```
ALTER TABLE tblname
       CHANGE [COLUMN] oldcolname newcolname column_definition;
```

Notice For the columns of the VARCHAR data type, the values of only the VARCHAR data type can be increased. The other data types are not supported.

For example, execute the following statement to change the field name d in the t2 table to c and change the data type of the field.

```
ALTER TABLE t2 CHANGE COLUMN d c CHAR(10);
```

Specify the block size of a partitioned table

```
ALTER TABLE tblname
    SET BLOCK_SIZE [=] blocksize;
```

Specify the number of replicas for a table

```
ALTER TABLE tblname
    SET REPLICA_NUM [=] num;
```

REPLICA_NUM specifies the number of replicas for the specified table.

Specify the compression method for a table

```
ALTER TABLE tblname
    SET COMPRESSION [=] '{NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}';
```

Specify whether to use a Bloom filter

```
ALTER TABLE tblname
    SET USE_BLOOM_FILTER [=] {True | Flase};
```

Add descriptions

```
ALTER TABLE tblname
    SET COMMENT [=] 'commentstring';
```

Specify the number of macro-blocks that are to be merged at a time for progressive compaction

```
ALTER TABLE tblname
  SET PROGRESSIVE_MERGE_NUM [=] num;
```

You can execute this statement to specify the number of macro-blocks that are to be merged at a time for progressive compaction. The value of the `PROGRESSIVE_MERGE_NUM` parameter ranges from 1 to 64.

Specify the zone for a table

```
ALTER TABLE tblname
    zone_specification...;
zone_specification:
    PRIMARY_ZONE [=] zone
```

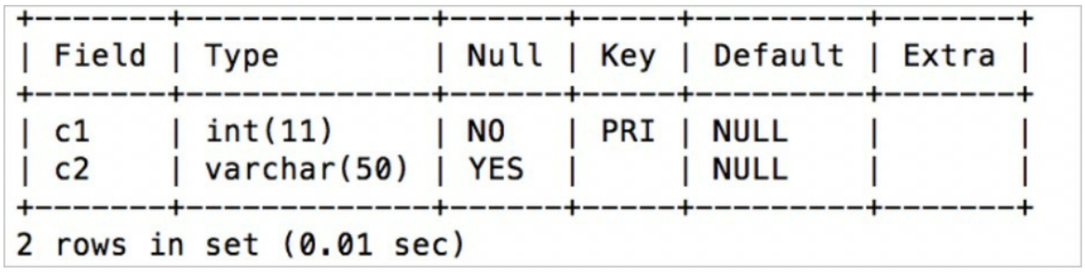
Examples

Example 1:

1. Before you add columns, execute the following statement to view the table information. Table information before you add columns shows the table information:

   ```
   DESCRIBE test;
   ```

   Table information before you add columns



```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| c1    | int(11)     | NO   | PRI | NULL    |       |
| c2    | varchar(50) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

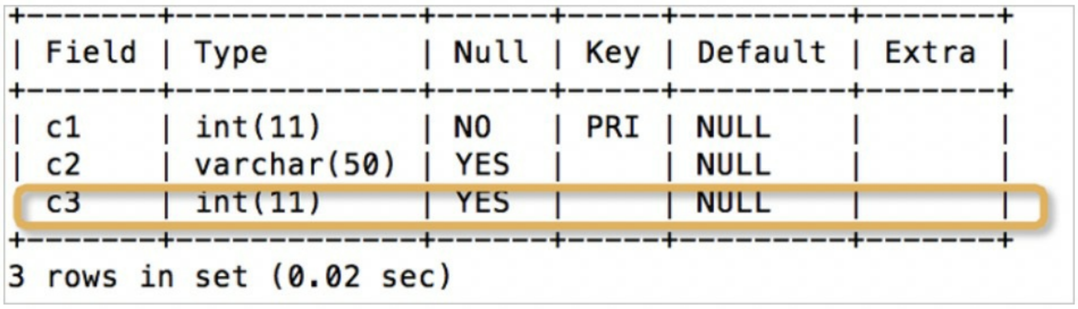
2. Execute the following statement to add the c3 column:

   ```
   ALTER TABLE test ADD c3 int;
   ```

3. After you add the column, execute the following statement to view the table information that is shown in Table information after you add the column:

   ```
   DESCRIBE test;
   ```

   Table information after you add the column



```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| c1    | int(11)     | NO   | PRI | NULL    |       |
| c2    | varchar(50) | YES  |     | NULL    |       |
| c3    | int(11)     | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

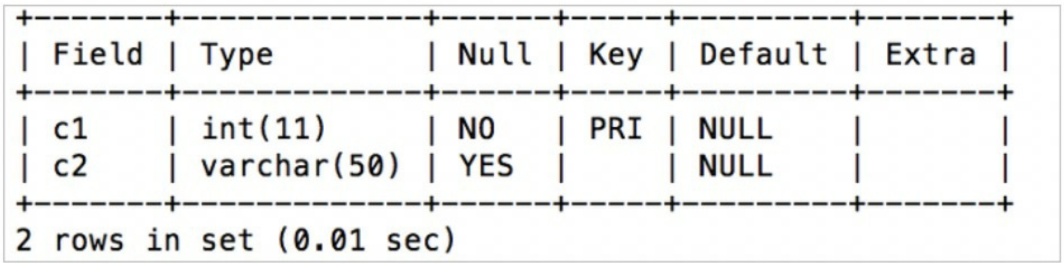
4. Execute the following statement to delete the c3 column:

   ```
   ALTER TABLE test DROP c3;
   ```

5. After you delete the column, execute the following statement to view the table information that is shown in Table information after you delete the column:

   ```
   DESCRIBE test;
   ```

   Table information after you delete the column



```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| c1    | int(11)     | NO   | PRI | NULL    |       |
| c2    | varchar(50) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Example 2:

```
ALTER TABLE test SET REPLICA_NUM=2, ADD COLUMN c5 INT;
```

Errors

- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax` .
- If the specified table does not exist, the system returns the following error: `ERROR 1146 (42S02): Table 'XXX' doesn't exist` .
- If you add primary key columns to an existing table, the system returns the following error: `ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table` .

  ```
  mysql> select * from employees;
  +----+-------+-------+------------+------------+----------+----------+
  | id | frame | lname | hired      | separated  | job_code | store_id |
  +----+-------+-------+------------+------------+----------+----------+
  |  4 | 4     | 4     | 2000-04-04 | 2044-04-04 |        1 |        4 |
  |  5 | 5     | 5     | 2000-05-05 | 2022-05-05 |      123 |        5 |
  |  8 | 8     | 8     | 2000-05-05 | 2001-08-08 |      123 |        9 |
  |  2 | test  | 2     | 2000-02-02 | 2024-02-02 |        2 |        2 |
  |  7 | 7     | 7     | 1999-02-02 | 2007-07-07 |        7 |       10 |
  |  3 | 3     | 3     | 2000-03-03 | 2034-03-03 |        3 |        3 |
  | 11 | test  | test  | 2003-03-03 | 2003-05-05 |       11 |       11 |
  +----+-------+-------+------------+------------+----------+----------+
  7 rows in set (0.01 sec)
  mysql> alter table employees add id2 int primary key;
  ERROR 1068 (42000): Multiple primary key defined
  ```

- If the column that you want to delete does not exist, the system returns the following error: `ERROR 1091 (42000): Can't DROP 'XXX'; check that column/key exists` . The following example is provided to explain the error.

  ```
  mysql> alter table employees drop id3;
  ERROR 1091 (42000): Can't DROP 'id3'; check that column/key exists
  ```

- If the specified data type does not meet the requirements of the default data length, the system returns the following error: `ERROR 1067 (42000): Invalid default value for 'XXX'` . The following example is provided to explain the error.

  ```
  mysql> alter table test1 add colum1 VARCHAR(10)  default 'tttttttttttttttttttttt' ;
  ERROR 1067 (42000): Invalid default value for 'colum1'
  ```

5.10.7. DROP TABLE

The DROP TABLE statement allows you to drop tables that are stored in an ApsaraDB for OceanBase database.

Syntax

```
DROP TABLE [IF EXISTS] tbl_list;
tbl_list:
     tblname [, tblname …]
```

If you add `IF EXISTS` to the statement and the table that you want to delete does not exist, the system does not return an error. If you do not add IF EXISTS to the statement, the system returns an error.

If you need to drop multiple tables at a time, separate the tables with commas (,).

Examples

```
DROP TABLE IF EXISTS test;
```

Errors

- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax` .
- If the table that you want to delete does not exist, the system returns the following error: `ERROR 1051 (42S02): Unknown table 'XXXX'` .

5.10.8. CREATE INDEX

Indexes are created on tables to sort the values of one or more columns of the tables. Indexes are used to reduce the response time of queries and reduce the performance overhead of database systems.

Note

1. In ApsaraDB for OceanBase, a new index takes effect only after a daily major freeze operation is performed. For example, you can create a unique index. Before the unique index takes effect, you can insert indexes that violate the uniqueness constraint. After the major freeze operation is performed, the unique index cannot take effect because duplicate indexes exist. In this scenario, the system changes the status of the unique index to index_error.
2. You can execute the `SHOW INDEX from <table>` statement to view the index status.

Syntax

```
CREATE [UNIQUE] INDEX indexname
        ON tblname (index_col_name,...)
      [index_type] [index_options]
| CREATE FULLTEXT INDEX indexname ON tblname (index_col_name,...) CTXCAT(index_col_name,...) [index_options]
index_type:
      USING BTREE
index_options:
      index_option [index_option…]
index_option:
    GLOBAL [LOCAL]
    | COMMENT 'string'
    | COMPRESSION [=] {NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}
    | BLOCK_SIZE [=] size
    | STORING(columname_list)
index_col_name:
    colname [(length)] [ASC | DESC]
columname_list:
    colname [, colname…]
```

Notes:

- In `index_col_name` , you can specify ASC or DESC for each column name. ASC indicates that the values are sorted in ascending order and DESC indicates that the values are sorted in descending order. By default, the values are sorted in ascending order.
- In the CREATE INDEX statement, the indexes are first sorted based on the values of the first column in `index_col_name` . If the values in the first column are the same, the indexes are sorted based on the values in the next column. Similar rules apply to the other columns.
- You can execute the `SHOW INDEX FROM tblname` statement to view the created indexes.
- In `index_option` , you can use the GLOBAL keyword to specify global indexes and use the LOCAL keyword to specify local indexes. By default, `global indexes` are used.

  When you create a partitioned table, you must specify the LOCAL keyword to use local indexes. If you do not specify the LOCAL keyword, the system returns an error. Separate the `index option` parameters with spaces.

- STORING is an optional field. This field indicates that the specified columns are stored in the index table for redundant storage. This improves the query performance of systems. The STORING field is available only in ApsaraDB for OceanBase.

  Note B-tree indexes are created for the data that is stored in ApsaraDB for OceanBase.

Examples

1. Execute the following statement to create a table that is named test:

   ```
   CREATE TABLE test (c1 int primary key, c2 VARCHAR(10));
   ```

2. Execute the following statement to create indexes on the test table:

   ```
   CREATE INDEX test_index ON test (c1, c2 DESC);
   ```

3. Execute the following statement to view the indexes of the test table:

   ```
   SHOW INDEX FROM test;
   ```

Errors

- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax` .
- If the specified table name is invalid, the system returns the following error: `ERROR 1146 (42S02): Table 'XXX' doesn't exist` .
- If the length of the specified table name exceeds the maximum length, the system returns the following error: `ERROR 1059 (42000): Identifier name 'XXX' is too long` .
- If the specified column name is invalid, the system returns the following error: `ERROR 1072 (42000): Key column 'XXX' doesn't exist in table` .

Expression indexes

ApsaraDB for OceanBase allows you to use the specified syntax to create generated columns. Generated columns are divided into virtual columns and stored columns. Generated columns can be used to create indexes in the same way as general columns.

If INSERT, REPLACE, and UPDATE operations are performed on virtual columns, the values in the `virtual columns` are materialized into index tables. Stored columns are the same as general columns except that the values of the stored columns are obtained from expressions.

Prefix indexes

You can create indexes on the prefixes of string values that are stored in columns. You can use the col_name(length) syntax to specify the prefix length.

Notes:

- You can create prefix indexes on the columns of the CHAR, VARCHAR, BINARY, or VARBINARY data type.
- You must specify prefixes if you need to create indexes on the columns of the BLOB or TEXT data type.

FULLETXT CTXCAT indexes

ApsaraDB for OceanBase supports FULLTEXT CTXCAT indexes. The full-text indexes of this type can cover the columns that are used for text analysis and the columns that are not used for text analysis. General indexes can cover only the columns that are used for text analysis. You can use the following syntax to create FULLTEXT CTXCAT indexes.

```
CREATE FULLTEXT INDEX indexname ON tblname (index_col_name,...) CTXCAT(index_col_name,...) [index_options]
```

where:

- `tblname (index_col_name,...)` specifies the table columns on which the index is created. It also specifies the ordinal positions of the columns in the index.
- `CTXCAT(index_col_name,...)` specifies the columns that are used for full-text analysis.

In ApsaraDB for OceanBase, multiple columns can be used for full-text analysis. If multiple columns are used for full-text analysis, the text set for full-text analysis is the sum of the text sets that are stored in the corresponding columns. The OR logical operator is used to combine the text sets that are stored in the corresponding columns. Only the columns of the STRING data type can be used for full-text analysis.

In the column list for the index, you must declare the columns that are specified for full-text analysis if you execute a DDL statement to create a CTXCAT index. The ordinal positions of the specified columns that are used for full-text analysis must be continuous. The following examples are provided to explain these rules:

```
create table t1(a int primary key, b varchar(100), c varchar(100), d int);
create fulltext index i1 on t1(b, d) ctxcat(c);
ERROR 1072 (42000): Key column 'c' doesn't exist in table
```

In the preceding example, the specified column for full-text analysis is not included in the column list of the index.

```
create table t1(a int primary key, b varchar(100), c varchar(100), d int);
create fulltext index i1 on t1(b, d, c) ctxcat(b, c);
ERROR 5291 (HY000): The CTXCAT column must be contiguous in the index column list
```

In the column list of the index, the ordinal positions of the specified columns for full-text analysis are not continuous in the preceding example.

For more information about how to use `FULLTEXT CTXCAT` indexes, see MATCH... AGAINST in Full-text search functions.

Invisible indexes

You can use invisible indexes in many scenarios. One example of the scenarios is that you do not want the optimizer to use an index. Another example is that an index has not been used by queries and is always in the idle state. However, you do not want to delete the index because the index may be used in subsequent queries or the risks of deleting the index cannot be assessed. In these scenarios, `invisible indexes` help you handle the challenges. You can mark the index as an invisible index so that the index is not used by the optimizer.

When you create an index, you can specify whether the index is an invisible index. The default status is visible. If you do not specify the index status, the default status is used.

You can specify whether indexes are invisible when you execute DDL statements to create tables and indexes or create only indexes. The following examples show how to specify the index status.

```
CREATE TABLE t1(c1 int primary key, c2 int, c3 int, c4 varchar(16), key idx1(c1) visible, index idx2(c2) invisible, uniqu
e key idx3(c3) visible, unique index idx4(c1, c2) invisible, unique idx5(c2,c3));
CREATE UNIQUE INDEX idx6 ON t1(c2) visible;
CREATE INDEX idx7 ON t1(c3) invisible;
CREATE INDEX idx8 ON t1(c4) ;
```

In the preceding example, the status of idx1, idx3, and idx6 is specified as visible and the status of idx2, idx4, and idx7 is specified as invisible. The status of idx5 and idx8 is not specified. Therefore, the default visible status is used for idx5 and idx8.

You can also use the `ALTER TABLE ALTER INDEX` syntax to change the status of the created indexes.

`ALTER TABLE t1 ALTER INDEX idx8 invisible;` changes the status of the idx8 index from visible to invisible. As a result, the optimizer cannot use the index.

5.10.9. DROP INDEX

High overheads are required to maintain a large number of indexes. We recommend that you drop the indexes that are not required.

Note After you execute the DROP INDEX statement to drop the index, the index cannot be immediately dropped. You must wait for a period before the index is dropped.

Syntax

```
DROP INDEX indexname
    ON tblname;
```

Examples

```
DROP INDEX test_index ON test;
```

Errors

- If a syntax error occurs, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax`.
- If the specified table name is invalid, the system returns the following error: `ERROR 1146 (42S02): Table 'XXX' doesn't exist`.
- If the length of the specified table name exceeds the maximum length, the system returns the following error: `ERROR 1059 (42000): Identifier name 'XXX' is too long`.
- If the specified index is invalid, the system returns the following error: `ERROR 1091 (42000): Can't DROP 'XXX'; check that`

`column/key exists` .

5.10.10. CREATE VIEW

Syntax

```
CREATE [OR REPLACE] VIEW viewname
 [(column_list)] AS select_stmt;
```

The CREATE VIEW statement allows you to create a view. If you add the `OR REPLACE` clause to the statement, you can execute the statement to replace an existing view.

In the syntax, select_stmt specifies a SELECT statement. The SELECT statement defines the view by selecting data from base tables or other views.

In views, column names must be unique. This rule for views is the same as that for base tables. By default, the column names that are returned by the SELECT statement are used as the column names for the view. To specify the column names of the view, use the optional column_list clause. You can use this clause to specify the column names. The specified column names are separated with commas (,). The number of the column names in the column_list clause must be equal to the number of columns that are returned by the SELECT statement.

The columns that are returned by the SELECT statement can be the specified columns of base tables. The columns that are returned by the SELECT statement can also store the computing results of the expressions that use functions, constant values, or operators.

Views are not stored as physical tables in databases. Views are generated each time you send a request to access the views. Views are created based on the outputs of the SELECT statements that are specified in the `CREATE VIEW` statements.

ApsaraDB for OceanBase supports only the views that cannot be updated.

5.10.11. DROP VIEW

Syntax

```
DROP VIEW [IF EXISTS]
    viewname [, viewname …] ;
```

The `DROP VIEW` statement allows you to delete one or more views. To drop the views, you must have the DROP permission on each view.

If you add `IF EXISTS` to the statement and the specified views do not exist, no errors are reported.

5.10.12. ALTER VIEW

Syntax

```
ALTER VIEW viewname [(column_list)]
     [(column_list)] AS select_statement
```

The ALTER VIEW statement allows you to change the definition of an existing view. The syntax of the statement is similar to that of the `CREATE VIEW` statement.

Notice

- ApsaraDB for OceanBase does not support the `ALTER VIEW` statement. This indicates that views cannot be modified.
- Views are logical tables. If you need to modify views, you can delete the views and create new views based on your business needs.

5.10.13. TRUNCATE TABLE

Syntax

```
TRUNCATE [TABLE] tblname;
```

The TRUNCATE TABLE statement allows you to delete a specified table and retain the table schema that includes the partition information about the table. The TRUNCATE TABLE statement implements the same logic as the `DELETE FROM` statement. You can execute the DELETE FROM statement to delete all the rows in a table. To execute the TRUNCATE TABLE statement, you must have the permissions to create and delete tables. The TRUNCATE TABLE statement is a DDL statement.

The `TRUNCATE TABLE` and `DELETE FROM` statements have the following differences:

- The TRUNCATE TABLE statement deletes and recreates the table. The response time for the TRUNCATE TABLE statement is shorter than that for the DELETE FROM statement. This is because the DELETE FROM statement deletes rows one after one.
- The output of the `TRUNCATE TABLE` statement shows that the number of affected rows is always 0.
- If you execute the `TRUNCATE TABLE` statement, each auto-incremented value is reset to the start value. The table manager does not store the latest auto-incremented value.
- You cannot execute the TRUNCATE TABLE statement when a transaction is processed or the table is locked. If you execute the statement in these scenarios, the system returns errors.
- If the file that defines the table is valid, you can execute the `TRUNCATE TABLE` statement to recreate the table as an empty table. This occurs even if the data or the index file has been corrupted.

In ApsaraDB for OceanBase, the specified table in the `TRUNCATE TABLE` statement can have only one partition.

ApsaraDB for OceanBase supports the `ALTER TABLE TRUNCATE PARTITION` statement. This allows you to implement distributed transactions. This occurs only if you can execute the TRUNCATE TABLE statement on tables that have multiple partitions in the later versions of ApsaraDB for OceanBase.

5.10.14. RENAME TABLE

Syntax

```
RENAME TABLE tblname TO newtblname
     [, tblname2 TO newtblname …] ;
```

The `RENAME TABLE` statement allows you to rename one or more tables.

After you execute the RENAME TABLE statement, the specified tables are automatically renamed. When the tables are renamed, other threads cannot read data from the tables.

Examples

For example, if you have a source table that is named oldtable, you can create an empty table that has the same schema as the source table. You can name the empty table newtable and replace the source table with this empty table:

```
CREATE TABLE newtable(…) ;
RENAME TABLE oldtable TO backuptable, newtable TO oldtable;
```

If you execute this statement to rename multiple tables, the tables are renamed based on the left-to-right order.

If you need to swap the names of two tables and the tmptable table does not exist, execute the following statement:

```
RENAME TABLE oldtable TO tmptable,
newtable TO oldtable,
temptable TO newtable
```

For the same tenant, you can rename database tables and move the tables from the databases to other databases.

```
RENANME TABLE currentdb.tblname TO otherdb.tblname
```

Before you execute the RENAME statement to rename tables, make sure that no tables are locked or involved in active transactions. To execute the RENAME TABLE statement, you must have ALTER and DROP permissions on the source table. You must also have CREATE and INSERT permissions on the destination table.

You can execute the `RENAME TABLE` statement to rename views. Note that the source view and the destination view must belong to the same database.

5.10.15. CREATE SYNONYM

Synonyms are aliases of objects in databases. You can define aliases for most database objects, such as tables, views, materialized views, sequences, functions, stored procedures, packages, and synonyms.

Synonyms simplify the SQL-based development of applications. If you need to use objects in SQL statements and no synonyms are specified for the objects, you must obtain the locations where the objects are stored. For example, you must obtain the information about the database that stores the specified table object. If you have specified synonyms for the objects, you do not need to concern yourself with this issue.

Synonyms also offer other benefits. The following example is provided to explain another benefit. In the example, a developer writes an SQL statement. The SQL statement involves a table that is used in the production system. If a synonym is specified for the table, the developer can associate the synonym with a mock table during the initial test stage. After the test is passed, the developer can associate the synonym with the table that is stored in the production system. This avoids code modifications and reduces the impact of the test on the production system.

Syntax

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM
[DATABASE.]synonym_name
FOR [DATABASE.]object_name;
```

You can use the `CREATE SYNONYM` syntax to create synonyms for objects. To create synonyms for objects, you do not need to have the permissions on the objects. However, you must have the permissions to create synonyms in the specified database. If you specify the PUBLIC keyword when you create a synonym, the synonym belongs to no databases. Therefore, the synonym does not share the same namespace with databases and belongs to the public namespace.

5.10.16. DROP SYNONYM

Syntax

```
DROP [PUBLIC] SYNONYM
[DATABASE.]synonym_name
[FORCE]
```

You can use the `DROP SYNONYM` syntax to drop synonyms. If you need to delete a public synonym, you must specify the PUBLIC keyword. In this scenario, you cannot specify the database for the synonym that is specified by the synonym_name parameter.

5.11. DML statements

5.11.1. Overview

Data manipulation language (DML) statements allow you to write, delete, and update data in databases.

ApsaraDB for OceanBase supports the following DML statements: INSERT, REPLACE, SELECT, UPDATE, and DELETE.

The following DML limits apply to partitioned tables:

- ApsaraDB for OceanBase distributes data across partitions. To ensure system performance, WHERE clauses in SELECT, UPDATE, and DELETE statements must include `partition(p0, p1,...)`. If WHERE clauses in SELECT, UPDATE, and DELETE statements do not include `partition(partition_list)`, all the partitions are scanned before the statements return data.
- If SELECT, UPDATE, and DELETE statements do not include `partition(partition_list)`, the system does not return syntax errors. In this scenario, the statements return results only after a full table scan is completed.
- You can specify only one partition in each REPLACE, INSERT, UPDATE, or DELETE statement. If you specify multiple partitions, the system returns errors.

If you need to specify multiple partitions in each SELECT statement, pay attention to the following considerations:

- If you send requests to read data from multiple `partition groups`, you can read data across partitions and only weak consistency is supported.
- If you send requests to read data from the same `partition group`, you can read data across partitions and strong consistency is supported. When data is migrated at the backend of ApsaraDB for OceanBase, partitions in the same `partition group` may be distributed across servers for a short period of several milliseconds. If this occurs, strong consistency cannot be ensured when you read data across partitions.

5.11.2. INSERT

The INSERT statement allows you to add one or more rows to a specified table.

Syntax

```
INSERT [INTO] tblname
          [(colname,...)]
       {VALUES|VALUE} ({expr | DEFAULT},...)
       [ ON DUPLICATE KEY UPDATE
          colname=expr
        [, colname=expr] ... ] ;
Or
INSERT [INTO] tblname
           [(colname,...)]
      {VALUES|VALUE} (colvalues,...)
[ON DUPLICATE KEY UPDATE
          colname=expr
        [, colname=expr] ... ] ;
```

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
Or:
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
Or:
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

where:

- `[(colname,...)]` specifies the columns into which the data is inserted.
- If you need to insert multiple columns at a time, separate the columns with commas (,).
- The `ON DUPLICATE KEY UPDATE` clause is supported.
- The INSERT statement cannot be followed by SET operations.

`If you execute the INSERT… ON DUPLICATE KEY UPDATE... statement,` the number of `affected rows` is calculated based on the following rules:

- If you do not specify the CLIENT_FOUND_ROWS flag in client_capabilities, the number of affected rows is calculated based on this rule:
  - If a row is inserted as a new row, `affected_row= 1` is returned.
  - If an inserted row conflicts with an existing row in the table and the data in the table remains the same after the update, `affected_row = 0` is returned. If the data in the table before the update is different from that after the update, `affected_row= 2` is returned.
- If you specify the CLIENT_FOUND_ROWS flag, the number of affected rows is calculated based on the following rules:
  - If a row is inserted as a new row, `affected_row=1` is returned.
  - If the data in the table remains the same after the update, `affected_row=1` is returned.
  - If the data in the table before the update is different from that after the update, `affected_row=2` is returned.
- If you do not specify the CLIENT_FOUND_ROWS flag, the value of the affected_row parameter is the number of updated rows. If you specify the CLIENT_FOUND_ROWS flag, the value of the affected_row parameter is the number of touched rows that conflict with existing rows. The data in the touched rows may not be modified.

Examples

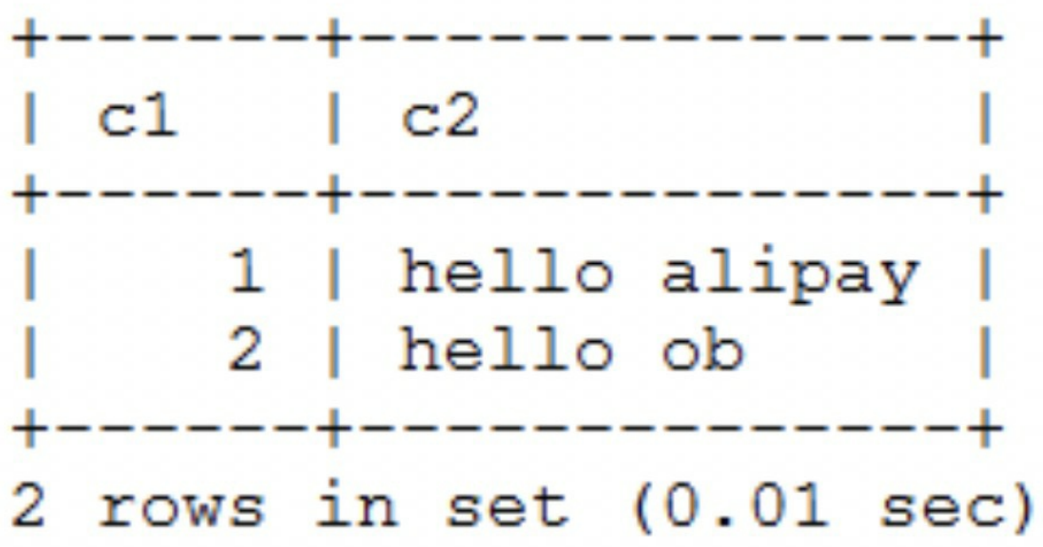
1. Execute the following statement to insert rows:

```
INSERT INTO test VALUES (1, 'hello alipay'),(2, 'hello ob');
```

2. Execute the following statement to view the inserted rows that are shown in Inserted rows:

```
SELECT * FROM test;
```

Inserted rows



```
+-------+---------------+
| c1    | c2            |
+-------+---------------+
|     1 | hello alipay  |
|     2 | hello ob      |
+-------+---------------+
2 rows in set (0.01 sec)
```

3. Execute the following statements to create a partitioned table and insert a row to a partition of the table:

```
mysql> create table employees( id int not null, frame VARCHAR(20),lname VARCHAR(20), hired date not null default '197
0-01-01', separated date not null default '9999-12-31', job_code int, store_id int ) partition by hash(store_id) part
itions 4;
Query OK, 0 rows affected (0.34 sec)
mysql> insert into employees  partition(p2)  values (7,'7','7','1999-02-02' ,'2007-07-07',7,10);
Query OK, 1 row affected (0.00 sec)
```

Errors

1. If an SQL syntax error occurs, the system returns the `1064` error.

```
mysql> insert employees into values employees(1, '1','1',2003-03-03,2003-03-03,1,1);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server vers
ion for the right syntax to use near 'into values employees(1, '1','1',2003-03-03,2003-03-03,1,1)' at line 1
```

2. If the type of the specified data is inconsistent with the data type that is defined in the table, the system returns the `1366` error. For example, if the data type that is defined in a table is INT and the type of the specified data is VARCHAR, the system returns this error. The 1366 error code indicates the error that the type of the specified data is invalid.

```
mysql> desc employees ;
+-----------+-------------+------+-----+------------+-------+
| Field     | Type        | Null | Key | Default    | Extra |
+-----------+-------------+------+-----+------------+-------+
| id        | int(11)     | NO   |     | NULL       |       |
| frame     | VARCHAR(20) | YES  |     | NULL       |       |
| lname     | VARCHAR(20) | YES  |     | NULL       |       |
| hired     | date        | NO   |     | 1970-01-01 |       |
| separated | date        | NO   |     | 9999-12-31 |       |
| job_code  | int(11)     | YES  |     | 1          |       |
| store_id  | int(11)     | YES  |     | NULL       |       |
+-----------+-------------+------+-----+------------+-------+
7 rows in set (0.01 sec)
mysql> insert into employees values (1000, 1,'1','2003-03-03','2003-03-03','test',1);
ERROR 1366 (HY000): Incorrect integer value: 'test' for column 'job_code' at row 1
```

3. If the type of the specified data cannot be recognized, the system returns the `1054` error. The 1054 error code indicates unknown column errors.

```
mysql> insert into employees values (1000, 1,'1','2003-03-03','2003-03-03',test,1);
ERROR 1054 (42S22): Unknown column 'test' in 'field list'
```

4. If the format of the specified time is invalid, the system returns the `1292` error. The 1292 error code indicates the error that the date value is invalid.

```
mysql> create table t2(id int, work date);
Query OK, 0 rows affected (0.08 sec)
mysql> insert into t2 values(1,2003-03-03);
ERROR 1292 (22007): Incorrect date value: '1997' for column 'work' at row 1
mysql> insert into t2 values(1,'2003-03-03');
Query OK, 1 row affected (0.02 sec)
mysql> insert into t2 values(1,test);
ERROR 1054 (42S22): Unknown column 'test' in 'field list'
mysql> insert into t2 values(1,2003);
ERROR 1292 (22007): Incorrect date value: '2003' for column 'work' at row 1
mysql> insert into t2 values(1,20030101);
Query OK, 1 row affected (0.01 sec)
mysql> insert into t2 values(1,2003-01-01);
ERROR 1292 (22007): Incorrect date value: '2001' for column 'work' at row 1
```

5. If the specified numeric value is not in the specified range, the system returns the `1264` error.
6. If the length of the specified characters exceeds the maximum length, the system returns the `1406` error.

```
mysql> desc t1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| s1    | int(11) | YES  |     | NULL    |       |
| s2    | char(5) | YES  |     | NULL    |       |
| s3    | float   | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
3 rows in set (0.02 sec)
mysql> insert into t1 values(11111111111111111111111111, '1', 1);
ERROR 1264 (22003): Out of range value for column 's1' at row 1
mysql> insert into t1 values(1,'111111111111111111111111111111111111111111111111', 1.0);
ERROR 1406 (22001): Data too long for column 's2' at row 1
mysql> insert into t1 values(1,'11',11111111111111111111111111111111111111111111111111111111111111111111111.11111);
ERROR 1264 (22003): Out of range value for column 's3' at row 1
```

7. If NULL is inserted into a column that stores only non-null values, the system returns the `1048` error.

```
mysql> create table t4(id int not null, s1 date);
Query OK, 0 rows affected (0.22 sec)
mysql> insert into t4 values(null, 20031010);
ERROR 1048 (23000): Column 'id' cannot be null
```

8. If the name of the specified table does not exist, the system returns the `1146` error.

```
mysql> insert into notaba values(1,2,3);
ERROR 1146 (42S02): Table 'test.notaba' doesn't exist
```

9. If an inserted value of the primary key is the same as an existing value of the primary key, the system returns the `1062` error.

```
mysql> create table t5(id int primary key, is2 int);
Query OK, 0 rows affected (0.11 sec)
mysql> insert into t5 values(1,2),(2,5);
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> insert into t5 values(1,9);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

10. If the number of inserted column values is different from the defined number of column values, the system returns the `1136` error.

```
mysql> insert into t5(id) values(234,10);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
mysql> insert into t5 values(234);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

11. If the names of the partitions into which data is inserted do not exist, the system returns the `1735` error.

```
mysql> insert into employees partition(p5) values(3,1,1,'2001-01-01','2001-10-10',1,1);
ERROR 1735 (HY000): Unknown partition 'p5' in table 'employees'
```

12. If the information about the partitions into which data is inserted is inconsistent with the partition information that is parsed by the corresponding function, the system returns the `1748` error.

```
mysql> show create table employees;
+-----------+---------------------------------------
| Table     | Create Table
------------------------------------------------------
| employees | CREATE TABLE `employees` (
  `id` int(11) NOT NULL,
  `frame` VARCHAR(20) DEFAULT NULL,
  `lname` VARCHAR(20) DEFAULT NULL,
  `hired` date NOT NULL DEFAULT '1970-01-01',
  `separated` date NOT NULL DEFAULT '9999-12-31',
  `job_code` int(11) DEFAULT '1',
  `store_id` int(11) DEFAULT NULL
) DEFAULT CHARSET = utf8mb4 COMPRESSION = 'lz4_1.0' REPLICA_NUM = 1 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE TABLE
T_SIZE = 134217728 PCTFREE = 10 partition by hash(store_id) partitions 4 |
+-----------+---------------------------------------
1 row in set (0.01 sec)
mysql> insert into employees partition(p2) values(3,1,1,'2001-01-01','2001-10-10',1,1);
ERROR 1748 (HY000): Found a row not matching the given partition set
```

13. If the inserted data is not in the data range that is defined in the partitions, the system returns the following error: `ERROR 1526 (HY000): Table has no partition for value XX` .

```
mysql> CREATE TABLE employees3 (
    ->        id INT NOT NULL,
    ->        fname VARCHAR(30),
    ->        lname VARCHAR(30),
    ->        hired DATE NOT NULL DEFAULT '1970-01-01',
    ->        separated DATE NOT NULL DEFAULT '9999-12-31',
    ->        job_code INT NOT NULL,
    ->        store_id INT NOT NULL
    ->    )
    ->    PARTITION BY RANGE (store_id) (
    ->        PARTITION ptest VALUES LESS THAN (6),
    ->        PARTITION ptestk VALUES LESS THAN (11),
    ->        PARTITION ptestl VALUES LESS THAN (16),
    ->        PARTITION ptestf VALUES LESS THAN (21)
    ->    );
Query OK, 0 rows affected (0.54 sec)
mysql> insert into employees3 values(3,'Meagge','simith','2011-01-10','2019-10-01',10,21);
ERROR 1526 (HY000): Table has no partition for value 21
mysql> insert into employees3 values(4,'Meery','simith','2012-01-10','2019-10-01',10,22);
ERROR 1526 (HY000): Table has no partition for value 22
```

14. If the inserted column values violate the unique index constraint, the system returns the following error: `ERROR 1062 (23000): Duplicate entry 'XXX' for key 'XXX'` .

```
mysql> create table estest(c1 int primary key, c2 int, c3 int, c4 int);
Query OK, 0 rows affected (0.28 sec)
mysql> insert into estest values(1, 2, 3, 5), (2, 3, 4, 5), (5, 6, 7, 8);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> create unique index uniindex on estest(c2); //The unique index uniindex is created.
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> select * from estest;
+----+------+------+------+
| c1 | c2   | c3   | c4   |
+----+------+------+------+
|  1 |    2 |    3 |    5 |
|  2 |    3 |    4 |    5 |
|  5 |    6 |    7 |    8 |
+----+------+------+------+
3 rows in set (0.00 sec)
mysql> insert into estest values(7,6,8,9); //The value 6 in the inserted row is the same as an existing value that is
stored in the unique index column. This violates the constraint of the unique index.
ERROR 1062 (23000): Duplicate entry '6' for key 'uniindex'
```

5.11.3. REPLACE

Syntax

The REPLACE statement is executed in a similar way to the INSERT statement. Assume that a `primary key` value or a value of the unique index column for an existing row is the same as that for the new row. In this scenario, the REPLACE statement deletes the existing row before the statement inserts the new row. This is the only difference between the REPLACE statement and the INSERT statement.

To execute the REPLACE statement, you must have INSERT and DELETE permissions on the specified table.

Syntax

```
REPLACE [INTO] tblname
    [(colname,...)]
    {VALUES|VALUE} ({expr | DEFAULT},...) ;
```

The following code block describes the MySQL syntax of the REPLACE statement:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
Or:
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    SET col_name={expr | DEFAULT}, ...
Or:
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
```

where:

- `[(colname,...)]` specifies the columns into which the data is inserted.
- If you need to replace data in multiple columns at a time, separate the columns with commas (,).

If you execute the REPLACE statement, the number of `affected rows` is calculated based on the following rules:

- If a row is inserted as a new row, `affected_row=1` is returned.
- If the data in the table before the replacement is different from that after the replacement, `affected_row=2` is returned. In this scenario, only one existing row has the same primary key value or the same unique index value as the new row. If multiple existing rows have the same primary key value or unique index value as the new row, the number of `affected rows` equals the number of conflicting rows plus 1.

- If the data remains the same after the replacement, the number of affected rows are calculated based on the following rules:
  - If a conflicting row is caused by the unique index in the table, the foreign key constraint is not used, and `ON DELETE TRIGGER` is not added to the REPLACE statement, `affected_row=1` is returned.
  - In other scenarios, `affected_row = 2` is returned.

Examples

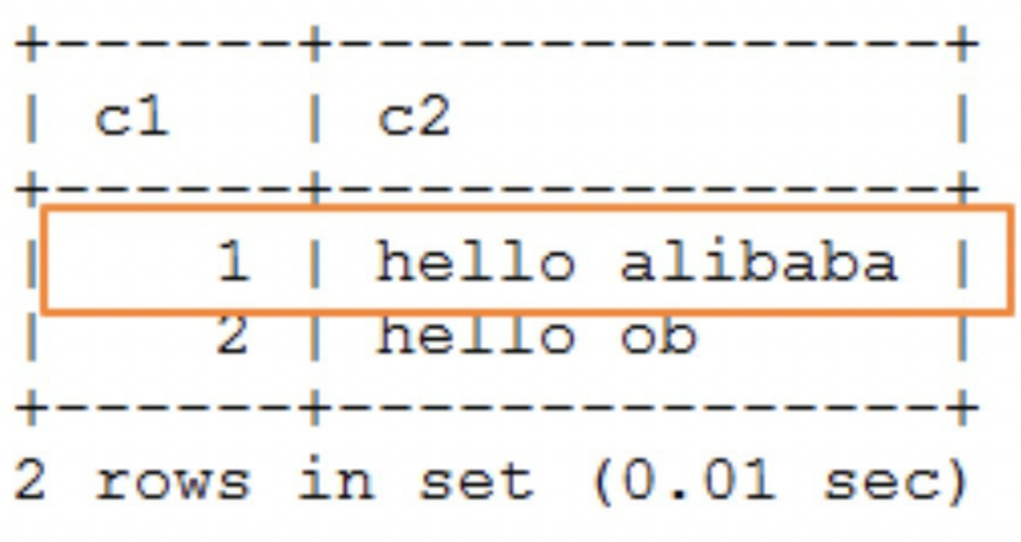
1. Execute the following statement to replace the specified rows in the test table:

```
REPLACE INTO test VALUES (1, 'hello alibaba'),(2, 'hello ob');
```

2. Execute the following statement to view the inserted rows that are shown in the following figure:

```
SELECT * FROM test;
```

Replace the table rows



```
+-------+-----------------+
| c1    | c2              |
+-------+-----------------+
|     1 | hello alibaba   |
|     2 | hello ob        |
+-------+-----------------+
2 rows in set (0.01 sec)
```

Errors

1. If an SQL syntax error occurs, the system returns the `1064` error.
2. If the type of the specified data is inconsistent with the data type that is defined in the table, the system returns the `1366` error. For example, if the data type that is defined in a table is INT and the type of the specified data is VARCHAR, the system returns this error. The 1366 error code indicates the error that the type of the specified data is invalid.
3. If the type of the specified data cannot be recognized, the system returns the `1054` error. The 1054 error code indicates unknown column errors.
4. If the format of the specified time is invalid, the system returns the `1292` error. The 1292 error code indicates the error that the date value is invalid.
5. If the specified numeric value is not in the specified range, the system returns the `1264` error.
6. If the length of the specified characters exceeds the maximum length, the system returns the `1406` error.
7. If NULL is inserted into a column that stores only non-null values, the system returns the `1048` error.
8. If the name of the specified table does not exist, the system returns the `1146` error.
9. If the number of inserted column values is different from the defined number of column values, the system returns the `1136` error.
10. If the names of the partitions into which data is inserted do not exist, the system returns the `1735` error.
11. If the information about the partitions into which data is inserted is inconsistent with the partition information that is parsed by the corresponding function, the system returns the `1748` error.

5.11.4. UPDATE

The UPDATE statement allows you to change field values in a specified table.

Syntax

Syntax

```
UPDATE tblname
SET colname=colvalues
       [, colname=colvalues...]
[WHERE where_condition]
[ORDER BY order_list]
[LIMIT row_count];
order_list:
 colname [ASC|DESC] [, colname [ASC|DESC]…]
```

The following code block describes the MySQL syntax of the UPDATE statement:

```
Single-table syntax:
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
Multiple-table syntax:
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
```

Examples

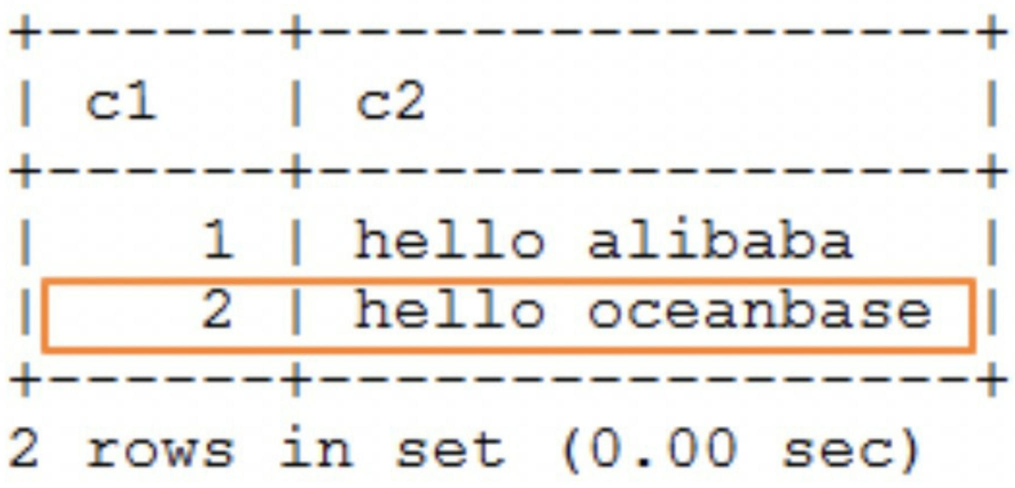
1. Execute the following statement to change `hello ob` to `hello oceanbase` :

   ```
   UPDATE test SET c2 ='hello oceanbase' WHERE c1=2;
   ```

2. Execute the following statement to view the changed information that is shown in Update the test table:

   ```
   SELECT * FROM test;
   ```

   Update the test table

   ```
   +------+-------------------+
   | c1   | c2                |
   +------+-------------------+
   |    1 | hello alibaba     |
   |    2 | hello oceanbase   |
   +------+-------------------+
   2 rows in set (0.00 sec)
   ```

3. Update a column value in a partition.

   Execute the following statements to change the value of job_code in the p0 partition of the employees table to 1:

```
mysql> select * from employees;
+----+-------+-------+------------+------------+----------+----------+
| id | frame | lname | hired      | separated  | job_code | store_id |
+----+-------+-------+------------+------------+----------+----------+
|  4 | 4     | 4     | 2000-04-04 | 2044-04-04 |        4 |        4 |
|  1 | 1     | 1     | 2000-01-01 | 2014-01-01 |        1 |        1 |
|  5 | 5     | 5     | 2000-05-05 | 2022-05-05 |        5 |        5 |
|  2 | 2     | 2     | 2000-02-02 | 2024-02-02 |        2 |        2 |
|  6 | 6     | 6     | 2000-06-06 | 2022-06-06 |        6 |        6 |
|  7 | 7     | 7     | 1999-02-02 | 2007-07-07 |        7 |       10 |
|  3 | 3     | 3     | 2000-03-03 | 2034-03-03 |        3 |        3 |
+----+-------+-------+------------+------------+----------+----------+
7 rows in set (0.05 sec)
mysql> update employees partition (p0)  set job_code=1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Errors

1. If an SQL syntax error occurs, the system returns the `1064` error.
2. If the type of the specified data is inconsistent with the data type that is defined in the table, the system returns the `1366` error. For example, if the data type that is defined in a table is INT and the type of the specified data is VARCHAR, the system returns this error. The 1366 error code indicates the error that the type of the specified data is invalid.
3. If the type of the specified data cannot be recognized, the system returns the `1054` error. The 1054 error code indicates unknown column errors.
4. If the format of the specified time is invalid, the system returns the `1292` error. The 1292 error code indicates the error that the date value is invalid.
5. If the specified numeric value is not in the specified range, the system returns the `1264` error.
6. If the length of the specified characters exceeds the maximum length, the system returns the `1406` error.
7. If NULL is inserted into a column that stores only non-null values, the system returns the `1048` error.
8. If the name of the specified table does not exist, the system returns the `1146` error.
9. If an inserted value of the primary key is the same as an existing value of the primary key, the system returns the `1062` error.
10. If the names of the partitions where data is updated do not exist, the system returns the `1735` error.
11. If the information about the partitions where data is updated is inconsistent with the partition information that is parsed by the corresponding function, the system returns the `1748` error.

5.11.5. DELETE

The DELETE statement allows you to delete the table rows that meet the specified conditions.

Syntax

```
DELETE FROM tblname
       [WHERE where_condition]
       [ORDER BY order_list]
       [LIMIT row_count];
```

The following code block describes the MySQL syntax of the DELETE statement:

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

Examples

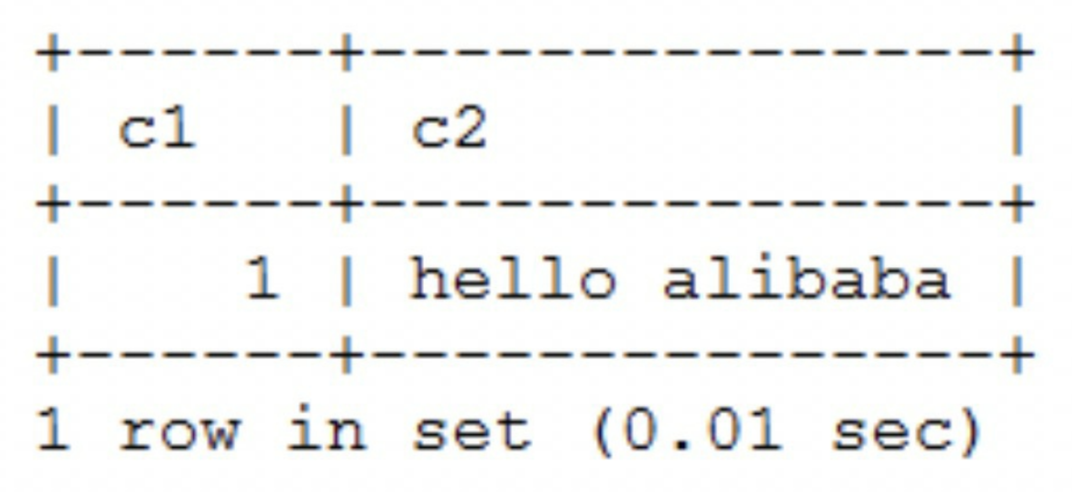
1. Execute the following statement to delete the row that meets the `c1=2` condition. The c1 column is the primary key in the test table.

```
DELETE FROM test WHERE c1 = 2;
```

2. Execute the following statement to view the table information after the row is deleted. Delete the row shows the output of the statement.

```
SELECT * FROM test;
```

Delete the row



```
+------+----------------+
| c1   | c2             |
+------+----------------+
|    1 | hello alibaba  |
+------+----------------+
1 row in set (0.01 sec)
```

Errors

1. If an SQL syntax error occurs, the system returns the `1064` error.
2. If the type of the specified data cannot be recognized, the system returns the `1054` error. The 1054 error code indicates unknown column errors.
3. If the name of the specified table does not exist, the system returns the `1146` error.
4. If the names of the partitions into which data is inserted do not exist, the system returns the `1735` error.

5.11.6. SELECT

The SELECT statement allows you to query data that is stored in tables.

Basic queries

Syntax

```
SELECT
       [ALL | DISTINCT]
     selectexpr [[AS] othername] [, selectexpr ...]
       [FROM table_references ]
[PARTITION(partitionid [, partitionid…])]
     [WHERE where_conditions]
       [GROUP BY group_by_list]
       [HAVING search_confitions]
       [ORDER BY order_list]
       [LIMIT {[offset,] row_count | row_count OFFSET offset}]
     [FOR UPDATE];
```

The following code block describes the MySQL syntax of the SELECT statement:

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

describes SELECT clauses.

SELECT clauses

Clause	Description
ALL | DISTINCT	Specifies whether the statement returns only distinct rows. A database table may contain duplicate values. • If you add `DISTINCT` to the statement, only distinct rows are returned in the query results. • If you add `ALL` to the statement, all the matched rows are returned. • If you do not add DISTINCT or ALL to the statement, the default setting `ALL` is used.
select_expr	Specifies the required expressions or column names. The expressions specify the columns that you want to retrieve. If you need to specify multiple expressions or column names, separate the specified expressions or column names with commas (,). You can use an asterisk (*) to indicate that all the columns in the table are returned.
AS *othername*	Renames output fields.
FROM *table_references*	• Specifies the tables from which data is read. • Allows you to retrieve data from multiple tables.
WHERE*where_conditions*	Specifies the filter conditions. The query results contain only the data that meets the filter conditions. This clause is optional. *where_conditions* specifies an expression.
GROUP BY *group_by_list*	Divides data into groups.
HAVING *search_confitions*	Specifies the filter conditions. HAVING clauses are similar to WHERE clauses. The difference between HAVING and WHERE clauses is that you can use aggregate functions in HAVING clauses, such as SUM and AVG.

Clause	Description
ORDER BY *order_list* *order_list* : *colname* [ASC | DESC] [,*colname* [ASC | DESC]...]	Displays the query results in ascending or descending order. ASC indicates the ascending order and DESC indicates the descending order. If you do not specify the order, the default order ASC is used.
[LIMIT {[*offset*,] *row_count* |*row_count* OFFSET *offset*}]	Limits the number of rows that are returned by the SELECT statement. You can specify one or two arguments of the numeric data type for the LIMIT clause. The arguments must be integer constants. • If you specify two arguments, the first argument specifies the offset for the first row to be returned and the second argument specifies the maximum number of rows to be returned. The initial offset for the first row is 0 instead of 1. • If you specify only one argument, the argument specifies the maximum number of rows to be returned and the offset is 0.
FOR UPDATE	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.
PARTITION(*partition_list*) Format: partition(p0,p1...)	Specifies the partition information of the specified tables.

JOIN syntax

Joins are divided into inner joins and outer joins. Outer joins are divided into left joins, right joins, and full joins. After two tables are joined, you can use the ON clause to filter the data in the tables.

In ApsaraDB for OceanBase, you can use USING clauses for joins. At least one of the join conditions must use the equal to (=) operator.

For inner joins, the query results contain only the matched rows in both tables.

For left joins, the query results contain all the rows in the table on the left of the `LEFT [OUTER] JOIN` keyword, and the matched rows in the table on the right of the keyword.

For right joins, the query results contain all the rows in the table on the right of the `[RIGHT] [OUTER] JOIN` keyword, and the matched rows in the table on the left of the keyword.

For full joins, the query results contain all the rows in both tables.

Set operations

In ApsaraDB for OceanBase, the main clauses for set operations are UNION, EXCEPT, and INTERSECT clauses.

- UNION clause

  The UNION clause combines the result sets of two or more SELECT statements. To use the UNION clause, pay attention to the following considerations:

  - For the UNION clause, the number of columns, the data types of the columns, and the column sequence for each of the SELECT statements must be the same.
  - By default, the UNION operator returns result sets that do not contain duplicate values. If you need to obtain duplicate values, use the `UNION ALL` clause.
  - The column names in each UNION result set are always the same as those in the first SELECT statement for the UNION clause.

  The UNION clause combines the results of two or more SELECT statements. The JOIN clause retrieves data from two or more tables. Therefore, the UNION clause is similar to the JOIN clause. The difference between the two clauses is that the UNION clause combines the results of two or more SELECT statements and the JOIN clause joins two or more tables.

- EXCEPT clause

  The EXCEPT clause returns the data that is included in the result set of the first SELECT statement but is excluded from the result set of the second SELECT statement.

- INTERSECT clause

  The INTERSECT clause returns the data that is included in the result sets of both SELECT statements.

DUAL virtual table

DUAL is a virtual table. The DUAL virtual table can be considered as a special table that has one row and no columns. If you do not need to retrieve data from specific tables but need to execute the SELECT statement to retrieve some required information, DUAL helps you retrieve the required information. You can use the SELECT syntax that includes DUAL to retrieve the values of user variables or system variables.

If the SELECT statement does not include the FROM clause, `FROM DUAL` is equivalent to the FROM clause. In this scenario, the expressions in the SELECT statement can be only constant expressions.

Syntax

```
SELECT
       [ALL | DISTINCT]
       select_list
       [FROM DUAL [WHERE where_condition]]
    [LIMIT {[offset,] rowcount | rowcount OFFSET offset}];
```

`SELECT… FOR UPDATE` statement

Syntax

```
SELECT ... FOR UPDATE [WAIT n| NOWAIT];
```

where:

- The WAIT clause specifies the number of seconds to wait. The current user must wait for the specified number of seconds before another user releases the row locks. This prevents endless waiting.
- NOWAIT indicates that the current user does not wait for the row locks to release.

You can execute the `SELECT … FOR UPDATE` statement to apply an exclusive lock on each row in the query results. This prevents other transactions from concurrently updating these rows. This also prevents other transactions from concurrently reading the rows for which some transaction isolation levels are specified. To be more specific, you can use the `FOR UPDATE` clause to lock the tuples of the query results. In this scenario, the `UPDATE` , `DELETE` and `FOR UPDATE` operations cannot be performed on these tuples before the transaction is committed.

Notice In ApsaraDB for OceanBase, you can run each query on only a single table.

Example:

```
SELECT * FROM a FOR UPDATE;
```

IN and OR logical operators

ApsaraDB for OceanBase supports `IN` and `OR` logical operators.

Errors

1. If an SQL syntax error occurs, the system returns the `1064` error.
2. If the type of the specified data cannot be recognized, the system returns the `1054` error. The 1054 error code indicates unknown column errors.
3. If the name of the specified table does not exist, the system returns the `1146` error.
4. If the names of the partitions from which data is retrieved do not exist, the system returns the `1735` error.
5. If a SELECT subquery returns multiple rows, the system returns the `1241` error. The 1241 error code indicates that the result violates the rule: The subquery results can contain only one field.

```
mysql> select * from employees where store_id=5;
+----+-------+-------+------------+------------+----------+----------+
| id | frame | lname | hired      | separated  | job_code | store_id |
+----+-------+-------+------------+------------+----------+----------+
|  5 | 5     | 5     | 2000-05-05 | 2022-05-05 |      123 |        5 |
+----+-------+-------+------------+------------+----------+----------+
1 row in set (0.00 sec)
mysql> select (select id from employees where store_id=5 );
+----------------------------------------------+
| (select id from employees where store_id=5 ) |
+----------------------------------------------+
|                                            5 |
+----------------------------------------------+
1 row in set (0.00 sec)
mysql> select (select id,frame from employees where store_id=5 );
ERROR 1241 (21000): Operand should contain 1 column(s)
```

6. If the referenced group functions are invalid, the system returns the `1111` error.

```
mysql> CREATE TABLE t222 (s1 INT,  s2 INT,  s3 INT,  PRIMARY KEY(s3) );
mysql> insert into t222 values(1, 1, 1), (1, 2, 2), (1, 2, 3);
mysql> select avg(sum(s1)) from t222 group by s1;
ERROR 1111 (HY000): Invalid use of group function
```

```
mysql> select avg(a1) from (select sum(s1) as a1 from t222 group by s1);
```

5.12. Transaction language

A database transaction is a single logical unit of work that consists of a collection of operations.

Transaction processing ensures that SQL operations in a batch are all executed or are not executed at all. You can use transactions to maintain the data integrity of databases.

An explicit transaction is a user-defined or user-specified transaction. An explicit transaction is a transaction that starts with the `BEGIN TRANSACTION`, `BEGIN`, or `BEGIN WORK` statement and ends with the COMMIT or ROLLBACK statement. BEGIN and BEGIN WORK are supported as aliases of the `START TRANSACTION` statement.

To start a transaction, use the following statement syntax:

```
START TRANSACTION
    [WITH CONSISTENT SNAPSHOT];
BEGIN [WORK] ;
COMMIT [WORK] ;
ROLLBACK [WORK];
```

ApsaraDB for OceanBase supports only the `READ COMMITTED` isolation level.

- The `WITH CONSISTENT SNAPSHOT` clause starts a consistent read. This clause has the same effect as issuing a `START TRANSACTION` statement that is followed by a SELECT statement that queries an ApsaraDB for OceanBase table.

  You can specify the `WITH CONSISTENT SNAPSHOT` clause in the START TRANSACTION statement. However, the `WITH CONSISTENT SNAPSHOT` does not take effect.

- `BEGIN` and `BEGIN WORK` are supported as aliases of `START TRANSACTION` and are used to initialize a transaction. `START TRANSACTION` is a standard SQL syntax and is the recommended way to start an ad hoc transaction. After a transaction is started, the SQL statements that follow the START TRANSACTION statement such as INSERT, UPDATE, and DELETE take effect only when the transaction is explicitly committed. However, a REPLACE statement is not subject to this limit.

To commit a transaction, use the following statement syntax:

```
COMMIT [WORK];
```

To roll back a transaction, use the following statement syntax:

```
ROLLBACK [WORK];
```

Example:

Perform a transaction on Table A: Find the row whose id is 3, and change the value for the name column to c, and insert a new row about sale records for product a.

Table A

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43
2	b	200	2013-06-21 13:07:21
3	a	50	2013-06-21 13:08:15

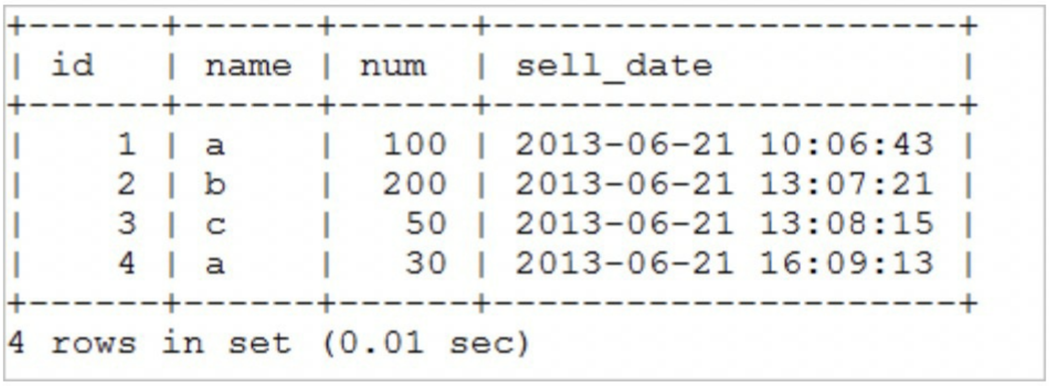
1. Execute the following statements in sequence to start the transaction.

```
START TRANSACTION;
UPDATE a SET name = 'c' WHERE id = 3;
INSERT INTO a VALUES (4, 'a', 30, '2013-06-21 16:09:13');
COMMIT;
```

2. After the transaction is committed, execute the following statement to query data of Table A. Table A information shows the execution result.

```
SELECT * FROM a;
```

Table A information



```
+------+------+------+---------------------+
| id   | name | num  | sell_date           |
+------+------+------+---------------------+
|    1 | a    |  100 | 2013-06-21 10:06:43 |
|    2 | b    |  200 | 2013-06-21 13:07:21 |
|    3 | c    |   50 | 2013-06-21 13:08:15 |
|    4 | a    |   30 | 2013-06-21 16:09:13 |
+------+------+------+---------------------+
4 rows in set (0.01 sec)
```

Note

Before you commit a transaction, you can check whether the operations in the transaction have taken effect. For example, you can insert a `SELECT * FROM a;` statement before the COMMIT clause.

The session within which this transaction is executed can read the updated result. A session outside this transaction cannot read the updated result. Before the transaction is committed, your previous operations are invisible outside the transaction session. To roll back a transaction, execute the ROLLBACK statement to undo the COMMIT operation.

5.13. Database management language

5.13.1. CREATE RESOURCE UNIT

```
CREATE RESOURCE UNIT unitname
MAX_CPU [=] cpunum,
MAX_MEMORY [=] memsize,
MAX_IOPS [=] iopsnum,
MAX_DISK_SIZE [=] disksize,
MAX_SESSION_NUM [=] sessionnum,
[MIN_CPU [=] cpunum,]
[MIN_MEMORY [=] memsize,]
[MIN_IOPS [=] iopsnum] ;
```

When you create a resource unit, you must specify the values of MAX_CPU, MAX_MEMORY, MAX_IOPS, MAX_DISK_SIZE, and MAX_SESSION_NUM. MIN_CPU, MIN_MEMORY, and MIN_IOPS are optional. The default values of MIN_CPU, MIN_MEMORY, and MIN_IOPS are the same as the default values of MAX_CPU, MAX_MEMORY, and MAX_IOPS.

- The value range of MAX_MEMORY is [1073741824, +∞). The unit is byte. The minimum value is 1 GB.
- The value range of MAX_IOPS is [128, +∞).
- The value range of MAX_DISK_SIZE is [536870912, +∞]. The unit is byte. The minimum value is 512 MB.
- The value range of MAX_SESSION_NUM is [64, +∞).

You can replace *memsize* or *disksize* with a value that is a combination of a number and a unit, for example, 1 GB or 100 MB. You can also specify the value in bytes, for example, 1073741824 or 104857600.

For example, the following two statements are equivalent:

```
mysql> CREATE RESOURCE UNIT unit1 max_cpu 1, max_memory '1G', max_iops 128,max_disk_size '10G', max_session_num 64, MIN_C
PU=1, MIN_MEMORY= '1G', MIN_IOPS=128;
Query OK, 0 rows affected (0.02 sec)
```

Equivalent to:

```
mysql> CREATE RESOURCE UNIT unit1 max_cpu 1, max_memory 1073741824, max_iops 128, max_disk_size 10737418240, max_session_
num 64, MIN_CPU=1, MIN_MEMORY=1073741824, MIN_IOPS=128;
Query OK, 0 rows affected (0.01 sec)
```

5.13.2. ALTER RESOURCE UNIT

```
ALTER RESOURCE UNIT unitname
MAX_CPU [=] cpunum,
MAX_MEMORY [=] memsize,
MAX_IOPS [=] iopsnum,
MAX_DISK_SIZE [=] disksize,
MAX_SESSION_NUM [=] sessionnum,
[MIN_CPU [=] cpunum,]
[MIN_MEMORY [=] memsize,]
[MIN_IOPS [=] iopsnum] ;
```

When you modify the properties of a resource unit, follow the same configuration rules that are described in the `CREATE RESOURCE UNIT` topic.

5.13.3. DROP RESOURCE UNIT

```
DROP RESOURCE UNIT unitname
```

You can execute this statement to delete a resource unit.

For example, execute the following statement to delete unit1:

```
mysql> DROP RESOURCE UNIT unit1;
Query OK, 0 rows affected (0.00 sec)
```

5.13.4. CREATE RESOURCE POOL

Syntax

```
CREATE RESOURCE POOL poolname
UNIT [=] unitname,
UNIT_NUM [=] unitnum,
ZONE_LIST [=] (‘zone’ [, ‘zone’ …]) ;
```

A resource pool contains multiple resource units. You must specify the zones to which the resource units belong.

ApsaraDB for OceanBase allows only one type of resource units in a resource pool. UNIT_NUM indicates the number of resource units in a zone. The UNIT_NUM value must be smaller than the number of OBServers in the zone.

Examples

```
mysql> CREATE RESOURCE POOL pool1 unit='unit1', unit_num=1, zone_list=('zone1');
Query OK, 0 rows affected (0.01 sec)
```

5.13.5. ALTER RESOURCE POOL

Syntax

```
ALTER RESOURCE POOL poolname
UNIT [=] unitname,
UNIT_NUM [=] unitnum,
ZONE [=] ('zone' [, 'zone' …]) ;
```

You can execute this statement to modify the properties of a resource pool.

If your modification results in no resource unit in the resource pool, make sure that the resource pool is not used by a tenant when you execute the `ALTER RESOURCE POOL` statement. If the resource pool is in use, the system returns an error.

When you modify the properties of a resource pool by executing the `ALTER RESOURCE POOL` statement, you can modify only one property at a time. To modify two or more of the UNIT, UNIT_NUM, and ZONE properties, you must execute the ALTER RESOURCE POOL statement two or more times.

Examples

```
//If you modify multiple properties of a resource pool at a time, the system returns an error.
mysql> ALTER RESOURCE POOL pool1 unit='unit2', unit_num=1, zone_list=('zone1');
ERROR 1235 (0A000): alter unit_num, resource_unit, zone_list in one cmd not supported
//Modify one property at a time.
mysql> ALTER RESOURCE POOL pool1 unit='unit2';
Query OK, 0 rows affected (0.00 sec)
```

5.13.6. DROP RESOURCE POOL

Syntax

```
DROP RESOURCE POOL poolname;
```

You can execute this statement to delete a resource pool.

Examples

Execute the following statement to delete pool1:

```
mysql> DROP RESOURCE POOL pool1;
Query OK, 0 rows affected (0.00 sec)
```

5.13.7. CREATE TENANT

Syntax

```
CREATE TENANT [IF NOT EXISTS] tenantname
     [tenant_characteristic_list]
tenant_characteristic_list:
tenant_characteristic [, tenant_characteristic...]
tenant_characteristic:
COMMENT 'string'
|{CHARACTER SET | CHARSET} [=] charsetname
|COLLATE [=]  collationname
|REPLICA_NUM [=] num
|ZONE_LIST [=] (zone [, zone…])
|PRIMARY_ZONE [=] zonelist
|DEFAULT TABLEGROUP [=] {NULL | tablegroup}
|RESOURCE_POOL_LIST [=](poolname [, poolname…])
      |LOCALITY [=] locality
```

If the specified tenant name is already used and the `IF NOT EXISTS` option is not specified, the system returns an error.

The validity requirements for tenant names are the same as those for variable names. A tenant name must be up to 64 bytes in length and can contain only letters, digits, and underscores (_). The name must start with a letter or an underscore (_) and cannot be a keyword that is reserved for ApsaraDB for OceanBase.

Before you execute the `CREATE TENANT` statement to create a tenant, you must connect your root user to the root tenant (root@ROOT).

Note

You must specify `RESOURCE_POOL_LIST` when you create a tenant.

When you specify `RESOURCE_POOL_LIST` for the `CREATE TENANT` statement, only one `resource pool` is supported.

`DEFAULT TABLEGROUP` specifies a default table group for the tenant. If you do not specify this parameter, the value is NULL.

Examples

```
mysql> CREATE TENANT IF NOT EXISTS t1 charset='utf8mb4', replica_num=1, zone_list=('zone1'), primary_zone='zone1', resour
ce_pool_list=('pool1');
Query OK, 0 rows affected (0.26 sec)
```

Errors

- If your statement has a syntax error, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .
- If the specified tenant name is already used and the `IF NOT EXISTS` option is not specified, the system returns the following error: `ERROR(OB_ERR_ALREADY_EXISTS, -5025, -1, "42S01", "Already exist");` .

5.13.8. ALTER TENANT

Syntax

```
ALTER TENANT tenantname [SET] [tenant_options]
tenant_options:
 tenant_option [ tenant_option...]
tenant_options:
COMMENT [=]'string'
|{CHARACTER SET | CHARSET} [=] charsetname
|COLLATE [=]  collationname
|REPLICA_NUM [=] num
|ZONE_LIST [=] (zone [, zone…])
|PRIMARY_ZONE [=] zonelist
|RESOURCE_POOL_LIST [=](poolname [, poolname…])
|DEFAULT TABLEGROUP [=] {NULL | tablegroupname}
|{READ ONLY | READ WRITE}
      | LOCALITY [=] locality
```

Note You must specify `RESOURCE_POOL_LIST` when you create a tenant.

The system tenant users and the administrator of the current tenant have the permission to execute the `ALTER TENANT` statement.

`DEFAULT TABLEGROUP` specifies a default table group for the tenant. A NULL value indicates no default table group for the database.

Errors

If your statement has a syntax error, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .

5.13.9. Lock or unlock a tenant

Syntax

```
ALTER TENANT tenantname  LOCK|UNLOCK;
```

You can execute this statement to lock a tenant. After the tenant is locked, you cannot create sessions on the tenant. The existing sessions remain unchanged. You can lock a tenant if the subscription of the service is not renewed upon expiration. After the subscription is renewed, you can unlock the tenant.

Examples

- Execute the following statement to lock TENANT1:

  ```
  ALTER TENANT TENANT1 LOCK;
  ```

- Execute the following statement to unlock TENANT1:

  ```
  ALTER TENANT TENANT1 UNLOCK;
  ```

Errors

If your statement has a syntax error, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .

5.13.10. DROP TENANT

Syntax

```
DROP TENANT tenantname;
```

`DROP` You can execute this statement to delete an ApsaraDB for OceanBase tenant.

Before you execute the DROP TENANT statement to delete a tenant, you must connect your root user to the root tenant (root@ROOT).

You can delete a tenant only if the tenant is in the Locked state. If you execute the DROP TENANT statement to delete an unlocked tenant, the system returns an error.

Examples

Execute the following statement to delete TENANT1:

```
DROP TENANT TENANT1;
```

Errors

If your statement has a syntax error, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .

5.13.11. CREATE TABLEGROUP

Syntax

```
CREATE TABLEGROUP [IF NOT EXISTS] tablegroupname
```

If the specified table group name is already used and the `IF NOT EXISTS` option is not specified, the system returns an error.

A table group name must be up to 64 characters in length and can contain letters, digits, and underscores (_). The name must start with a letter or an underscore (_) and cannot be a keyword that is reserved for ApsaraDB for OceanBase.

Only the tenant administrator can create a table group.

Examples

```
CREATE TABLEGROUP myTableGroup1;
```

5.13.12. DROP TABLEGROUP

Syntax

```
DROP TABLEGROUP [IF EXISTS] tablegroupname
```

Examples

```
DROP TABLEGROUP myTableGroup1;
```

Errors

- If your statement has a syntax error, the system returns the following error: `ERROR 1064 (42000): You have an error in your SQL syntax;` .
- Before you delete a table group, make sure that no object such as a table or an index is using the table group. If the table group is still in use, the system returns an error.

5.13.13. ALTER TABLEGROUP

Syntax

```
ALTER TABLEGROUP tablegroupname ADD [TABLE] tblname [, tblname…]
```

You can execute this statement to add multiple tables to a table group. Table names are separated with commas (,).

If you add multiple tables at a time, you can specify duplicate table names. If a table to be added already exists in the table group that is specified by `tablegroupname` , the system does not return an error.

5.13.14. Users and permissions

You can manage users and permissions on databases. These management operations include creating a user, deleting a user, changing a password, changing a username, locking a user, granting permissions, and revoking permissions.

ApsaraDB for OceanBase users are divided into two types: users of the system tenant and users of general tenants.

When you create a user, if the current session is created on the system tenant, the new user is created for the system tenant. If the current session is created on a general tenant, the new user is created for the general tenant.

Usernames are unique within a tenant. Users under different tenants can have the same name. The Username@TenantName must be globally unique in the cluster. Usernames of users of the system tenant have a predetermined prefix. You can distinguish users of the system tenant from users of general tenants based on this characteristic. The system tenant or each general tenant has a built-in root user. The root user for a system tenant is the system administrator, and the root user for a general tenant is the tenant administrator. If you purchase a general tenant, you have the permissions to use the root user and the password of the general tenant to manage resources within the tenant.

Users of a general tenant can access only objects in the general tenant. This is the same as the logic of MySQL. Users of the system tenant can access objects that belong to different tenants. Users of the system tenant cannot access user tables that are stored in general tenants. When you log on to the ApsaraDB for OceanBase system, you must specify a unique tenant name. If you are using a user of the system tenant, you can execute the `CHANGE EFFECTIVE TENANT tenantname` statement to access another tenant after you log on to the system. However, if you are using a user of a general tenant, you cannot switch the tenant.

Create users

You can execute the `CREATE USER` statement to create an ApsaraDB for OceanBase user.

After a user is created, you can use the user to connect to ApsaraDB for OceanBase.

Syntax

```
CREATE USER user_specification_list;
user_specification_list:
    user_specification [, user_specification]…;
user_specification:
    user IDENTIFIED BY 'authstring'
    user IDENTIFIED BY PASSWORD 'hashstring'
```

Notes:

- To execute the `CREATE USER` statement, you must have the global `CREATE USER` permission.
- After a user is created, a new row is added for the user to the *mysql.user* table. If the username is already used by an existing user, the system returns an error.
- You can specify the `IDENTIFIED BY` clause to set the password for your user.
- `user IDENTIFIED BY Specify user IDENTIFIED BY 'authstring'` to set a plaintext password. After the password is saved to the *mysql.user* table, the password is stored in ciphertext on the server.
- Specify `user IDENTIFIED BY PASSWORD 'hashstring'` to set a ciphertext password.
- If you create multiple users at a time, separate pairs of user information with commas (,).

Examples

```
Oceanbase>CREATE USER 'sqluser01' IDENTIFIED BY '123456', 'sqluser02' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.07 sec)
Oceanbase>select user from user;
+-----------+
| user      |
+-----------+
| root      |
| test      |
| sqluser01 |
| sqluser02 |
+-----------+
4 rows in set (0.01 sec)
```

Delete users

You can execute the `DROP USER` statement to delete one or more ApsaraDB for OceanBase users.

Syntax

```
DROP USER username [, username...] ;
```

Notes:

- To execute the `DROP USER` statement, you must have the global `CREATE USER` permission.
- You cannot delete a user by performing a DELETE operation on the *mysql.user* table.
- After a user is deleted, all permissions of the user are deleted.
- If you delete multiple users at a time, separate the usernames with commas (,).

Examples

Execute the following statement to delete the user `sqluser02`:

```
DROP USER 'sqluser02';
```

Change a password

You can change the password for a user that is used to log on to the ApsaraDB for OceanBase system.

Syntax

```
SET PASSWORD [FOR user] = password_option;
password_option: {
    PASSWORD('authstring')
    |'hashstring'}
```

Or

```
ALTER USER username IDENTIFIED BY 'password';
```

Notes:

- If you do not specify the `FOR user` clause in the statement, the system changes the password for the current user. After a user logs on to the ApsaraDB for OceanBase system, the user can change the password.
- If you specify the `FOR user` clause in the statement, or you use the second syntax, the system changes the password for the specified user. To change the password for a specified user, you must have the global `CREATE USER` permission.

Examples

Execute the following ALTER USER statement to change the password of `sqluser01` to `abc123`:

```
ALTER USER sqluser01 IDENTIFIED BY 'abc123';
```

Execute the following `SET PASSWORD` statement to change the password:

```
Oceanbase>set password for test = password('abc123');
Query OK, 0 rows affected (0.03 sec)
# If you do not specify the password function, the system returns the following error:
Oceanbase>set password for test = 'abc123';
ERROR 1827 (42000): The password hash doesn't have the expected format. Check if the correct password algorithm is being
used with the PASSWORD() function.
```

Change a username

You can change the username of a user that is used to log on to the ApsaraDB for OceanBase system.

Syntax

```
RENAME USER
    'oldusername' TO 'newusername'
    [,'oldusername' TO 'newusername'...] ;
```

Notes:

- To execute this statement, you must have the global `CREATE USER` permission.
- If you change multiple usernames at a time, separate the pairs of username information with commas (,).
- After you change the username for a user, the permissions of the user remain unchanged.
- The username must be up to 16 bytes in length.

Examples

```
Oceanbase>select user from user;
+---------+
| user    |
+---------+
| root    |
| testall |
+---------+
2 rows in set (0.00 sec)
```

Change a username.

```
Oceanbase>rename user testall to test;
Query OK, 0 rows affected (0.03 sec)
Oceanbase>select user from user;
+------+
| user |
+------+
| root |
| test |
+------+
2 rows in set (0.00 sec)
```

Lock a user

You can lock or unlock a user. Locked users are not allowed to log on to the ApsaraDB for OceanBase system.

Syntax

```
ALTER USER user [lock_option]
lock_option:{
    ACCOUNT LOCK
    | ACCOUNT UNLOCK}
```

To execute this statement, you must have the global `UPDATE USER` permission.

Examples

- Lock a user.

  ```
  Oceanbase>alter user test account lock;
  Query OK, 0 rows affected (0.04 sec)
  ```

- Unlock a user.

```
Oceanbase>alter user test account unlock;
Query OK, 0 rows affected (0.02 sec)
```

Grant permissions to a user

You can execute the GRANT statement as a system administrator to grant permissions to a user.

Syntax

```
GRANT priv_type
       ON priv_level
     TO user_specification [, user_specification]...
    [WITH with_option ...]
priv_level:
     *
    | *.*
    | db_name.*
    | db_name.tbl_name
    | tbl_name
user_specification:
    user [IDENTIFIED BY [PASSWORD] 'password']
with_option:
     GRANT OPTION
```

Permissions can be divided into the following levels:

- Global permissions

  Global permissions apply to all databases. To grant global permissions, use the `GRANT ALL ON *. *` syntax.

- Database permissions

  Database permissions apply to all objects in a specified database. To grant permissions on a specified database, use the `GRANT ALL ON db_name. *` syntax.

- Table permissions

  Table permissions apply to all columns in a specified table. To grant permissions on a specified table, use the `GRANT ALL ON db_name.tbl_name` syntax.

Notes:

- You must grant permissions to specific users. If a specified user does not exist, the system can create the user if the SQL mode is NO_AUTO_CREATE_USER. You can specify this SQL mode by executing `sql_mode='no_auto_create_user'`. If you do not use the `IDENTIFIED BY` clause to specify the password, the system cannot create the user.
- To grant permissions to a user, you must have the permission that you are granting. For example, if you use user1 to grant user2 the SELECT permission on table t1, user1 must have the SELECT permission on table t1. You must also have the `GRANT OPTION` permission.
- After a permission is granted, the authorized user must relog on to the ApsaraDB for OceanBase system so that the permission can take effect.
- You can use an asterisk (*) instead of specifying a table name to grant permissions on all tables in the specified database.
- If you grant multiple permissions to a user at a time, separate the permission types with commas (,).
- If you grant permissions to multiple users at a time, separate the usernames with commas (,).
- lists the priv_type values that can be specified for the GRANT statement.

Description of the priv_type values

Permission	Description
ALL PRIVILEGES	All permissions except `GRANT OPTION`.
ALTER	The permission to execute the `ALTER TABLE` statement.
CREATE	The permission to execute the `CREATE TABLE` statement.
CREATE USER	The permission to execute the `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES` statements.
CREATE TABLEGROUP	The global permission to execute the `CREATE TABLEGROUP` statement.

Permission	Description
DELETE	The permission to execute the DELETE statement.
DROP	The permission to execute the DROP statement.
GRANT OPTION	The permission to execute the `GRANT OPTION` statement.
INSERT	The permission to execute the INSERT statement.
SELECT	The permission to execute the SELECT statement.
UPDATE	The permission to execute the UPDATE statement.
SUPER	The permission to execute the `SET GLOBAL` statement to modify global system parameters.
SHOW DATABASES	The global permission to execute the `SHOW DATABASES` statement to display all databases.
INDEX	The permission to execute the `CREATE INDEX` and `DROP INDEX` statements.
CREATE VIEW	The permission to create or delete a view.
SHOW VIEW	The permission to execute the `SHOW CREATE VIEW` statement.

Note You cannot grant users the permission to execute the `CHANGE EFFECTIVE TENANT` statement. However, all users of the system tenant have this permission.

Revoke permissions

You can execute the REVOKE statement as a system administrator to revoke the specified permissions from users.

Syntax

```
REVOKE priv_type
        ON database.tblname
       FROM 'user';
```

Notes:

- To revoke permissions from a user, you must have the permission that you are revoking. For example, if you use user1 to revoke the SELECT permission on table t1 from user2, user1 must have the SELECT permission on table t1. You must also have the `GRANT OPTION` permission.
- To revoke the `ALL PRIVILEGES` and `GRANT OPTION` permissions, you must have the global GRANT OPTION permission or the UPDATE and DELETE permissions on the permission list.
- Revocations do not have cascading effects. Assume that user1 grants permissions to user2. When the permissions are revoked from user1, the permissions are not revoked from user2.
- You can use an asterisk (*) instead of specifying a table name to revoke permissions on all tables in the specified database.
- If you revoke multiple permissions from a user at a time, separate the permission types with commas (,).
- If you revoke permissions from multiple users at a time, separate the usernames with commas (,).
- lists the priv_type values that can be specified for the REVOKE statement.

Examples

Revoke all permissions from `obsqluser01`.

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'obsqluser01';
```

View permissions

You can execute the `SHOW GRANTS` statement as a system administrator to view permissions that have been granted to a user.

Syntax

```
SHOW GRANTS [FOR username];
```

Notes:

- If you do not specify the username, this statement returns the permissions that have been granted to the current user. You can always view the permissions of the user that you are using.
- To view the permissions of a non-current user, you must have the SELECT permission on the *mysql.user* table.

Examples

```
Oceanbase>show grants for test;
+------------------------------+
| Grants for test              |
+------------------------------+
| GRANT USAGE ON *. * TO 'test' |
+------------------------------+
1 row in set (0.01 sec)
```

5.13.15. Modify system variables

System variables such as *autocommit* and *tx_isolation* affect SQL functionality and are stored in the *__all_sys_variable* table.

ApsaraDB for OceanBase maintains two types of variables:

- Global variables

  Global variables affect overall operations on the ApsaraDB for OceanBase system. When the ApsaraDB for OceanBase system starts, the system initializes all global variables to the default values. To modify global variables, you must have the SUPER permission.

- Session variables

  Session variables affect operations on the clients that are connected to the ApsaraDB for OceanBase system. When a client connects to the ApsaraDB for OceanBase system, the system initializes the session variables of the client to the current values of the corresponding global variables. To set session variables, you do not need special permissions. You can modify only the session variables of your client. You cannot modify the session variables of other clients.

  Note Modifications to global variables do not affect the session variables of clients that are connected to the ApsaraDB for OceanBase system. This is true even if the `SET GLOBAL` statement is executed on the connected clients.

Syntax

- Set a global variable.

  ```
  SET GLOBAL system_var_name = expr;
  ```

  Or

  ```
  SET @@GLOBAL.system_var_name = expr;
  ```

- Set a session variable.

  ```
  SET [SESSION | @@SESSION. | LOCAL | LOCAL. | @@]system_var_name =expr;
  ```

- Display the values of system variables.

  If the GLOBAL modifier is specified, the SHOW VARIABLES statement displays the values of global system variables. If the SESSION modifier is specified, this statement displays the values of system variables that are specific to the current session. If no modifier is specified, this statement displays the values of system variables that are specific to the current session.

  ```
  SHOW [GLOBAL | SESSION]
        VARIABLES
      [LIKE 'system_var_name' | WHERE expr];
  ```

  You can execute a `SELECT @@var_name` statement to search for a variable. If you do not specify the GLOBAL modifier, the system returns the value of a session-specific variable. If you specify the GLOBAL modifier, the system returns the value of a global variable.

Configure the READ ONLY property of a tenant

```
SET GLOBAL READ_ONLY={ON | OFF};
```

Or

```
SET @@GLOBAL.READ_ONLY={ON | OFF};
```

Examples

- Execute the following statement to change the value of the ob_trx_timeout session variable:

```
SET @@SESSION.ob_tx_timeout = 900000;
```

- Execute the following statement to query the value of the `ob_trx_timeout` variable:

```
Oceanbase>show variables like 'ob_trx_timeout';
+----------------+-----------+
| Variable_name  | Value     |
+----------------+-----------+
| ob_trx_timeout | 100000000 |
+----------------+-----------+
1 row in set (0.02 sec)
```

- Execute the following SELECT statement to query the value of the wait_timeout system variable:

```
SELECT @@wait_timeout;
+----------------+
| @@wait_timeout |
+----------------+
| 0              |
+----------------+
1 row in set (0.00 sec)
```

5.13.16. Modify user variables

A user variable in one statement stores a user-defined value that you can refer to later in another statement. This helps you pass values between statements. User variables are session-specific. A user variable that is defined by one client cannot be queried or used by another client. When a client exits, all variables that are specific to the client session are automatically released.

Syntax

```
SET @var_name = expr;
```

- User variables are expressed as @var_name. The variable name var_name consists of alphanumeric characters, decimal points (.), underscores (_), and dollar signs ($) from the current character set.
- The expression of a variable can be an integer, real number, string, or NULL value.
- If you set multiple user variables at a time, separate the variable definitions with commas (,).

Examples

- Execute the following SET statements to set user variables:

```
Oceanbase>SET @a=2, @b=3;
Query OK, 0 rows affected (0.01 sec)
Oceanbase>SET @c = @a + @b;
Query OK, 0 rows affected (0.00 sec)
```

- Execute the following statement to query user variables:

```
Oceanbase>SELECT @a, @b, @c;
+------+------+------+
| @a   | @b   | @c   |
+------+------+------+
|    2 |    3 |    5 |
+------+------+------+
1 row in set (0.01 sec)
```

- Execute the following SELECT statement to set user variables:

```
Oceanbase> SELECT @a:=2, @b:=3, @c:=@a+@b;
+-------+-------+-----------+
| @a:=2 | @b:=3 | @c:=@a+@b |
+-------+-------+-----------+
|     2 |     3 |         5 |
+-------+-------+-----------+
1 row in set (0.01 sec)
```

5.13.17. ALTER SYSTEM database management statements

5.13.17.1. Overview

You can execute ALTER SYSTEM statements to send commands to the ApsaraDB for OceanBase system to perform specified operations.

5.13.17.2. System-level management statements

Bootstrap the system

```
ALTER SYSTEM BOOTSTRAP
[REGION [=] 'region'] ZONE [=] 'zone' SERVER [=] 'ip:port'
[, [REGION [=] 'region'] ZONE [=] 'zone' SERVER [=] 'ip:port' ...] ;
```

To bootstrap the system, you must specify the root servers. Use commas (,) to separate multiple root servers.

REGION specifies a region. You must specify a region if your ApsaraDB for OceanBase service is deployed across zones in multiple regions.

A cluster can have only one root server to act as the leader to provide services.

Example:

```
ALTER SYSTEM BOOTSTRAP ZONE 'zone1' SERVER '10.218.248.178:55410';
```

Separate multiple root servers with commas (,).

```
ALTER SYSTEM BOOTSTRAP ZONE 'zone1' SERVER '172.24.65.24:55410', ZONE 'zone2' SERVER '172.24.65.114:55410';
```

Update the system version

```
ALTER SYSTEM BEGIN UPGRADE  # Start the system update.
ALTER SYSTEM END UPGRADE  # End the system update.
```

Notes:

1. When the administrator executes `ALTER SYSTEM BEGIN UPGRADE` to update the system version, the leader root server performs a series of checks. For example, it checks the servers for version consistency and modifies the `enable_upgrade_mode` parameter.
2. The leader root server maintains the minimum server version of the cluster. The leader root server analyzes server versions in the *__all_server* table and saves the minimum version in the min_observer_version parameter.

   Note ApsaraDB for OceanBase supports 3-bit version numbers, such as V1.2.3. In a version number, the first digit represents observer_major_version, the second digit represents observer_minor_version, and the third digit represents observer_patch_version.

3. When a system update begins, the leader root server checks whether all servers in the cluster are of the same version and whether the version is the same as the minimum version.

   When the ALTER SYSTEM END UPGRADE statement is executed to end the update, the leader root server checks whether all servers are updated and changes the min_observer_version value.

Manage the schema

To refresh the schema, execute the following statement:

```
ALTER SYSTEM REFRESH SCHEMA {SERVER='ip:port'| ZONE='zone'};
```

When the system executes Data Definition Language (DDL) operations, the leader root server notifies all OBServers to refresh the schema.

If an exception occurs to an OBServer, the OBServer is disconnected from the leader root server. You must manually refresh the schema. You can refresh the schema for an OBServer or a cluster.

Examples:

- Refresh the schema for an OBServer.

  ```
  ALTER SYSTEM REFRESH SCHEMA SERVER='172.24.65.24:55410';
  ```

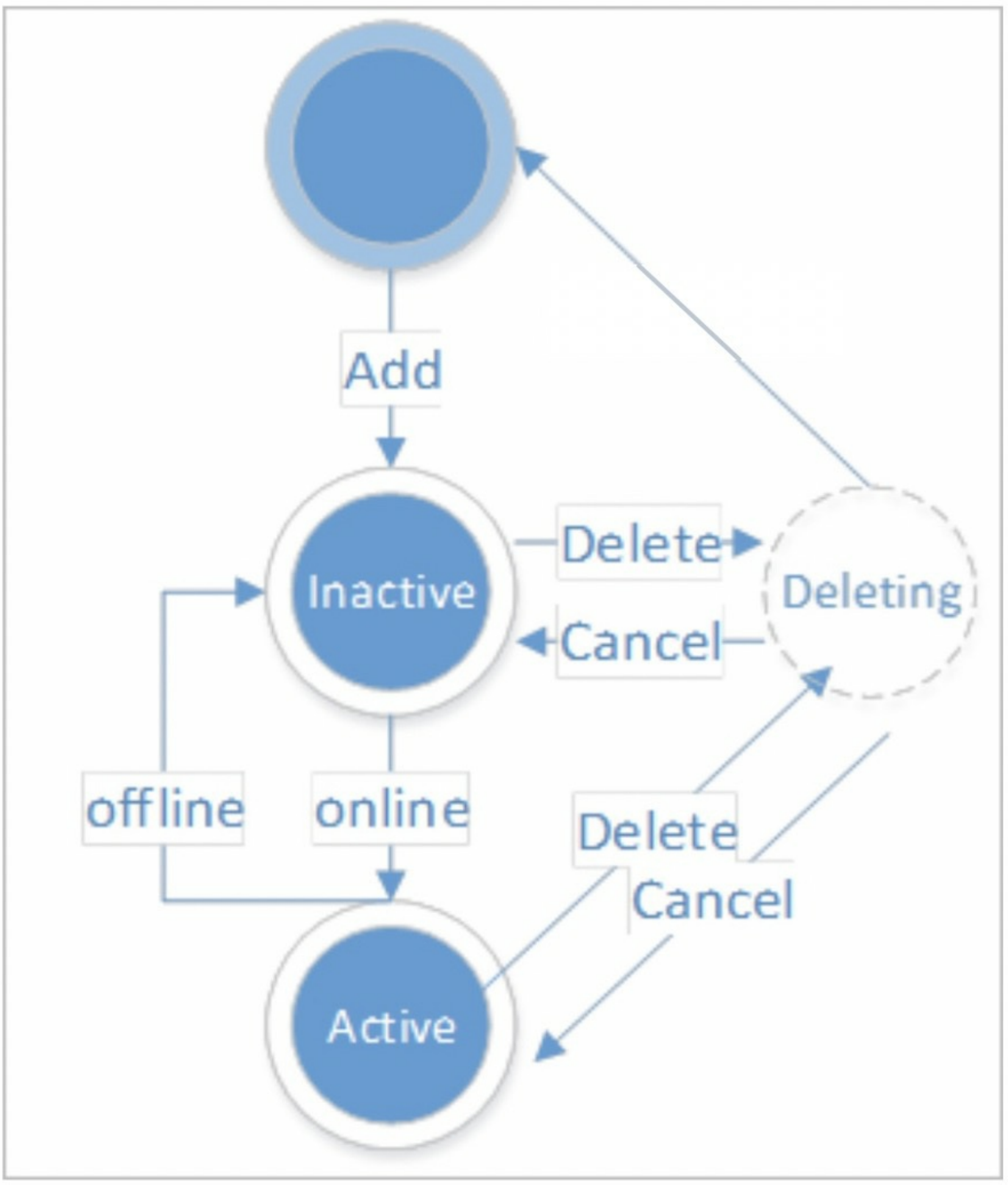
- Refresh the schema for all OBServers that are deployed in a zone.

```
ALTER SYSTEM REFRESH SCHEMA ZONE='zone1';
```

Manage servers

Server status shows the status of a server when different operations are performed.

Server status



Notes:

1. You can execute the ALTER SYSTEM ADD SERVER or ALTER SYSTEM DELETE SERVER statement to add a server to or delete a server from the server list. Only servers on the list can provide services.
2. When the ALTER SYSTEM DELETE SERVER statement is executed, the ApsaraDB for OceanBase system selects a new replica leader and replicates replicas.
3. The DELETE operation is time-consuming. You can execute the ALTER SYSTEM CANCEL DELETE SERVER statement to abort this operation.

To add a server, execute the following statement:

```
ALTER SYSTEM ADD SERVER 'ip:port' [,'ip:port'…] [ZONE[=]'zone'];
```

If a zone is specified, the system checks whether the server that you want to add is deployed in the specified zone.

Example:

```
ALTER SYSTEM ADD SERVER '172.24.65.113:55410' ZONE 'zone1';
```

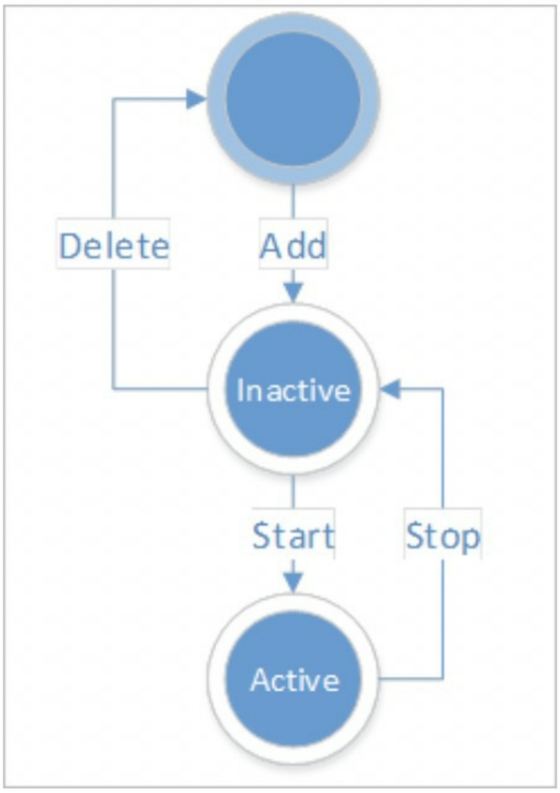
To delete and undelete a server, execute the following statements:

```
ALTER SYSTEM DELETE SERVER 'ip:port' [,'ip:port'…] [ZONE[=]'zone']
ALTER SYSTEM CANCEL DELETE SERVER 'ip:port' [,'ip:port'…] [ZONE[=]'zone'];
```

Manage zones

Zone status shows the status of a zone when different operations are performed.

Zone status



- Add a zone.

  ```
  ALTER SYSTEM ADD ZONE 'zone' [REGION 'region'];
  ```

- Delete a zone.

  ```
  ALTER SYSTEM DELETE ZONE 'zone';
  ```

- Activate or deactivate a zone.

  ```
  ALTER SYSTEM {START|STOP} ZONE 'zone';
  ```

Modify the region property for a zone

You can specify the region property for a zone in ApsaraDB for OceanBase V1.3 and later versions.

Syntax:

```
ALTER SYSTEM {ALTER|MODIFY|CHANGE} ZONE 'zone_name' [set] alter_zone_option
alter_zone_option: region [=] 'region_name'
```

Manage sessions

If you are a user of a general tenant, execute the following statement to close a session:

```
KILL [GLOBAL|LOCAL] [CONNECTION] 'sessionid';
```

If you are a system administrator, execute the following statement to close a session:

```
ALTER SYSTEM KILL [GLOBAL|LOCAL] SESSION 'sessionid';
```

Manage partitions

- Select a new replica leader.

  ```
  ALTER SYSTEM
  SWITCH REPLICA LEADER | FOLLOWER
  {PARTITION_ID='partidx%partcount@tableid' SERVER='ip:port' | SERVER='ip:port' | ZONE='zone'};
  ```

- Delete a replica.

  ```
  ALTER SYSTEM
         DROP REPLICA
         PARTITION_ID = 'partidx%partcount@tableid'
         SERVER = 'ip:port'
       [CREATE_TIMESTAMP = ctimestamp]
         [ ZONE='zone'];
  ```

  To delete a replica on the specified OBServer, you must specify the following parameters: PARTITION_ID, SERVER, and CREATE_TIMESTAMP.

- Migrate or replicate a replica from one OBServer to another OBServer.

  ```
  ALTER SYSTEM
         {MOVE|COPY} REPLICA
         PARTITION_ID 'part_idx%part_count@table_id'
         SOURCE='ip:port'
         DESTINATION='ip:port' ;
  ```

  To migrate or replicate a replica, you must specify the source OBServer, the destination OBServer, and the partition ID.

- Cancel a replica migration or replication task.

  ```
  ALTER SYSTEM
      CANCEL [PARTITION MIGRATION] TASK 'task_id';
  ```

  To cancel a migration task, you must specify the task ID. You can query task IDs from the __all_virtual_sys_task_status table.

- Report replicas.

  ```
  ALTER SYSTEM
         REPORT REPLICA
         {SERVER = 'ip:port' | ZONE='zone'};
  ```

  This statement requires an OBServer or all OBServers in a zone to report replicas.

- Recycle replicas that are not used.

  ```
  ALTER SYSTEM RECYCLE REPLICA {SERVER = 'ip:port' | ZONE='zone'};
  ```

- Modify the properties of a replica.

  ```
  ALTER SYSTEM
    {alter|change|modify} REPLICA
    PARTITION_ID 'part_idx%part_count@table_id'
    SERVER = 'ip:port'
    [set] change_actions;
  change_actions: REPLICA_TYPE = 'replica_type'
  ```

  This statement modifies the type of a specified replica.

  The valid values of replica type are FULL, READONLY, and LOGONLY. You can set the value of `replica_type` to the full name or the abbreviation of a valid replica type, such as F, R, or L. The value is not case-sensitive.

Run a system job

To run a system job, execute the following statement:

```
ALTER SYSTEM RUN JOB 'job_name' {SERVER='ip:port'| ZONE='zone'}
```

If the specified job name contains a special character, enclose the job name in single quotation marks ('). In other cases, single quotation marks are optional.

The system supports the following jobs:

- check_partition_table

  You can run this job to check partitioned tables on an OBServer.

- root_inspection

  You can run this job to trigger the leader root server to check the following information:

  - Whether the hard-coded schema in the system table is the same as the schema in the internal table.
  - Whether the hard-coded system variables are the same as the system variables in the internal table.
  - Whether the hard-coded zone_info and sys_stat are the same as those in the internal table.

Perform daily major freeze operations

You can perform the following daily major freeze operations:

- Initiate a daily major freeze request.

  ```
  ALTER SYSTEM MAJOR FREEZE
  ```

- Activate a manually triggered major freeze operation.

  ```
  ALTER SYSTEM SET ENABLE_MANUAL_MERGE='True'
  ```

- Deactivate a manually triggered major freeze operation.

  ```
  ALTER SYSTEM SET ENABLE_MANUAL_MERGE='False'
  ```

- Start the daily major freeze operation.

  ```
  ALTER SYSTEM START MERGE ZONE='zone';
  ```

- Suspend the daily major freeze operation.

  ```
  ALTER SYSTEM SUSPEND MERGE [ZONE='zone']
  ```

- Resume the daily major freeze operation.

  ```
  ALTER SYSTEM RESUME MERGE [ZONE='zone']
  ```

  ```
  ALTER SYSTEM CLEAR ROOTTABLE [TENANT='tenantname'];
  ```

Manage memory

You can perform the following memory management operations:

- Initiate a minor freeze request.

  ```
  ALTER SYSTEM MINOR FREEZE
      [{TENANT [=] ('tt1' [, 'tt2'...]) | PARTITION_ID [=] 'partidx%partcount@tableid'}]
  [SERVER [=] ('ip:port' [, 'ip:port'...])] ;
  ```

- Clear a plan cache.

  ```
  ALTER SYSTEM FLUSH PLAN CACHE
  ```

- Clear KV caches.

  ```
  1. Clear the schema cache of the system tenant.
  ALTER SYSTEM FLUSH KVCACHE TENANT='sys' CACHE='schema_cache';
  2. Clear the KV caches of the system tenant.
  ALTER SYSTEM FLUSH KVCACHE TENANT='sys';
  3. Clear the KV caches of all tenants.
  ALTER SYSTEM FLUSH KVCACHE;
  ```

Configure parameters

- Modify system parameters.

```
ALTER SYSTEM [SET] param_name [=] expr
        [COMMENT [=]'text']
        [SCOPE [=] conf_scope]
      {SERVER [=] 'ip:port' | ZONE [=] 'zone'};
```

  For more information about how to modify system parameters, see .

Clause description

Clause	Description
param_name = *expr*	The name of a system parameter.
COMMENT '*text*'	Optional. This clause is used to add a comment about the modification. We recommend that you add a comment.
SCOPE = *conf_scope*	The effective range of the modification. The following three values are supported: ◦ MEMORY: Only the parameter value in the memory is modified. The modified parameter takes effect immediately after the modification and becomes invalid after the server is restarted. However, no parameter supports this option. ◦ SPFILE: Only the parameter value in the configuration table is modified. The modified parameter takes effect after the server is restarted. ◦ BOTH: The parameter values in the configuration table and in the memory are modified. The modified parameter takes effect immediately after the modification and remains valid after the server is restarted. **Note** The default value is BOTH. If you specify BOTH or MEMORY for a parameter that cannot take effect immediately after the modification, the system returns an error.
SERVER_TYPE = *server_type*	The server type. Valid values: ROOTSERVER, UPDATESERVER, CHUNKSERVER, and MERGESERVER.
ZONE = '*zone_name*'	The name of the zone, which indicates that the modified parameter applies to the specified type of servers in the specified cluster. If you do not specify the zone name, the modified parameter applies to the specified type of servers in all clusters.
SERVER = '*ip:port*'	The IP address and port number of the server, which indicates that only the parameter of a specified server is modified.

  If you modify multiple system parameters in one request, separate the parameters with commas (,).

- Query system parameters.

```
SHOW PARAMETERS [LIKE 'pattern' | WHERE expr];
```

Manage time zone information

```
ALTER SYSTEM REFRESH TIME_ZONE_INFO
```

ApsaraDB for OceanBase uses mysql_tzinfo_to_sql to generate an SQL script based on the time zone information in the operating system. The system then adds the time zone information into the following four system tables: __all_time_zone, __all_time_zone_name, __all_time_zone_transition, and __all_time_zone_transition_type.

The `ALTER SYSTEM REFRESH TIME_ZONE_INFO` statement notifies all servers in the cluster to update the local time zone information based on the related system tables.

Reset the valid flag for a disk

```
ALTER SYSTEM SET DISK VALID SERVER [=] 'ip:port'
```

ApsaraDB for OceanBase automatically checks whether a disk has failed. If a possible disk failure is detected, the system sets a flag to migrate the leader from the current OBServer to another OBServer.

This flag is valid for all partitions on the OBServer.

When the database administrator finds that the disk restores to a normal status, the administrator can execute this statement to reset the flag.

5.13.17.3. Tenant-level management statements

Modify tenant-level parameters

```
ALTER SYSTEM SET param_name = expr
        [COMMENT 'text']
        [SCOPE = conf_scope]
        [TENANT = 'tenantname']
```

Enable or disable the migration feature or the replication feature

- Enable or disable the migration feature.

  ```
  ALTER SYSTEM SET ENABLE_REBALANCE = {'true'|'false'} [TENANT='tenantname'];
  ```

- Enable or disable the replication feature.

  ```
  ALTER SYSTEM SET ENABLE_REREPLICATION= {'true'|'false'} [TENANT='tenantname'];
  ```

5.14. READ ONLY

5.14.1. Overview

ApsaraDB for OceanBase allows you to set the READ ONLY property at the tenant, database, or table level.

General syntax

The following rules apply:

1. To define the READ ONLY property, you can execute the following ALTER statement, which is consistent with Oracle in syntax.

   ```
   ALTER {TENANT | DATABASE | TABLE }
   {tenant_name | database_name | table_name}
   {READ ONLY | READ WRITE}
   ```

2. To be compatible with MySQL instances, you can execute the `SET @@GLOBAL.READ_ONLY=ON|OFF` statement to set the READ ONLY property for a tenant.
3. After you configure the READ ONLY property at a specific level, the specified objects can be in one of the following states: READ ONLY, READ WRITE, and PARTIALLY READ ONLY. PARTIALLY READ ONLY is an intermediate status, which indicates that the specified objects are in the READ ONLY state on some OBServers.

Notes

When you set the READ ONLY property, follow these rules:

1. The READ ONLY property can apply to the following levels in descending order: tenant level, database level, or table level. The READ ONLY property at a higher level takes precedence over that at a lower level.

   For example, if a tenant is in the READ ONLY state, the databases and tables within the tenant are in the READ ONLY state regardless of whether the READ ONLY property is set.

2. To be compatible with MySQL, you must have the SUPER permission to perform read and write operations on the objects that are in the READ ONLY state.

5.14.2. Tenant-level READ ONLY property

Configure the READ ONLY property

Use one of the following two methods based on your needs:

- In a system tenant, log on to the system by using a user that has the SUPER permission, and execute the following statement to set the READ ONLY property for all tenants:

  ```
  ALTER TENANT tenant_name {READ ONLY | READ WRITE};
  ```

- In a general tenant, log on to the system by using a user that has the SUPER permission, and execute one of the following statements to set the READ ONLY property for the current tenant:

```
SET GLOBAL READ_ONLY={ON | OFF};
Or
SET @@GLOBAL.READ_ONLY={ON | OFF};
```

Query the READ ONLY property

Use one of the following three methods to query the READ ONLY property of the current tenant:

- Execute the `SHOW TENANT STATUS` statement.

```
SHOW TENANT STATUS;
```

  Example:

```
OceanBase (admin@test)> show tenant status;
+--------+-----------+
| Tenant | State     |
+--------+-----------+
| sys    | read only |
+--------+-----------+
1 row in set (0.01 sec)
```

- Query the internal table *oceanbase.__tenant_virtual_tenant_status*.

```
OceanBase (admin@test)> select * from oceanbase.__tenant_virtual_tenant_status;
+--------+-------------+-------+-----------+
| tenant | host        | port  | read_only |
+--------+-------------+-------+-----------+
| sys    | 172.24.65.24 | 55410 |         1 |
+--------+-------------+-------+-----------+
1 row in set (0.00 sec)
```

  You can query the status of the current tenant on all OBServers based on the *oceanbase.__tenant_virtual_tenant_status* table.

- Query the read_only system variable.

```
show global variables like 'read_only'
```

  Or

```
select @@global.read_only
```

5.14.3. Database-level READ ONLY property

Configure the READ ONLY property

```
ALTER DATABASE database_name {READ ONLY | READ WRITE};
```

Query the READ ONLY property

Use one of the following two methods to query the READ ONLY property of the databases under the current tenant:

- Execute the `SHOW DATABASES STATUS` statement to query the READ ONLY property of the specified databases.

  The system returns the status information for each database. In the output, one column shows whether each database is in the READ ONLY state. Each database supports three states: READ ONLY, READ WRITE, and PARTIALLY READ ONLY. READ ONLY indicates that this database is only readable. READ WRITE indicates that this database is readable and writable. PARTIALLY READ ONLY indicates an intermediate status where the database is in the READ WRITE state on some OBServers and in the READ ONLY state on the other OBServers.

  You can include a LIKE or WHERE clause in this statement. The clause is used to specify the databases for which you want to query the status.

- Query the internal table *oceanbase.__tenant_virtual_database_status*.

```
OceanBase (admin@test)> select * from oceanbase.__tenant_virtual_database_status;
+-------+-------------+-------+-----------+
| db    | host        | port  | read_only |
+-------+-------------+-------+-----------+
| mysql | 172.24.65.24 | 55410 |         0 |
| test  | 172.24.65.24 | 55410 |         0 |
+-------+-------------+-------+-----------+
2 rows in set (0.00 sec)
```

You can query the status of all databases under the current tenant on all OBServers based on the *oceanbase.__tenant_virtual_database_status* table.

5.14.4. Table-level READ ONLY property

Configure the READ ONLY property

```
ALTER TABLE table_name [SET] {READ ONLY | READ WRITE}
```

Query the READ ONLY property

Use one of the following two methods to query the READ ONLY property of tables:

- Execute the `SHOW TABLE STATUS` statement.

  ```
  SHOW TABLE STATUS
  ```

- Query the internal table oceanbase.__tenant_virtual_table_status.

  You can query the status of all tables in all databases under the current tenant on all OBServers based on the *oceanbase.__tenant_virtual_table_status* table.

5.15. Other SQL statements

5.15.1. SHOW statements

You can execute SHOW statements to query information such as the status of databases, tables, columns, or servers in the ApsaraDB for OceanBase system.

SHOW CHARACTER SET

```
SHOW CHARACTER SET
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW CHARACTER SET` statement returns information about all available character sets.

This statement can contain a LIKE or WHERE clause.

The clause is used to specify the names of character sets or columns that you want to match.

```
mysql> SHOW CHARACTER SET WHERE charset='utf8mb4';
+---------+---------------+--------------------+--------+
| Charset | Description   | Default collation  | Maxlen |
+---------+---------------+--------------------+--------+
| utf8mb4 | UTF-8 Unicode | utf8mb4_general_ci |      4 |
+---------+---------------+--------------------+--------+
1 row in set (0.00 sec)
mysql> SHOW CHARACTER SET like 'utf8%';
+---------+---------------+--------------------+--------+
| Charset | Description   | Default collation  | Maxlen |
+---------+---------------+--------------------+--------+
| utf8    | UTF-8 Unicode | utf8_general_ci    |      3 |
| utf8mb4 | UTF-8 Unicode | utf8mb4_general_ci |      4 |
+---------+---------------+--------------------+--------+
2 rows in set (0.00 sec)
```

SHOW COLLATION

```
SHOW COLLATION
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW COLLATION` statement returns all available collations.

This statement can contain a LIKE or WHERE clause.

The clause is used to specify the names of character sets or columns that you want to match.

```
mysql> SHOW COLLATION LIKE 'utf8mb4_bin%';
+-------------+---------+----+---------+----------+---------+
| Collation   | Charset | Id | Default | Compiled | Sortlen |
+-------------+---------+----+---------+----------+---------+
| utf8mb4_bin | utf8mb4 | 46 |         | Yes      |       1 |
+-------------+---------+----+---------+----------+---------+
1 row in set (0.01 sec)
mysql> SHOW COLLATION WHERE charset='utf8mb4';
+------------------------+---------+-----+---------+----------+---------+
| Collation              | Charset | Id  | Default | Compiled | Sortlen |
+------------------------+---------+-----+---------+----------+---------+
| utf8mb4_general_ci     | utf8mb4 |  45 | Yes     | Yes      |       1 |
| utf8mb4_bin            | utf8mb4 |  46 |         | Yes      |       1 |
+------------------------+---------+-----+---------+----------+---------+
```

SHOW COLUMNS

```
SHOW [FULL] COLUMNS {FROM | IN} tblname [{FROM | IN} dbname]
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW COLUMNS` statement returns information about the columns of a specified table.

You can also execute this statement to return column information for a specified view.

`SHOW FIELDS` is equivalent to `SHOW COLUMNS`.

- If the FULL keyword is specified, the output contains the permissions that you have and the comments for each column.
- The `dbname.tblname` syntax is an alternative to `tblname FROM dbname`. These two statements are equivalent.

  Example:

  ```
  mysql> SHOW COLUMNS FROM mytable FROM mydb;
  mysql> SHOW COLUMNS FROM mydb.mytable;
  ```

```
mysql> SHOW COLUMNS IN users IN test;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| name     | varchar(30) | YES  |     | NULL    |                |
| class    | varchar(30) | YES  |     | NULL    |                |
| hometown | varchar(30) | YES  |     | NULL    |                |
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
+----------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
mysql> SHOW FULL COLUMNS IN users IN test;
+----------+-------------+-----------------+------+-----+---------+----------------+---------------------------------+---
------+
| Field    | Type        | Collation       | Null | Key | Default | Extra          | Privileges                      | Co
mment |
+----------+-------------+-----------------+------+-----+---------+----------------+------------------------
| name     | varchar(30) | utf8_general_ci | YES  |     | NULL    |                | select,insert,update,references |
|
| class    | varchar(30) | utf8_general_ci | YES  |     | NULL    |                | select,insert,update,references |
|
| hometown | varchar(30) | utf8_general_ci | YES  |     | NULL    |                | select,insert,update,references |
|
| id       | int(11)     | NULL            | NO   | PRI | NULL    | auto_increment | select,insert,update,references |
|
+----------+-------------+-----------------+------+-----+---------+----------------+------------------------
4 rows in set (0.03 sec)
```

SHOW CREATE DATABASE

```
SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] dbname
```

The SHOW CREATE DATABASE statement returns the `CREATE DATABASE` statement that creates a specified database. You can also execute the `SHOW CREATE SCHEMA` statement to accomplish the same effect.

SHOW DATABASES STATUS

```
SHOW DATABASES STATUS [LIKE 'pattern' | WHERE expr]
```

The SHOW DATABASES STATUS statement returns the status of databases. In the output, one column shows whether a database is read only. A database supports the following states: READ ONLY, READ WRITE, and PARTIALLY READ ONLY. READ ONLY indicates that the database is only readable. READ WRITE indicates that the database is readable and writable. PARTIALLY READ ONLY indicates an intermediate status where the database is in the READ WRITE state on some OBServers and in the READ ONLY state on the other OBServers.

This statement can contain a LIKE or WHERE clause. The clause is used to specify the databases that you want to match.

SHOW CREATE TABLE

```
SHOW CREATE TABLE tblname
```

The SHOW CREATE TABLE statement returns the `CREATE TABLE` statement that creates a specified table.

You can also execute this statement to return the CREATE TABLE statement that creates a specified view.

SHOW CREATE VIEW

```
SHOW CREATE VIEW viewname
```

The SHOW CREATE VIEW statement returns the `CREATE VIEW` statement that creates a specified view.

SHOW DATABASES

```
SHOW {DATABASES | SCHEMAS}
          [LIKE 'pattern' | WHERE expr]
```

The `SHOW DATABASES` statement returns information about ApsaraDB for OceanBase databases.

You can query information about only databases on which you have permissions unless you have the global `SHOW DATABASES` permission.

`SHOW SCHEMAS` is equivalent to `SHOW DATABASES`.

This statement can contain a LIKE or WHERE clause.

The clause is used to specify the databases that you want to match.

SHOW ERRORS

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

The SHOW ERRORS statement is similar to the `SHOW WARNINGS` statement in the effect. The difference is that the SHOW ERRORS statement returns only errors, whereas the SHOW WARNINGS statement also returns warnings and notes.

The `SHOW COUNT(*) ERRORS` statement returns the number of errors.

SHOW WARNINGS

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

`SHOW WARNINGS [LIMIT [offset,] The SHOW WARNINGS` statement returns the errors, warnings, and notes that were generated during the execution of the previous statement. The LIMIT clause specifies the number of rows that can be displayed.

The `SHOW COUNT(*) WARNINGS` statement returns the total number of errors, warnings, and notes.

SHOW GRANTS

```
SHOW GRANTS [FOR user]
```

The `SHOW GRANTS` statement returns information about the permissions that have been granted to an ApsaraDB for OceanBase user.

If you do not specify the username, this statement returns the permissions that have been granted to the current user.

SHOW INDEX

```
SHOW {INDEX | INDEXES | KEYS}
       {FROM | IN} tblname
     [{FROM | IN} dbname]
     [WHERE expr]
```

The SHOW INDEX statement returns information about the indexes of a table.

The `dbname.tblname` syntax is an alternative to `tblname FROM dbname` .

The following two statements are equivalent:

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

SHOW PRIVILEGES

```
SHOW PRIVILEGES
```

The `SHOW PRIVILEGES` statement returns the system permissions that are supported by ApsaraDB for OceanBase.

SHOW PROCESSLIST

```
SHOW [FULL] PROCESSLIST
```

The `SHOW PROCESSLIST` statement returns the information about running threads.

If you have the SUPER permission, you can view all threads. If you do not have the SUPER permission, you can view only your own threads.

If the FULL keyword is not specified, this statement returns only the first 100 characters of each query.

SHOW STATUS

```
SHOW [GLOBAL | SESSION] STATUS
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW STATUS` statement returns information about server status.

If you specify the GLOBAL modifier, this statement returns the status values for all connections to the ApsaraDB for OceanBase system. If you specify the SESSION modifier, this statement returns status values for the current connection. If neither of the two modifiers is specified, the default is SESSION.

LOCAL is equivalent to SESSION.

Note

Some status variables have only global values.

For these variables, this statement returns the same result regardless of whether you use the GLOBAL or SESSION modifier.

SHOW TABLE STATUS

```
SHOW TABLE STATUS [{FROM | IN} dbname]
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW TABLE STATUS` statement is similar to the `SHOW TABLES` statement. However, the SHOW TABLE STATUS statement returns detailed information about each non-temporary table.

You can also execute this statement to return information about views.

SHOW TABLES

```
SHOW [FULL] TABLES [{FROM | IN} dbname]
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW TABLES` statement lists the non-temporary tables in a specified database.

This statement also lists the views in the database.

The FULL keyword is optional. If this keyword is specified, the `SHOW FULL TABLES` statement returns the second output column.

For a table, the value of the second column is BASE TABLE. For a view, the value of the second column is VIEW. If you do not have permissions on a table, this table is not displayed in the output from the `SHOW TABLES` statement.

SHOW VARIABLES

```
SHOW [GLOBAL | SESSION] VARIABLES
     [LIKE 'pattern' | WHERE expr]
```

The `SHOW VARIABLES` statement returns the values of system variables that are supported by ApsaraDB for OceanBase.

If you specify the GLOBAL modifier, this statement returns the values that are used for new connections to the ApsaraDB for OceanBase system. If you specify the SESSION modifier, this statement returns the values that are in effect for the current connection. If neither of the two modifiers is specified, the default is SESSION.

LOCAL is equivalent to SESSION.

If you specify a LIKE or WHERE clause, this statement returns only variables that match the specified condition pattern.

SHOW PARAMETERS

```
 SHOW PARAMETERS
     [LIKE 'pattern' | WHERE expr]
```

The SHOW PARAMETERS statement returns the value of each parameter.

SHOW CREATE TENANT

```
SHOW CREATE TENANT tenantname
```

The SHOW CREATE TENANT statement returns the information about a specified tenant.

```
SHOW CREATE TENANT test;
+-------+---------------------------------------------------------------------------------------------------+
| Tenant | Create Tenant   |
+-------+---------------------------------------------------------------------------------------------------+
| test   | CREATE TENANT test
charset='utf8mb4', replica_num=1, zone_list=('zone1'), primary_zone='zone1', resource_pool_list=('p1'); |
+-------+---------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SHOW TENANT

```
SHOW TENANT
```

The SHOW TENANT statement returns the information about the current tenant.

```
SHOW TENANT
+-----------------------------------+
| CURRENT TENANT NAME |
+-----------------------------------+
| test|
+-----------------------------------+
1 row in set (0.00 sec)
```

SHOW TABLEGROUPS statement

```
SHOW TABLEGROUPS
          [LIKE 'pattern' | WHERE expr]
```

The `SHOW TABLEGROUPS` statement returns the information about all table groups that are created in the current tenant.

This statement can contain a LIKE or WHERE clause.

The clause is used to specify the table groups you want to match. Then, you can query the tables that are contained in each specified table group.

5.15.2. KILL statement

```
KILL [GLOBAL | LOCAL] [CONNECTION | QUERY] 'sessionid'
```

Each connection to the ApsaraDB for OceanBase system runs in an independent thread. To query the running threads, execute the `SHOW PROCESSLIST;` statement. To kill a thread, execute the `KILL sessionid` statement.

When you execute the KILL statement, follow these rules:

- The `KILL CONNECTION` statement closes the connection that is associated with the specified thread. This is the same as a KILL statement that does not contain a modifier.
- The `KILL QUERY` statement terminates the statement that is being executed over the connection. The connection state remains unchanged.

If you have the PROCESS permission, you can view all threads.

If you have the SUPER permission, you can kill all threads and statements. If you do not have the PROCESS or SUPER permission, you can view or kill only your own threads and statements.

5.15.3. DESCRIBE statement

```
{DESCRIBE | DESC | EXPLAIN} tblname [colname | wild];
```

This statement is equivalent to the `SHOW COLUMNS FROM` statement.

Example:

```
mysql> DESCRIBE city;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

Notes:

- The Key field indicates whether the column is indexed.

  Valid values:

  - PRI: indicates that the column is a primary key or is one of the columns in a multiple-column primary key of the table.
  - UNI: indicates that the column is part of a unique index.
  - MUL: indicates that a given value can appear in the column multiple times.

    If a unique index is a composite unique index that consists of multiple columns, the value MUL is displayed in the unique index. Although the combination of columns is unique, each column may contain a given value that appears multiple times. Note that in a composite index, only the leftmost column of the index can be included in the Key field.

- The Default field indicates whether a default value is assigned to the column.
- The Extra field lists additional information about the given column.

  In this example, the Extra field indicates that the AUTO_INCREMENT keyword is used when the Id column is created.

5.15.4. USE statement

Syntax

```
USE dbname
```

`USE The USE dbname` statement informs the client to use a specified database as the default database for subsequent statements. This database acts as the default database until the session ends or another USE statement is issued.

Examples

```
mysql> USE db1;
mysql> SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
mysql> USE db2;
mysql> SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

If you do not execute the USE statement to specify the current database, the following error is returned:

```
root@(none) 04:59:12>create table t1 (a int);
ERROR 1046 (3D000): No database selected
root@(none) 04:59:24>select * from t1;
ERROR 1046 (3D000): No database selected
```

If you execute the USE statement to specify a database as the current database, you can still access tables in other databases.

The following example shows how to access the author table from the db1 database and the editor table from the db2 database:

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
->        WHERE author.editor_id = db2.editor.editor_id;
```

Errors

If the database name that follows the USE keyword is not found, the system returns the following error: `ERROR 1049 (42000): Unknown database 'test1'` .

5.15.5. Hints

Syntax

Hints are a special type of comments for SQL statements. The general syntax for comments is `/* ... */`. This is the same as the syntax for C language comments. Based on the general comment syntax, the hint syntax adds a plus sign (+) that follows the /* comment opening sequence and is displayed as `/*+ ... */` .

If the server does not recognize the hint in your SQL statement, it ignores the hint and does not return an error. Therefore, when an SQL statement is distributed to different databases, the hints in this statement do not cause syntax errors.

Hints only affect the internal optimization logic of the database server. They do not affect the semantics of SQL statements.

ApsaraDB for OceanBase is compatible with Oracle in hint syntax, except for some unique hints that are specific to ApsaraDB for OceanBase databases.

Hints that are supported by ApsaraDB for OceanBase have the following characteristics:

- Some hints do not contain parameters, for example, / `*+ KAKA */` .
- Some hints contain parameters, for example, `/*+ HAHA(param) */` .
- Multiple hints can be included in one hint comment and are separated with commas (,). For example, `/*+ KAKA, HAHA(param) */` .
- A hint in the SELECT statement must follow the SELECT keyword and precede other keywords. For example, `SELECT /*+ KAKA */ ...` .
- A hint in the UPDATE statement or the DELETE statement must follow the UPDATE or DELETE keyword.

Specify a valid subquery

In data manipulation language (DML) statements, QB_NAME indicates the name of a query block, which uniquely identifies the query block. The query block name can be system-generated or user-specified.

If you do not specify QB_NAME in the hint, the system assigns SEL$1, SEL$2, UPD$1, and DEL$1 to query blocks in sequence from left to right. The system also uses the same sequence to resolve these query block names.

You can find each table based on QB_NAME, or specify the actions on a specific query block.

You can specify QB_NAME in TBL_NAME to find the table. The first QB_NAME in the hint allows you to find the query block to which the hint applies.

For example, you can specify the primary key and the index for outer queries and subqueries in the following scenario:

```
OceanBase (root@test)> create table t1(c1 int, c2 int, key t_c1(c1));
select /*+INDEX(t@SEL$1 PRIMARY) INDEX(@SEL$2 t@SEL$2 t_c1)*/ *
from t ,
     (select * from t where c2 = 1) ta
where t.c1 = 1;
Query Plan: ============================================================
|ID|OPERATOR                        |NAME   |EST. ROWS|COST |
------------------------------------------------------------
|0 |NESTED-LOOP INNER JOIN CARTESIAN|       |1        |16166|
|1 | TABLE SCAN                     |t      |1        |1397 |
|2 | TABLE SCAN                     |t(t_c1)|1        |14743|
============================================================
Outputs & filters:
-------------------------------------
  0 - output([t.c1], [t.c2], [t.c1], [t.c2]), filter(nil),
      conds(nil), nl_params_(nil)
  1 - output([t.c1], [t.c2]), filter([t.c1 = 1]),
      access([t.c1], [t.c2]), partitions(p0)
  2 - output([t.c2], [t.c1]), filter([t.c2 = 1]),
      access([t.c2], [t.c1]), partitions(p0)
```

The following table lists the hints that are supported by ApsaraDB for OceanBase.

Hints

hint	Parameter	Applicable statement	Description
READ_CONSISTENCY	WEAK, STRONG, or FROZEN	SELECT	Indicates whether to perform strongly consistent reads or weakly consistent reads. If the value is not specified in the SQL statement, the default value depends on the value of the system variable ob_read_consistency. FROZEN indicates that the system reads the data from the latest frozen point. The frozen version is automatically selected by the system.
FROZEN_VERSION	Frozen version	SELECT	Reads only the data of the specified version.
READ_ZONE	FOLLOWER or LEADER	SELECT	Specifies the cluster to which the request is sent. `READ_ZONE(FOLLOWER)` indicates that the system preferentially sends the request to a secondary cluster regardless of whether a secondary cluster exists. This hint only controls the routing policy of an ApsaraDB for OceanBase client on SQL statements. It is not associated with READ_CONSISTENCY.

hint	Parameter	Applicable statement	Description
QUERY_TIMEOUT	Time-out period in microseconds	All DML statements	Defines the timeout period for SQL statements when they are executed on the server. After a statement times out, the server terminates the execution and returns a timeout error code.
USE_NL	USE_NL, which specifies the name of the right physical table	SELECT	Performs a NESTED LOOP JOIN to join two tables. For more information, see the usage of Oracle statements.
USE_MERGE	Table name	SELECT	In a MERGE JOIN, the system first sorts each table you want to join based on the column to be joined. Then, the system extracts data from the associated sorted tables to another sorted table to match data. MERGE JOINs require multiple sorts and are resource-consuming.
LEADING	Table name	SELECT	In a multi-table join query, this hint specifies the driving table and the table to which the optimizer firstly accesses.
ORDERED	_	SELECT	The system selects the driving table based on the sequence of the tables that follow the FROM keyword.
USE_PLAN_CACHE	DEFAULT/NONE	All DML statements	Defines an execution plan to be used.
INDEX	INDEX, which is in the <TableName IndexName> format	All DML statements	Similar to the INDEX hint in Oracle, this hint specifies that the data from the specified table must be queried based on the index name. If the index does not exist or is unavailable, the system does not return an error. Example: `SELECT /*+ INDEX(t1 i1) , INDEX(t2 i2)*/ from t1, t2 WHERE t1.c1=t2.c1;` `DELETE /*+ INDEX(t1 i1) */ from t1 WHERE t1.c1=1;`

hint	Parameter	Applicable statement	Description
PARALLEL	PARALLEL(N)	SELECT	Specifies the maximum number of SQL statements that can be concurrently executed. Only SELECT statements that meet one of the following conditions can be concurrently executed: • SELECT statements that contain an aggregate function • SELECT statements that contain a GROUP BY clause • SELECT statements that contain an ORDER BY clause • SELECT statements that do not contain a LIMIT clause Example: `select /*+ parallel(5) */ count(*) from t1;`
TOPK	TOPK(param1 param2)	SELECT	Obtains an approximate value. On-Line Analytical Processing (OLAP) queries that retrieve large amounts of data are time-consuming. In some cases where accuracy is not the major concern, you can reduce the query time by setting a lower accuracy rate. `select /*+ topk(90 1000) */ sum(c2), c1 from t1 group by c1 order by sum(c2) limit 10` In the TOPK(param1 param2) syntax, note that no comma exists between the two parameters. The first parameter is an integer that ranges from 0 to 100. The value of the first parameter indicates the accuracy rate of results. For example, 100 indicates full accuracy, and 90 indicates 90% accuracy. A higher value of the first parameter indicates that the result is more accurate, but the query process consumes a longer length of time. The second parameter is a positive integer that starts from 1. This value is related to the execution of the SQL statement. This approximation algorithm takes effect for only queries that contain the GROUP BY, ORDER BY, and LIMIT clauses.

hint	Parameter	Applicable statement	Description

LOG_LEVEL	• ERROR
	• USER_ERROR
	• WARN
	• INFO
	• TRACE
	• DEBUG
		All DML statements	Defines the log level for a statement.
			Example:
			`select /*+log_level('debug')*/ * from t;`
			`select /*+log_level('sql.*:debug, common.*:info')*/ * from t;`

USE_LATE_MATERIALIZATION NO_USE_LATE_MATERIALIZATION	N/A	N/A	USE_LATE_MATERIALIZATION: generates a late materialization plan for a query. NO_USE_LATE_MATERIALIZATION: generates a materialization plan for a query. This plan is not a late materialization plan. ApsaraDB for OceanBase V1.0 provides the late materialization feature. The system determines whether to automatically rewrite an index scan in the form of `self join` based on the cost. You can use this feature if you need to rewrite a large number of SQL statements in the form of `self join` .

USE_HASH
NO_USE_HASH
	USE_HASH(relation1 [comma] relation2)
	NO_USE_HASH(relation1 [comma] relation2)
		SELECT	Specifies whether to use the `hash join` algorithm to join tables.

5.15.6. HELP statements

Syntax

- Query the help information for all SQL queries that are supported by ApsaraDB for OceanBase.

```
HELP
```

- Query the help information for the syntax that is used at the SQL server.

```
HELP contents
```

- Query the help information for a specified SQL query.

```
HELP searchstring
```

Examples

- Statement: `mysql> help;`

  Output:

```
For information about MySQL products and services, visit:
   http://www.mysql.com/
For developer information, including the MySQL Reference Manual, visit:
   http://dev.mysql.com/
To buy MySQL Enterprise support, training, or other products, visit:
   https://shop.mysql.com/
List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing binlog with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
For server side help, type 'help contents'
ERROR 1046 (3D000): No database selected
root@(none) 04:59:24>select * from t1;
```

- Statement: `mysql> help contents;`

  Output:

```
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the following
categories:
   Data Types
   Functions
   Operator
   Escape character
   Data Definition
   Data Manipulation
   Transaction Statements
   Prepared Statements
   Compound Statements
   Administration
   Utility
```

- Statement: `mysql> help use`

  Output:

```
Name: 'USE'
Description:
```

5.16. OUTLINE

5.16.1. Overview

You can use the outline feature to specify throttling rules or execution plans for SQL statements.

When you test statements that include hints, you must enable the `-c` option on the MySQL client. If the -c option is disabled, the statements do not take effect.

5.16.2. CREATE OUTLINE

You can execute the `CREATE OUTLINE` statement to create an outline. Based on the outline, the optimizer of ApsaraDB for OceanBase generates a throttling rule or an execution plan for the specified SQL statements.

Syntax

```
CREATE [OR REPLACE] OUTLINE outline_name ON stmt [ TO target_stmt ];
```

where:

- The stmt parameter specifies a data manipulation language (DML) statement that contains a hint.
- The hint in stmt determines whether to implement the throttling or plan stability feature.
- `TO target_stmt` specifies the SQL statements on which you want to implement the throttling or plan stability feature.
- `OR REPLACE` indicates that you can replace an existing execution plan or a throttling rule.

  Note A throttling rule and an execution plan can replace each other.

Plan stability

The plan stability feature is used to generate the specified execution plans for SQL statements. After you create an outline to enable the plan stability feature, the optimizer of ApsaraDB for OceanBase generates an execution plan based on the specified hint for the specified SQL statements.

Notes:

- If the specified hint is not MAX_CONCURRENT(NUM), the `CREATE OUTLINE` statement enables the plan stability feature instead of the throttling feature.
- If you create an outline to enable the throttling feature, you can specify question marks (`?`). However, if you create an outline to enable the plan stability feature, you cannot specify question marks (`?`). When the CREATE OUTLINE statement is executed, the portions that can be represented by parameters are converted to question marks (?). If you specify question marks (`?`) in this scenario, the system returns the following error: `invalid outline` .
- One outline_name is associated with only one execution plan.
- If you do not specify a hint in stmt for the CREATE OUTLINE statement, the statement creates an outline to enable the plan stability feature. In this scenario, the optimizer of ApsaraDB for OceanBase generates the execution plan based on the outline.

Example:

```
OceanBase (root@oceanbase)> create outline ol_1 on  select /*+index(t1 c1)*/* from t1 where c1 =1;
```

In this example, the ol_1 outline is created. For the statements that meet the `SELECT * FROM t1 WHERE c1 =?` condition, execution plans are generated based on `/*+ BEGIN_OUTLINE_DATA INDEX(@"SEL$1" "oceanbase.t1"@"SEL$1" "c1") END_OUTLINE_DATA */` .

Throttling

If you specify the MAX_CONCURRENT(NUM) hint in the CREATE OUTLINE statement, the statement applies a throttling rule on the specified SQL statements. The throttling rule controls the maximum number of SQL statements that can be concurrently executed in an OBServer.

Notes:

- Similar to the plan stability feature, the throttling feature is implemented based on the specified hint. To use the throttling feature, specify the MAX_CONCURRENT(NUM) hint. In the hint, NUM specifies the maximum number of SQL statements that can be concurrently executed.

  If the number of concurrent SQL statements exceeds the upper limit, the server returns the following error: `SQL reach max concurrent num` .
- You can execute the `CREATE OUTLINE` statement to enable the throttling or plan stability feature. To ensure the correctness and clarity of semantics, do not specify the throttling hint together with other types of hints in one `CREATE OUTLINE` statement.
- When you create throttling rules, you can use question marks (`?`) to distinguish variable parameters and constant parameters.

  Note If you execute the CREATE OUTLINE statement to enable the plan stability feature, you cannot specify question marks (`?`). In this scenario, you must use parameters to represent constants.

- One outline_name can be associated with multiple throttling rules that have the same signature.

- Each throttling rule specifies the maximum number of concurrent statements. If an SQL statement matches multiple throttling rules, the system uses the throttling rule that specifies the smallest value among the upper limits.

Example:

```
OceanBase (root@oceanbase)> create outline ol_1 on select /*+max_concurrent(1)*/ * from t2 where c1 = 1 and c2 = ? ;
```

In this example, the ol_1 outline is created. For the SQL statements that meet the `SELECT * FROM t2 WHERE c1 = 1 and c2 = ?` condition, only one of the statements is executed on a single OBServer.

`c2 = ?` indicates that the question mark can be replaced with a constant value.

For example, throttling is implemented on the following SQL statements:

```
select  * from t2 where c1 = 1 and c2 = 1;
select  * from t2 where c1 = 1 and c2 = 2;
select  * from t2 where c1 = 1 and c2 = "2";
select  * from t2 where c1 = 1 and c2 = true;
```

`c1=1` indicates that throttling is implemented on only SQL statements that contain `c1=1`. For example, throttling is not implemented on the following SQL statements:

```
select  * from t2 where c1 = 2 and c2 = 1;
select  * from t2 where c1 = "2" and c2 = 2;
select  * from t2 where c1 = false and c2 = "2";
```

5.16.3. ALTER OUTLINE

Syntax

```
ALTER OUTLINE outline_name ADD stmt [ TO target_stmt ]
```

You can use TO target_stmt in this statement and TO target_stmt in the `CREATE OUTLINE` statement in the same way. You can specify TO target_stmt if you need to implement the throttling or plan stability feature on the SQL statements that contain the corresponding hints.

Examples

- Execute the ALTER OUTLINE statement to add a throttling rule.

```
OceanBase (root@oceanbase)> alter outline ol_1 add select /*+max_concurrent(1)*/ * from t1 where c1 = 1 and c2 = ? ;
OceanBase (root@oceanbase)> alter outline ol_1 add select /*+max_concurrent(1)*/ * from t1 where c1 = ? and c2 = 1;
```

- Execute the ALTER OUTLINE statement to add an execution plan.

```
OceanBase (root@oceanbase)> create outline ol_2 on select /*+max_concurrent(1)*/ * from t1,t2 where t1.c1 = 1;
OceanBase (root@oceanbase)> alter outline ol_2 add select /*+use_nl(t2)*/ * from t1,t2 where t1.c1 = 1;
```

Notes

- One outline_name is associated with only one execution plan. If you execute the `CREATE OUTLINE` statement to specify an execution plan, you cannot add another execution plan by executing the ALTER OUTLINE statement.
- When you execute the `ALTER OUTLINE` statement, you can specify only a throttling rule or an execution plan for each execution. The similar rule applies to the `CREATE OUTLINE` statement.
- To ensure that the `ALTER OUTLINE` statement is valid, the value of the outline_name parameter must match the signature of the outline.

5.16.4. DROP OUTLINE

You can execute the DROP OUTLINE statement to drop an outline for an ApsaraDB for OceanBase database.

Syntax

```
DROP OUTLINE outline_name;
```

Examples

```
DROP OUTLINE ol_1;
```

Errors

- If the specified outline does not exist, the ApsaraDB for Oceanbase database returns the following error: `ERROR HY000: Outline 'XXX' doesn't exist` .
- If no database is selected for the current session, the ApsaraDB for Oceanbase database returns the following error: `ERROR 3D000: No database selected` .

5.16.5. Others

5.16.5.1. Considerations

Considerations on tenants

All outline-related features such as throttling and plan stability take effect within a tenant. You cannot use the system tenant to create, modify, or drop outlines for other tenants. To create an outline for a tenant, you must execute the CREATE OUTLINE statement within this tenant.

Considerations on databases

When you create an outline, the namespace of stmt is parsed based on the database of the current session. The throttling or plan stability feature takes effect only if the specified SQL statements are executed in this database.

The following scenarios are two examples:

- Assume that you execute the `CREATE OUTLINE ol_1 ON SELECT/*+use_nl(t1)*/ * FROM t2,t1` statement in the db1 database. When you execute the `SELECT * FROM t2` statement in database db2, the query result is not affected by the ol_1 outline.
- If an SQL statement is executed and no database is specified, you cannot create an outline for the SQL statement.

Considerations on MySQL clients

Assume that you execute the CREATE OUTLINE statement on a MySQL client. The MySQL client adds a space between the hint and the first SELECT expression in the statement if the two elements are not separated with characters. The test results show that this issue does not occur if Java and Python are used to establish connections.

The following code shows an example:

```
OceanBase (root@oceanbase)> create outline ol_1 on select /*+full(t1)*/* from t1;
OceanBase (root@oceanbase)> select sql_text from __all_outline;
+--------------------------------+
| sql_text                       |
+--------------------------------+
| select /*+full(t1)*/ * from t1 |
+--------------------------------+
1 row in set (0.60 sec)
```

Concurrency

The concurrency settings limit the number of SQL statements that can be concurrently executed on a single OBServer based on physical execution plans. Therefore, the maximum number of the specified SQL statements that can be concurrently executed for a cluster is greater than that specified in the hint.

Assume that you specify a throttling rule that allows only one of the statements that meet the `SELECT * FROM t2 WHERE c1 = ?` condition to be executed on a single OBServer. These statements are affected by the following factors:

The throttling feature is implemented on a per-OBServer basis. If the OBProxy sends `SELECT * FROM t2 WHERE c1 = 1` to OBServer 1 and OBServer 2, two SQL statements are concurrently executed in the cluster.

ApsaraDB for OceanBase implements throttling by matching SQL statements to physical execution plans based on the signatures of the SQL statements. Therefore, the database separately controls the maximum number of concurrent SQL statements for each physical execution plan in the plan cache.

Signature

`create After you create an outline` , a record is generated in the system table __all_outline. The signature column of the record indicates the unique identifier of the outline and is used in multiple scenarios.

A signature is generated based on the SQL statement and the corresponding not_params. In the SQL statement, the hint is excluded and constants are replaced with question marks (?).

Note In the latest code, the parameter value of not_param is backfilled into the SQL statement. Therefore, the signature is generated based on only the SQL statement.

When a data manipulation language (DML) statement is executed, a signature is generated. The system checks whether an outline that has the same signature already exists.

When you create or add throttling rules or outlines, all throttling rules or outlines that are subordinate to the same outline name must have the same signature.

When you execute the `CREATE OUTLINE` statement, the system returns an `already exist` error if the outline name or the signature already exists.

`create or replace When you create an outline and use it to replace an existing outline`, the operation succeeds only if the outline name and the signature are matched. If only one of the two attributes is matched, the system returns an `already exist` error.

When you modify an outline, the system checks whether the signature of the stmt that is specified in the `ALTER OUTLINE` statement matches the signature of the outline that is specified by outline_name.

5.16.5.2. Related system tables

Table name	Property	Description
__all_outline	System table	This table stores meta information about outlines.
__all_outline_history	System table	This table stores meta information about outlines.
__tenant_virtual_outline	Virtual table	This table shows the outline information for the current tenant. You can use this table to migrate outlines.
gv$outline	View	This table is created based on the __tenant_virtual_outline table.
dba_outlines	View	This table is created based on the __all_outline table.
__tenant_virtual_concurrent_limit_sql	Virtual table	This table shows the throttling rules of the current tenant.
gv$concurrent_limit_sql	View	This table is created based on the __tenant_virtual_concurrent_limit_sql table.

5.17. Hierarchical queries

You can use hierarchical queries to view hierarchical data that is organized based on hierarchy levels.

Hierarchical data is organized based on hierarchy levels. The hierarchy levels are frequent occurrences in real-life events. The following examples are used to explain hierarchy levels:

- Relationships between leaders and members in an organization
- Relationships between superior and subordinate departments in an enterprise
- Relationships between web pages that are connected by hyperlinks

ApsaraDB for OceanBase uses the CONNECT BY clause to perform hierarchical queries. ApsaraDB for OceanBase also provides relevant virtual columns and functions for hierarchical queries.

Syntax

```
SELECT select_list
FROM table_expression
[ WHERE ... ]
[ START WITH start_expression ]
CONNECT BY [NOCYCLE] { PRIOR child_expr = parent_expr | parent_expr = PRIOR child_expr }
[ ORDER SIBLINGS BY ...]
[ GROUP BY ... ]
[ HAVING ... ]
[ ORDER BY ... ]
```

Execution process

The key to using a hierarchical query is to understand the execution process. The following steps show the process to run a hierarchical query:

1. The ApsaraDB for OceanBase system performs the specified SCAN or JOIN operation that follows the FROM keyword.
2. The ApsaraDB for OceanBase system generates hierarchical relationships based on the START WITH and CONNECT BY clauses.
3. The ApsaraDB for OceanBase system executes the remaining clauses in the hierarchical query by following the same process of running general queries. The clauses include the WHERE, GROUP, and ORDER BY clauses.

The following steps show the detailed process to generate hierarchical relationships:

1. The ApsaraDB for OceanBase system queries the root rows based on the expression in the START WITH clause.
2. The ApsaraDB for OceanBase system generates the child rows of each root row based on the expression in the CONNECT BY clause.
3. The ApsaraDB for OceanBase system uses the generated child rows as new root rows to further generate child rows. The system repeats this step until no new row is generated.

5.18. Materialized views

Materialized views are a special type of views that store or materialize query results to accelerate specific queries. You can also use materialized views to implement read/write splitting.

Syntax

Create a materialized view

```
CREATE MATERIALIZED VIEW view_name AS subquery
```

Drop a materialized view

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

Limits

Only materialized views that join two tables can be created. The following limits apply to materialized views:

- Only user-created physical tables can be joined. Views, virtual tables, system tables, and other materialized views cannot be joined.
- A join condition must contain the primary key columns of the right table.
- You must specify R{all_server} for the right table. To simplify the procedure, specify the ALL_SERVER type for your cluster. This ensures that each server in the zone of your cluster has a read-only replica of the right table.
- Aliases cannot be specified for tables.
- The PROJECTION, JOIN, and `ORDER BY` operations are supported. You must include the `ORDER BY` clause in the statement.
- The LIMIT clause is not supported.
- AGGREGATE operations are not supported.
- Set operations such as UNION, UNION ALL, and INTERSECT are not supported.
- The ROWNUM clause is not supported.
- You can specify only original columns of the left table in the `ORDER BY` clause. You cannot specify expressions or generated columns in the clause.
- You can specify only original columns of the left table and the right table in the PROJECTION clause. You cannot specify expressions or generated columns in the clause.
- Each join condition must use the equal to (=) operator that is applied on an original column of the left table and an original column of the right table. You cannot specify expressions or generated columns in the join condition. You can use the AND operator to specify multiple join conditions. The OR operator is not supported. The specified original column in each join condition must uniquely identify each row of the right table. The primary key or unique key can uniquely identify each row. However, you can use only the primary key of the right table when you specify the join condition.
- Nested subqueries are not supported.
- You cannot specify columns that have the same name in the PROJECTION clause.

The following limits apply to data definition language (DDL) operations on a materialized view:

- A materialized view takes effect only after a daily major freeze operation is performed.
- When you modify the attributes of a column in the left table or the right table, follow the existing modification rules if the column does not affect the materialized view.
- If the column affects the materialized view, modify the attributes in a compatible manner. For example, you can change the data type of a column from int to bigint, or from `varchar(10)` to `varchar(20)` .

- To delete a column that is used by a materialized view, you must drop the materialized view first.
- To truncate the left table or the right table that is used by the materialized view, you must drop the materialized view first.
- To drop a right table that is used by a materialized view, you must drop the materialized view first.
- When you drop a left table, the materialized view that uses the left table is also dropped.
- A materialized view cannot be truncated.
- After you drop a left table that is used by a materialized view, the table and the view cannot be recovered from the recycle bin.

Examples

```
create table collect_info (user_id int, item_id int, info_collect_time timestamp, primary key(user_id, item_id));
create table collect_item(item_id int primary key, item_url varchar(5000), item_price int) locality = 'FULL{1},READONLY{A
LL_SERVER}@zone1';
create view info_item_view as select user_id,  collect_info.item_id,  info_collect_time,  item_url  from collect_info joi
n collect_item  on collect_info.item_id = collect_item.item_id  order by user_id, info_collect_time;  create materialized
view info_item_view as select user_id,  collect_info.item_id,  info_collect_time,  item_url  from collect_info join colle
ct_item  on collect_info.item_id = collect_item.item_id  order by user_id, info_collect_time;
```

To query data from info_item_view, you can read the data that is stored in the materialized view info_item_mv and do not need to perform the JOIN operation.

The materialized view can also accelerate the following query:

```
select user_id,  collect_info.item_id,  info_collect_time,  item_url  from collect_info join collect_item  on collect_inf
o.item_id = collect_item.item_id where user_id = 1  order by user_id, info_collect_time;
```

This statement is rewritten to the following statement at the SQL layer:

```
select * from info_item_view where user_id = 1
```

You can execute the following EXPLAIN statement to identify whether a query is performed based on the materialized view:

```
mysql> explain select user_id, collect_info.item_id, info_collect_time, item_url from collect_info join collect_item on c
ollect_info.item_id = collect_item.item_id where user_id = 1;
 ====================== |ID|OPERATOR  |NAME          |EST. ROWS|COST| ----------------------- |0 |TABLE SCAN|(info_item_m
v)|1        |30  | ======================= Outputs & filters:
---------------------------   0 - output([info_item_mv.user_id], [info_item_mv.item_id], [info_item_mv.info_collect_time]
, [info_item_mv.item_url]), filter([info_item_mv.item_id = info_item_mv.item_id]),        access([info_item_mv.item_id],
[info_item_mv.item_id], [info_item_mv.user_id], [info_item_mv.info_collect_time], [info_item_mv.item_url]), partitions(p0
)
```

5.19. SQL modes

You can perform operations on an ApsaraDB for OceanBase server in different SQL modes for different clients. You can specify the SQL mode of the server for each application based on your business needs.

An SQL mode determines the SQL syntax that ApsaraDB for OceanBase supports and the data validation checks that ApsaraDB for OceanBase performs. This allows you to use ApsaraDB for OceanBase in different environments.

You can set the SQL mode by specifying the `--sql-mode="modes"` option. To enable the strict mode, specify either or both of the `STRICT_TRANS_TABLES` and `STRICT_ALL_TABLES` modes.

ApsaraDB for OceanBase supports the following SQL modes:

- HIGH_NOT_PRECEDENCE

  By default, the NOT operator has a lower precedence than other operators. For example, the expression `NOT a BETWEEN b AND c` is parsed as `NOT (a BETWEEN b AND c)`. If you need the expression to be parsed as `(NOT a) BETWEEN b AND c`, enable the HIGH_NOT_PRECEDENCE SQL mode. This SQL mode improves the precedence of the NOT operator.

- ONLY_FULL_GROUP_BY

  This SQL mode rejects queries for nonaggregated columns that are not specified in the `GROUP BY` clause.

- PAD_CHAR_TO_FULL_LENGTH

  By default, spaces are truncated from values that are retrieved from CHAR columns. If the PAD_CHAR_TO_FULL_LENGTH mode is enabled, truncation does not occur. This mode does not apply if you retrieve values from VARCHAR columns.

  Example:

```
mysql> CREATE TABLE t1 (c1 CHAR(10));
Query OK, 0 rows affected (0.09 sec)
mysql> INSERT INTO t1 (c1) VALUES('xy');
Query OK, 1 row affected (0.00 sec)
mysql> SET sql_mode = '';      //The default SQL mode
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------+-----------------+
| c1   | CHAR_LENGTH(c1) |
+------+-----------------+
| xy   |               2 |
+------+-----------------+
1 row in set (0.00 sec)
mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';  //Change the SQL mode.
Query OK, 0 rows affected (0.00 sec)
mysql>  SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------------+-----------------+
| c1         | CHAR_LENGTH(c1) |
+------------+-----------------+
| xy         |              10 |
+------------+-----------------+
1 row in set (0.02 sec)
```

- PIPES_AS_CONCAT

  This SQL mode treats `||` as a string concatenation operator (+) rather than as a synonym for the OR operator. II functions the same as CONCAT().

- STRICT_ALL_TABLES

  This SQL mode enables the strict SQL mode for all storage engines. Invalid values are rejected.

- STRICT_TRANS_TABLES

  This SQL mode enables the strict SQL mode for transactional storage engines. In some cases, this SQL mode may also enable the strict mode for non-transactional storage engines.

  The strict mode controls how ApsaraDB for OceanBase handles invalid or missing values in statements. A value can be invalid for several reasons. For example, a value is of an invalid data type for the column, or a value is out of a specified range. If a new row to be inserted does not contain a value for a column that has no explicit Default clause in its definition, the value is missing.

  For transactional tables, if either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled, ApsaraDB for OceanBase returns an error if invalid or missing values are found in a statement. The statement is aborted and rolled back.

  For non-transactional tables, if a bad value occurs in the first row to be inserted or updated, ApsaraDB for OceanBase handles in the same way under either of the two strict modes. The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and a bad value occurs in the second or later row, the processing method of ApsaraDB for OceanBase depends on the strict mode that is enabled:

  - If `STRICT_ALL_TABLES` is enabled, ApsaraDB for OceanBase returns an error and ignores the rest of the rows. However, a partial update is complete because the earlier rows are inserted or updated. This may not meet your requirements. To avoid this issue, we recommend that you use single-row statements. If you abort a single-row statement, your original table remains unchanged.
  - If `STRICT_TRANS_TABLES` is enabled, ApsaraDB for OceanBase converts an invalid value to the closest valid value for the column and inserts the adjusted value. If a value is missing, ApsaraDB for OceanBase inserts the implicit default value in the column. In all cases, ApsaraDB for OceanBase generates a warning rather than an error and continues to execute the statement.

    In strict mode, invalid dates such as `2004-04-31` are not allowed. Dates that have zero parts, such as `2004-04-00`, or zero dates are also rejected as invalid dates.

    If you do not enable the `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` mode, ApsaraDB for OceanBase adjusts the invalid or missing values, inserts the adjusted values, and then generates warnings.

- NO_AUTO_VALUE_ON_ZERO

  `NO_AUTO_VALUE_ON_ZERO` affects how you handle `AUTO_INCREMENT` columns. In most cases, you can insert NULL or 0 into a column to generate the next sequence number for the column. However, 0 is disabled in `NO_AUTO_VALUE_ON_ZERO` mode. You can insert only NULL to generate the next sequence number.

5.20. System views

5.20.1. Overview

ApsaraDB for OceanBase provides a complete set of system views for you to check the running status of the system. The name of each system view starts with `V$` or `GV$`. `V$` indicates a view for a single OBServer and `GV$` indicates a view for all OBServers.

5.20.2. v$statname

Shows definitions of all statistical events.

View definition

```
view_definition =
  "select tenant_id as CON_ID,
  stat_id as STAT_ID,
  `statistic#` as `STATISTIC#`,
  name as NAME,
  display_name as DISPLAY_NAME,
  class as CLASS
  from oceanbase.__tenant_virtual_statname"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
STAT_ID	bigint(20)	The ID of the statistical event.
STATISTICS#	bigint(20)	The number that represents the statistical event.
NAME	varchar(64)	The name of the statistical event.
DISPLAY_NAME	varchar(64)	The alias of the statistical event.
CLASS	bigint(20)	The alias of the class of the statistical event.

5.20.3. v$event_name

Shows definitions of all wait events.

View definition

```
  view_definition =
  "select tenant_id as CON_ID,
  event_id as EVENT_ID,
  `event#` as `EVENT#`,
  name as NAME,
  display_name as DISPLAY_NAME,
  parameter1 as PARAMETER1,
   parameter2 as PARAMETER2,
  parameter3 as PARAMETER3,
  wait_class_id as WAIT_CLASS_ID,
  `wait_class#` as `WAIT_CLASS#`,
  wait_class as WAIT_CLASS
  from oceanbase.__tenant_virtual_event_name"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
EVENT_ID	bigint(20)	The ID of the wait event.
EVENT#	bigint(20)	The number that represents the wait event.
NAME	varchar(64)	The name of the wait event.

Name	Data type	Description
DISPLAY_NAME	varchar(64)	The alias of the wait event.
PARAMETER1	varchar(64)	The first parameter of the wait event.
PARAMETER2	varchar(64)	The second parameter of the wait event.
PARAMETER3	varchar(64)	The third parameter of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CALSS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.

5.20.4. v$session_event

Shows the wait events that occur in each session.

View definition

```
view_definition =
"select SID,CON_ID,EVENT_ID,EVENT,WAIT_CLASS_ID,
`WAIT_CLASS#`,WAIT_CLASS,TOTAL_WAITS,TOTAL_TIMEOUTS,TIME_WAITED,
 MAX_WAIT,AVERAGE_WAIT,TIME_WAITED_MICRO
from oceanbase.gv$session_event
where SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
SID	bigint(20)	The ID of the session.
CON_ID	bigint(20)	The ID of the tenant.
EVENT_ID	bigint(20)	The ID of the wait event.
EVENT	varchar(64)	The description of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
TOTAL_WAITS	bigint(20)	The total number of waits for the event.
TOTAL_TIMEOUTS	bigint(20)	The total number of wait time-outs for the event.
TIME_WAITED	double	The total amount of time that was spent on waiting for the event. Unit: 10 ms.
MAX_WAIT	double	The maximum amount of time that was spent on waiting for the event. Unit: 10 ms.
AVERAGE_WAIT	double	The average amount of time that was spent on waiting for the event. Unit: 10 ms.
TIME_WAITED_MICRO	bigint(20)	The total amount of time that was spent on waiting for the event. Unit: microseconds.

5.20.5. v$session_wait

Shows details about the current wait event for each session. For each wait event, ApsaraDB for OceanBase provides three parameters to record the corresponding values.

View definition

```
view_definition =
  "select SID,CON_ID,EVENT,P1TEXT,P1,
  P2TEXT,P2,P3TEXT,P3,WAIT_CLASS_ID,
  `WAIT_CLASS#`,WAIT_CLASS,STATE,WAIT_TIME_MICRO,TIME_REMAINING_MICRO,
  TIME_SINCE_LAST_WAIT_MICRO
  from oceanbase.gv$session_wait
  where SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
SID	bigint(20)	The ID of the session.
CON_ID	bigint(20)	The ID of the tenant.
EVENT	varchar(64)	The description of the wait event.
P1TEXT	varchar(64)	The name of the first parameter of the wait event.
P1	bigint(20) unsigned	The value of the first parameter of the wait event.
P2TEXT	varchar(64)	The name of the second parameter of the wait event.
P2	bigint(20) unsigned	The value of the second parameter of the wait event.
P3TEXT	varchar(64)	The name of the third parameter of the wait event.
P3	bigint(20) unsigned	The value of the third parameter of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
STATE	varchar(19)	The state of the wait event.
WAIT_TIME_MICRO	bigint(20)	The amount of time that was spent on waiting for the event. Unit: microseconds.
TIME_REMAINING_MICRO	bigint(20)	The amount of time that is remaining for the current wait before it times out. Unit: microseconds.
TIME_SINCE_LAST_WAIT_MICRO	bigint(20)	The amount of time that elapsed between the current wait event and the previous wait event. Unit: microseconds.

5.20.6. v$session_wait_history

Shows details about the last 10 wait events for each session.

View definition

```
view_definition =
  "SELECT SID,CON_ID,`SEQ#`,`EVENT#`,EVENT,
  P1TEXT,P1,P2TEXT,P2,P3TEXT,
  P3,WAIT_TIME_MICRO,TIME_SINCE_LAST_WAIT_MICRO,WAIT_TIME
   FROM oceanbase.gv$session_wait_history
   WHERE SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

For more information about the fields, see the Fields section in v$session_wait.

5.20.7. v$sesstat

Shows summary information about statistical events based on sessions.

View definition

```
view_definition =
  "select SID,CON_ID,`STATISTIC#`,VALUE from oceanbase.gv$sesstat"
  where SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
SID	bigint(20)	The ID of the session.
CON_ID	bigint(20)	The ID of the tenant.
STATISTICS#	bigint(20)	The number that represents the statistical event.
VALUE	bigint(20)	The total number of occurrences of the statistical event.

5.20.8. v$sysstat

Shows summary information about statistical events based on tenants.

View definition

```
view_definition =
  "SELECT CON_ID,`STATISTIC#`,VALUE,STAT_ID,NAME,CLASS from oceanbase.gv$sysstat
  WHERE SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
STATISTICS#	bigint(20)	The number that represents the statistical event.
VALUE	bigint(20)	The result value of the statistical event.
STAT_ID	bigint(20)	The ID of the statistical event.
NAME	varchar(64)	The name of the statistical event.
CLASS	bigint(20)	The class of the statistical event.

5.20.9. v$system_event

Shows statistics about wait events based on tenants.

View definition

```
view_definition =
  "SELECT CON_ID,EVENT_ID,EVENT,WAIT_CLASS_ID,`WAIT_CLASS#`,WAIT_CLASS,TOTAL_WAITS,TOTAL_TIMEOUTS,TIME_WAITED,AVERAGE_WAI
T, TIME_WAITED_MICRO
  FROM oceanbase.gv$system_event
  WHERE SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
EVENT_ID	bigint(20)	The ID of the wait event.
EVENT	varchar(64)	The description of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
TOTAL_WAITS	bigint(20)	The total number of waits for the event.
TOTAL_TIMEOUTS	bigint(20)	The total number of wait time-outs for the event.
TOTAL_WAITED	double	The total amount of time that was spent on waiting for the event. Unit: 10 ms.
AVERAGE_WAIT	double	The average amount of time that was spent on waiting for the event. Unit: 10 ms.
TOTAL_WAITED_MICRO	bigint(20)	The total amount of time that was spent on waiting for the event. Unit: microseconds.

5.20.10. v$memory

Shows memory statistics based on tenants.

View definition

```
  view_definition =
  "SELECT TENANT_ID, CONTEXT, COUNT, USED, ALLOC_COUNT, FREE_COUNT
  FROM oceanbase.gv$memory
  WHERE IP=HOST_IP() AND PORT=RPC_PORT()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
CONTEXT	varchar(256)	The name of the mod to which the memory belongs.
COUNT	bigint(20)	The number of memory units that are used in the mod, which is the difference between the number of allocated units and the number of available units.
USED	bigint(20)	The memory size that is used in the mod.
ALLOC_COUNT	bigint(20)	The total number of memory units that are allocated to the mod.

Name	Data type	Description
FREE_COUNT	bigint(20)	The total number of available memory units in the mod.

5.20.11. v$memstore

Shows MemStore statistics based on tenants.

View definition

```
  view_definition =
        "SELECT TENANT_ID, ACTIVE, TOTAL, `FREEZE_TRIGGER`, `MEM_LIMIT`
        FROM oceanbase.gv$memstore
        WHERE SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
ACTIVE	bigint(20)	The active MemStore size of the tenant.
TOTAL	bigint(20)	The total MemStore size of the tenant.
FREEZE_TRIGGER	bigint(20)	The MemStore size that triggers a major freeze action.
MEM_LIMIT	bigint(20)	The maximum MemStore size of the tenant.

5.20.12. v$memstore_info

Shows MemStore statistics based on tenants. The statistics include the details about all partitions.

View definition

```
 view_definition =
  "SELECT TENANT_ID, IP, PORT, TABLE_ID, PARTITION_ID, VERSION, IS_ACTIVE, USED, HASH_ITEMS, BTREE_ITEMS
  FROM oceanbase.gv$memstore_info
  WHERE SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
TABLE_ID	bigint(20)	The ID of the table.
PARTITION_ID	bigint(20)	The ID of the partition.
VERSION	varchar(128)	The version number.
IS_ACTIVE	bigint(20)	Indicates whether the MemStore is active.
USED	bigint(20)	The memory size that is used by the MemStore.
HASH_ITEMS	bigint(20)	The hash indexes that are contained in the MemStore.

Name	Data type	Description
BTREE_ITEMS	bigint(20)	The B-tree indexes that are contained in the MemStore.

5.20.13. v$plan_cache_stat

Shows the basic status of a plan cache.

View definition

```
view_definition=
  "SELECT tenant_id,svr_ip,svr_port,sql_num,mem_used,access_count,hit_count,hit_rate,plan_num,mem_limit,hash_bucket,stmtk
ey_num
  FROM oceanbase.gv$plan_cache_stat
  WHERE svr_ip=HOST_IP() AND svr_port=RPC_PORT()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
SQL_NUM	bigint(20)	The number of SQL statements for which execution plans are stored in the plan cache for future use.
MEM_USED	bigint(20)	The memory size that is used by the plan cache.
ACCESS_COUNT	bigint(20)	The number of accesses to the plan cache.
HIT_COUNT	bigint(20)	The number of plan cache hits.
HIT_RATE	bigint(20)	The plan cache hit ratio.
PLAN_NUM	bigint(20)	The number of execution plans in the plan cache.
MEM_LIMIT	bigint(20)	The maximum memory size that can be used by the plan cache.
HASH_BUCKET	bigint(20)	The number of buckets that are contained in the `hash map` of the plan cache.
STMTKEY_NUM	bigint(20)	The number of stmt_keys in the plan cache.

5.20.14. v$plan_cache_plan_stat

Shows the status of all execution plans in a plan cache.

View definition

```
view_definition=
  "SELECT tenant_id, svr_ip, svr_port, plan_id, sql_id, type, statement, plan_hash, first_load_time, schema_version, merg
ed_version, last_active_time, avg_exe_usec, slowest_exe_time, slowest_exe_usec, slow_count, hit_count, executions, disk_r
eads, direct_writes, buffer_gets, application_wait_time, concurrency_wait_time, user_io_wait_time, rows_processed, elapse
d_time, cpu_time, large_querys, delayed_large_querys, outline_version, outline_id
  FROM oceanbase.gv$plan_cache_plan_stat
  WHERE svr_ip=HOST_IP() AND svr_port=RPC_PORT()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address of the server.
SVR_PORT	bigint(20)	The port number of the server.
PLAN_ID	bigint(20)	The ID of the execution plan.
SQL_ID	varchar(32)	The ID of the SQL statement.
TYPE	bigint(20)	The type of the execution plan.
STATEMENT	varchar(4096)	The text of the SQL statement.
PLAN_HASH	bigint(20) unsigned	The hash value of the execution plan.
FIRST_LOAD_TIME	timestamp(6)	The time when the execution plan was loaded for the first time.
SCHEMA_VERSION	bigint(20)	The version number of the schema.
MERGED_VERSION	bigint(20)	The merge version number that corresponds to the cached execution plan.
LAST_ACTIVE_TIME	timestamp(6)	The time of the last execution.
AVG_EXE_USEC	bigint(20)	The average amount of execution time.
SLOWEST_EXE_TIME	timestamp(6)	The time when the slowest execution started.
SLOWEST_EXE_USEC	bigint(20)	The amount of time that was spent on the slowest execution.
SLOW_COUNT	bigint(20)	The number of slow queries.
HIT_COUNT	bigint(20)	The number of hits.
EXECUTIONS	bigint(20)	The number of executions.
DISK_READS	bigint(20)	The number of physical reads from disks.
DIRECT_WRITES	bigint(20)	The number of physical writes.
BUFFER_GETS	bigint(20)	The number of logical reads.
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
ROWS_PROCESSED	bigint(20)	The number of rows to be processed by the SQL statement. If a SELECT statement is executed, the value indicates the number of rows that are returned. If a DELETE, UPDATED, or INSERT statement is executed, the value indicates the number of rows that are affected.
ELAPSED_TIME	bigint(20) unsigned	The amount of time that was spent on processing the request after the request was received.
CPU_TIME	bigint(20) unsigned	The CPU time.

5.20.15. v$plan_cache_plan_explain

Shows basic information about all execution plans in a plan cache.

View definition

```
view_definition=
  "SELECT *
  FROM oceanbase.gv$plan_cache_plan_explain
  WHERE IP =HOST_IP() AND PORT = RPC_PORT()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
PLAN_ID	bigint(20)	The ID of the execution plan.
OPERATOR	varchar(128)	The name of the operator.
NAME	varchar(256)	The name of the table.
ROWS	bigint(20)	The estimated number of result rows.
COST	bigint(20)	The estimated cost.
PROPERTY	varchar(256)	The information about the operator.

5.20.16. v$sql_audit

Shows an SQL audit table that records the source and execution status of each SQL statement.

View definition

```
 view_definition='SELECT * FROM oceanbase.gv$sql_audit WHERE svr_ip=HOST_IP() AND svr_port=RPC_PORT()'
```

Fields

Name	Data type	Description
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
REQUEST_ID	bigint(20)	The ID of the request.
SQL_EXEC_ID	bigint(20)	The unique ID of the SQL statement. This ID is the MD5 hash value of the parameterized string of the SQL statement.
TRACE_ID	varchar(128)	The trace ID of the SQL statement.
SID	bigint(20) unsigned	The ID of the session.
CLIENT_IP	varchar(32)	The IP address of the client that sent the request.
CLIENT_PORT	bigint(20)	The port number of the client that sent the request.

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant that sent the request.
TENANT_NAME	varchar(64)	The name of the tenant that sent the request.
USER_ID	bigint(20)	The ID of the user that sent the request.
USER_NAME	varchar(64)	The name of the user that sent the request.
DB_ID	bigint(20) unsigned	The ID of the database where the SQL statement was executed.
DB_NAME	varchar(128)	The name of the database where the SQL statement was executed.
SQL_ID	varchar(32)	The ID of the SQL statement.
QUERY_SQL	varchar(65536)	The text of the SQL statement.
AFFECTED_ROWS	bigint(20)	The number of affected rows.
RETURN_ROWS	bigint(20)	The number of returned rows.
RET_CODE	bigint(20)	The return code of the execution result.
EVENT	varchar(64)	The name of the event for which the amount of waited time is the longest.
P1TEXT	varchar(64)	The name of the first parameter of the wait event.
P1	bigint(20) unsigned	The value of the first parameter of the wait event.
P2TEXT	varchar(64)	The name of the second parameter of the wait event.
P2	bigint(20) unsigned	The value of the second parameter of the wait event.
P3TEXT	varchar(64)	The name of the third parameter of the wait event.
P3	bigint(20) unsigned	The value of the third parameter of the wait event.
LEVEL	bigint(20)	The level of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
STATE	varchar(19)	The state of the wait event.
WAIT_TIME_MICRO	bigint(20)	The amount of time that was spent on waiting for the event. Unit: microseconds.
TOTAL_WAIT_TIME_MICRO	bigint(20)	The total amount of time for all wait events during the execution. Unit: microseconds.
TOTAL_WAITS	bigint(20)	The total number of waits during the execution.
RPC_COUNT	bigint(20)	The number of RPC requests that were sent.

Name	Data type	Description
PLAN_TYPE	bigint(20)	The type of the execution plan. Valid values: local, remote, and distribute.
IS_INNER_SQL	tinyint(4)	Indicates whether the request is an internal SQL query.
IS_EXECUTOR_RPC	tinyint(4)	Indicates whether the request is an RPC request.
IS_HIT_PLAN	tinyint(4)	Indicates whether the plan cache is hit.
REQUEST_TIME	bigint(20)	The time when the execution started.
ELAPSED_TIME	bigint(20)	The amount of time that was spent on processing the request after the request was received.
NET_TIME	bigint(20)	The amount of time that elapsed from the time when the RPC request was sent to the time when the RPC request was received.
NET_WAIT_TIME	bigint(20)	The amount of time that elapsed from the time when the request was received to the time when the request was added to the queue.
QUEUE_TIME	bigint(20)	The amount of time for which the request waited in the queue.
DECODE_TIME	bigint(20)	The amount of time that was spent on decoding the request after the request exited the queue.
GET_PLAN_TIME	bigint(20)	The amount of time that elapsed from the time when the SQL statement was processed to the time when an execution plan was generated.
EXECUTE_TIME	bigint(20)	The amount of time that was spent on executing the execution plan.
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
SCHEDULE_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all schedule events.
ROW_CACHE_HIT	bigint(20)	The number of row cache hits.
BLOOM_FILTER_CACHE_HIT	bigint(20)	The number of Bloom filter cache hits.
BLOCK_CACHE_HIT	bigint(20)	The number of block cache hits.
BLOCK_INDEX_CACHE_HIT	bigint(20)	The number of block index cache hits.
DISK_READS	bigint(20)	The number of physical reads from disks.
RETRY_CNT	bigint(20)	The number of times the SQL statement was re-executed from the redo log.
TABLE_SCAN	tinyint(4)	Indicates a table scan.
CONSISTENCY_LEVEL	bigint(20)	The consistency level of the SQL statement.

Name	Data type	Description
MEMSTORE_READ_ROW_COUNT	bigint(20)	The number of rows in the SQL result set that was returned from the MemStore.
SSSTORE_READ_ROW_COUNT	bigint(20)	The number of rows in the SQL result set that was returned from the disk.
REQUEST_MEMOMERY_USED	bigint(20)	The memory size used by the SQL statement.

5.20.17. v$latch

Shows latch information.

View definition

```
view_definition =
  "SELECT *
  FROM oceanbase.gv$latch
  WHERE SVR_IP=host_ip() and SVR_PORT=rpc_port()"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
ADDR	varchar(256)	The value is displayed as NULL.
LATCH#	bigint(20)	The number that represents the class of the latch.
LEVEL#	bigint(20)	The number that represents the level of the latch.
NAME	varchar(256)	The name of the latch.
HASH	bigint(20)	The value is displayed as 0.
GETS	bigint(20)	The number of successful low_lock operations.
MISSES	bigint(20)	The number of waits after the low_lock operations are performed more than the yield operations.
SLEEPS	bigint(20)	The total number of yield operations.
IMMEDIATE_GETS	bigint(20)	The number of successful try_lock operations.
IMMEDIATE_MISSES	bigint(20)	The number of failed try_lock operations.
SPIN_GETS	bigint(20)	The total number of spin operations.
WAIT_TIME	bigint(20)	The amount of time that was spent on waiting for the latch.

5.20.18. v$obrpc_outgoing

Shows statistics about RPC requests that were sent from an OBServer for different tenants based on the RPC packet codes.

View definition

```
view_definition =
  "SELECT *
  FROM gv$obrpc_outgoing
  WHERE IP=HOST_IP() AND PORT=RPC_PORT()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
PCODE	bigint(20)	rpc packet code
PCODE_NAME	varchar(256)	The name that corresponds to the RPC packet code.
COUNT	bigint(20)	The number of times that the RPC packet code was used.
TOTAL_TIME	bigint(20)	The total amount of time that was spent.
TOTAL_SIZE	bigint(20)	The total amount of data that was sent.
FAILURE	bigint(20)	The number of RPC requests that have failed sending.
TIMEOUT	bigint(20)	The number of sending time-outs.
SYNC	bigint(20)	The number of RPC requests that were sent synchronously.
ASYNC	bigint(20)	The number of RPC requests that were sent asynchronously.
LAST_TIMESTAMP	timestamp(6)	The time when the statistics was last updated.

5.20.19. v$obrpc_incoming

Shows statistics about RPC requests that were received by an OBServer for different tenants based on the RPC packet codes.

View definition

```
view_definition =
  "SELECT *
  FROM gv$obrpc_incoming
  WHERE IP=HOST_IP() AND  PORT=RPC_PORT()"
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
PCODE	bigint(20)	rpc packet code
PCODE_NAME	varchar(256)	The name that corresponds to the RPC packet code.

Name	Data type	Description
COUNT	bigint(20)	The number of times that the RPC packet code was used.
TOTAL_SIZE	bigint(20)	The total amount of data that was received.
NET_TIME	bigint(20)	The network time.
WAIT_TIME	bigint(20)	The amount of time that elapsed from the time when the request was received to the time when the request was added to the queue.
QUEUE_TIME	bigint(20)	The amount of time for which the request waited in the queue.
PROCESS_TIME	bigint(20)	The amount of time that was spent on processing the request.
LAST_TIMESTAMP	timestamp(6)	The last update time.

5.20.20. v$sql

Shows hot-updated SQL statistics for all execution plans. Each row in the table records the statistics of a single execution plan. The statistics contain summary information about multiple executions of the plan.

View definition

```
view_definition=
 "SELECT *
 FROM oceanbase.gv$sql
 WHERE svr_ip=HOST_IP() AND svr_port=RPC_PORT()"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
PLAN_ID	bigint(20)	The ID of the execution plan.
SQL_ID	varchar(32)	The identifier of the SQL statement.
TYPE	bigint(20)	The type of the execution plan. Valid values: local, remote, and distribute.
SQL_TEXT	varchar(4096)	The text of the SQL statement.
PLAN_HASH_VALUE	bigint(20) unsigned	The hash value of the execution plan.
FIRST_LOAD_TIME	timestamp(6)	The time of the first execution.
LAST_ACTIVE_TIME	timestamp(6)	The time of the last execution.
AVG_EXE_USEC	bigint(20)	The average amount of execution time.
SLOWEST_EXE_TIME	timestamp(6)	The time when the slowest execution started.
SLOWEST_EXE_USEC	bigint(20)	The amount of time that was spent on the slowest execution.

Name	Data type	Description
SLOW_COUNT	bigint(20)	The number of slow queries.
HIT_COUNT	bigint(20)	The number of plan cache hits.
PLAN_SIZE	bigint(20)	
EXECUTIONS	bigint(20)	The number of executions.
DISK_READS	bigint(20)	The number of physical reads from disks.
DIRECT_WRITES	bigint(20)	The number of physical writes.
BUFFER_GETS	bigint(20)	The number of logical reads.
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
ROWS_PROCESSED	bigint(20)	The number of rows to be processed by the SQL statement. If a SELECT statement is executed, the value indicates the number of rows that are returned. If a DELETE, UPDATED, or INSERT statement is executed, the value indicates the number of rows that are affected.
ELAPSED_TIME	bigint(20) unsigned	The amount of time that was spent on processing the request after the request was received.
CPU_TIME	bigint(20) unsigned	The CPU time.

5.20.21. v$sql_monitor

Shows statistics about slow SQL queries based on execution plans. Each slow SQL query has a statistical record that can be used to trace the execution plan.

View definition

```
view_definition=
  "SELECT tenant_id as CON_ID,\
         request_id as SQL_EXEC_ID,\
         job_id as JOB_ID,\
         task_id as TASK_ID,\
         svr_ip as SVR_IP,\
         svr_port as SVR_PORT,\
         sql_exec_start as SQL_EXEC_START, \
         plan_id as PLAN_ID,\
         scheduler_ip as SCHEDULER_IP, \
         scheduler_port as SCHEDULER_PORT, \
         monitor_info as MONITOR_INFO,\
         extend_info as EXTEND_INFO FROM oceanbase.__all_virtual_sql_monitor \
         WHERE is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) \
         and (tenant_id = effective_tenant_id() or effective_tenant_id() = 1) "
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.

Name	Data type	Description
SQL_EXEC_ID	bigint(20)	The unique ID of the SQL statement. This ID is the MD5 hash value of the parameterized string of the SQL statement.
JOB_ID	bigint(20)	The job ID. The executor uses this ID to identify a segment in the physical execution plan. Job IDs are globally incremental on a single OBServer.
TASK_ID	bigint(20)	The task ID, which uniquely identifies an execution of a segment that corresponds to a job ID in a distributed execution plan.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
SQL_EXEC_START	timestamp(6)	The time when the execution started.
PLAN_ID	bigint(20)	The plan ID, which uniquely identifies an execution plan in the plan cache on a single OBServer. Plan IDs are incremental values that are managed by the plan cache module. When a new execution plan is added to the plan cache, a plan ID is assigned to the new execution plan.
SCHEDULER_IP	varchar(32)	The IP address of the OBServer that is scheduled to execute the SQL statement.
SCHEDULER_PORT	bigint(20)	The port number of the OBServer that is scheduled to execute the SQL statement.
MONITOR_INFO	varchar(65535)	The relevant information such as the event for which the amount of waited time is the longest and the reception time.
EXTEND_INFO	varchar(65535)	The extended information such as all trace information that was generated in the process of executing the SQL statement.

5.20.22. v$sql_plan_monitor

Shows statistics about slow SQL queries based on execution plans. Each slow SQL query has a statistical record that can be used to trace the execution plan.

View definition

```
view_definition='SELECT * from oceanbase.gv$sql_plan_monitor  WHERE svr_ip=HOST_IP() AND svr_port=RPC_Port()'
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SQL_EXEC_ID	bigint(20)	The unique ID of the SQL statement. This ID is the MD5 hash value of the parameterized string of the SQL statement.
JOB_ID	bigint(20)	The job ID. The executor uses this ID to identify a segment in the physical execution plan. Job IDs are globally incremental on a single OBServer.

Name	Data type	Description
TASK_ID	bigint(20)	The task ID, which uniquely identifies an execution of a segment that corresponds to a job ID in a distributed execution plan.
OPERATION_ID	bigint(20)	The operator ID, which uniquely identifies an operator in a physical execution plan. Operator IDs are generated based on the postorder traversal of the physical execution plan tree.
SVR_IP	varchar(32)	The IP address of the OBServer.
SVR_PORT	bigint(20)	The port number of the OBServer.
SQL_EXEC_START	timestamp(6)	The time when the execution of the SQL statement started.
PLAN_ID	bigint(20)	The plan ID, which uniquely identifies an execution plan in the plan cache on a single OBServer. Plan IDs are incremental values that are managed by the plan cache module. When a new execution plan is added to the plan cache, a plan ID is assigned to the new execution plan.
SCHEDULER_IP	varchar(32)	The IP address of the OBServer that is scheduled to execute the SQL statement.
SCHEDULER_PORT	bigint(20)	The port number of the OBServer that is scheduled to execute the SQL statement.
PLAN_OPERATION	varchar(32)	The operators that are used in the execution plan.
MONITOR_INFO	varchar(65535)	The execution time statistics for each step in the execution plan.
EXTEND_INFO	varchar(65535)	The extended information.

5.20.23. gv$plan_cache_stat

Shows the plan cache status in a cluster. A cluster consists of multiple OBServers where multiple OBServer instances are deployed.

View definition

```
view_definition='
  "SELECT tenant_id,
         svr_ip,
         svr_port,
         sql_num,
         mem_used,
         access_count,
         hit_count,
         hit_rate,
         plan_num,
         mem_limit,
         hash_bucket,
          stmtkey_num
  FROM oceanbase.__all_virtual_plan_cache_stat
  WHERE is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and (tenant_id = effective_tenant_id() or effective_te
nant_id() = 1) "
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.

Name	Data type	Description
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
SQL_NUM	bigint(20)	The number of SQL statements for which the execution plans are stored in all plan caches in the cluster.
MEM_USED	bigint(20)	The memory size used by all plan caches in the cluster.
ACCESS_COUNT	bigint(20)	The number of accesses to all plan caches in the cluster.
HIT_COUNT	bigint(20)	The number of plan cache hits of the cluster.
HIT_RATE	bigint(20)	The plan cache hit ratio of the cluster.
PLAN_NUM	bigint(20)	The number of execution plans that are stored in the plan caches in the cluster.
MEM_LIMIT	bigint(20)	The maximum memory size that can be used by the plan caches in the cluster.
HASH_BUCKET	bigint(20)	The number of buckets that are contained in the hash maps of all plan caches in the cluster.
STMTKEY_NUM	bigint(20)	The number of stmt_keys in all plan caches in the cluster.

5.20.24. gv$plan_cache_plan_stat

Shows the status of each execution plan in the plan caches of each OBServer instance on all OBServers in a cluster.

View definition

```
view_definition=
  "SELECT tenant_id,
         svr_ip,
         svr_port,
         plan_id,
         sql_id,
         type,
         statement,
         plan_hash,
         first_load_time,
         schema_version,
         merged_version,
         last_active_time,
         avg_exe_usec,
         slowest_exe_time,
         slowest_exe_usec,
         slow_count,
         hit_count,
         executions,
         disk_reads,
         direct_writes,
         buffer_gets,
         application_wait_time,
         concurrency_wait_time,
         user_io_wait_time,
         rows_processed,
         elapsed_time,
         cpu_time,
         large_querys,
         delayed_large_querys,
         outline_version,
         outline_id
 FROM
    oceanbase.__all_virtual_plan_stat
WHERE
     is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and (tenant_id = effective_tenant_id() or effective_tenan
t_id() = 1)'
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
PLAN_ID	bigint(20)	The ID of the execution plan.
SQL_ID	varchar(32)	The identifier of the SQL statement.
TYPE	bigint(20)	The type of the execution plan. Valid values: local, remote, and distribute.
STATEMENT	varchar(4096)	The text of the SQL statement.
PLAN_HASH	bigint(20) unsigned	The hash value of the execution plan.
FIRST_LOAD_TIME	timestamp(6)	The time when the execution plan was loaded for the first time.
SCHEMA_VERSION	bigint(20)	The version number of the schema.
MERGED_VERSION	bigint(20)	The merge version that corresponds to the execution plan in the plan cache.
LAST_ACTIVE_TIME	timestamp(6)	The time of the last execution.

Name	Data type	Description
AVG_EXE_USEC	bigint(20)	The average amount of execution time.
SLOWEST_EXE_TIME	bigint(20)	The time when the slowest execution started.
SLOWEST_EXE_USEC	bigint(20)	The amount of time that was spent on the slowest execution.
SLOW_COUNT	bigint(20)	The number of slow queries.
HIT_COUNT	bigint(20)	The number of plan cache hits.
EXECUTIONS	bigint(20)	The number of executions.
DISK_READS	bigint(20)	The number of physical reads from disks.
DIRECT_READS	bigint(20)	The number of direct reads.
DIRECT_WRITES	bigint(20)	The number of physical writes.
BUFFER_GETS	bigint(20)	The number of logical reads.
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
ROWS_PROCESSED	bigint(20)	The number of rows to be processed by the SQL statement. If a SELECT statement is executed, the value indicates the number of rows that are returned. If a DELETE, UPDATED, or INSERT statement is executed, the value indicates the number of rows that are affected.
ELAPSED_TIME	bigint(20) unsigned	The amount of time that was spent on processing the request after the request was received.
CPU_TIME	bigint(20) unsigned	The CPU time.
LARGE_QUERYS	bigint(20)	
DELAYED_LARGE_QUERYS	bigint(20)	
OUTLINE_ID	bigint(20)	

5.20.25. gv$session_event

Shows information about wait events based on sessions for all OBServers in a cluster.

View definition

```
  view_definition = "select session_id as SID, \
                   tenant_id as CON_ID, \
                   svr_ip as SVR_IP, \
                   svr_port as SVR_PORT, \
                   event_id as EVENT_ID, \
                   event as EVENT, \
                   wait_class_id as WAIT_CLASS_ID, \
                   `wait_class#` as `WAIT_CLASS#`, \
                   wait_class as WAIT_CLASS, \
                   total_waits as TOTAL_WAITS, \
                   total_timeouts as TOTAL_TIMEOUTS, \
                   time_waited as TIME_WAITED, \
                   max_wait as MAX_WAIT, \
                   average_wait as AVERAGE_WAIT, \
                   time_waited_micro as TIME_WAITED_MICRO \
                   from oceanbase.__all_virtual_session_event where \
                   is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and \
                   (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)"
```

Fields

Name	Data type	Description
SID	bigint(20)	The ID of the session.
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
EVENT_ID	bigint(20)	The ID of the wait event.
EVENT	varchar(64)	The description of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
TOTAL_WAITS	bigint(20)	The total number of waits for the event.
TOTAL_TIMEOUTS	bigint(20)	The total number of wait time-outs for the event.
TIME_WAITED	double	The total amount of time that was spent on waiting for the event. Unit: 10 ms.
MAX_WAIT	double	The maximum amount of time that was spent on waiting for the event. Unit: 10 ms.
AVERAGE_WAIT	double	The average amount of time that was spent on waiting for the event. Unit: 10 ms.
TIME_WAITED_MICRO	bigint(20)	The total amount of time that was spent on waiting for the event. Unit: microseconds.

5.20.26. gv$session_wait

Shows details about all wait events based on sessions for all OBServers in a cluster.

View definition

```
view_definition = "select session_id as SID, \
                  tenant_id as CON_ID, \
                  svr_ip as SVR_IP, \
                  svr_port as SVR_PORT, \
                  event_id as EVENT_ID, \
                  event as EVENT, \
                  wait_class_id as WAIT_CLASS_ID, \
                  `wait_class#` as `WAIT_CLASS#`, \
                  wait_class as WAIT_CLASS, \
                  total_waits as TOTAL_WAITS, \
                  total_timeouts as TOTAL_TIMEOUTS, \
                  time_waited as TIME_WAITED, \
                  max_wait as MAX_WAIT, \
                  average_wait as AVERAGE_WAIT, \
                  time_waited_micro as TIME_WAITED_MICRO \
                  from oceanbase.__all_virtual_session_event where \
                  is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and \
                  (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)"
```

Fields

Name	Data type	Description
SID	bigint(20)	The ID of the session.
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
EVENT_ID	bigint(20)	The ID of the wait event.
EVENT	varchar(64)	The description of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
TOTAL_WAITS	bigint(20)	The total number of waits for the event.
TOTAL_TIMEOUTS	bigint(20)	The total number of wait time-outs for the event.
TIME_WAITED	double	The total amount of time that was spent on waiting for the event. Unit: 10 ms.
MAX_WAIT	double	The maximum amount of time that was spent on waiting for the event. Unit: 10 ms.
AVERAGE_WAIT	double	The average amount of time that was spent on waiting for the event.
TIME_WAITED_MICRO	bigint(20)	The total amount of time that was spent on waiting for the event. Unit: microseconds.

5.20.27. gv$session_wait_history

Shows details about all wait events based on sessions for all OBServers in a cluster. This view shows more detailed history information than the gv$session_wait view.

View definition

```
  view_definition = "select session_id as SID, \
                   tenant_id as CON_ID, \
                   svr_ip as SVR_IP, \
                   svr_port as SVR_PORT, \
                   `seq#` as `SEQ#`, \
                   `event#` as `EVENT#`, \
                   event as EVENT, \
                   p1text as P1TEXT, \
                   p1 as P1, \
                   p2text as P2TEXT, \
                   p2 as P2, \
                   p3text as P3TEXT, \
                   p3 as P3, \
                   wait_time_micro as WAIT_TIME_MICRO, \
                   time_since_last_wait_micro as TIME_SINCE_LAST_WAIT_MICRO, \
                   wait_time as WAIT_TIME \
                   from oceanbase.__all_virtual_session_wait_history where \
                   is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and \
                   (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)"
```

Fields

Name	Data type	Description
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
SEQ#	bigint(20)	
EVENT#	bigint(20)	The number that represents the wait event.
EVENT	varchar(64)	The description of the wait event.
P1TEXT	varchar(64)	The name of the first parameter of the wait event.
P1	bigint(20) unsigned	The value of the first parameter of the wait event.
P2TEXT	varchar(64)	The name of the second parameter of the wait event.
P2	bigint(20) unsigned	The value of the second parameter of the wait event.
P3TEXT	varchar(64)	The name of the third parameter of the wait event.
P3	bigint(20) unsigned	The value of the third parameter of the wait event.
WAIT_TIME_MICRO	bigint(20)	The amount of time that was spent on waiting for the event. Unit: microseconds.
TIME_SINCE_LAST_WAIT_MICRO	bigint(20)	The amount of time that elapsed between the current wait event and the previous wait event. Unit: microseconds.
WAIT_TIME	double	

5.20.28. gv$system_event

Shows information about wait events based on tenants for all OBServers in a cluster.

View definition

```
view_definition = "select tenant_id as CON_ID, \
                 svr_ip as SVR_IP, \
                 svr_port as SVR_PORT, \
                 event_id as EVENT_ID, \
                 event as EVENT, \
                 wait_class_id as WAIT_CLASS_ID, \
                 `wait_class#` as `WAIT_CLASS#`, \
                 wait_class as WAIT_CLASS, \
                 total_waits as TOTAL_WAITS, \
                 total_timeouts as TOTAL_TIMEOUTS, \
                 time_waited as TIME_WAITED, \
                 average_wait as AVERAGE_WAIT, \
                 time_waited_micro as TIME_WAITED_MICRO \
                 from oceanbase.__all_virtual_system_event  where \
                 is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and \
                 (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
EVENT_ID	bigint(20)	The ID of the wait event.
EVENT	varchar(64)	The description of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
TOTAL_WAITS	bigint(20)	The total number of waits for the event.
TOTAL_TIMEOUTS	bigint(20)	The total number of wait time-outs for the event.
TIME_WAITED	double	The total amount of time that was spent on waiting for the event. Unit: 10 ms.
MAX_WAIT	double	The maximum amount of time that was spent on waiting for the event. Unit: 10 ms.
AVERAGE_WAIT	double	The average amount of time that was spent on waiting for the event. Unit: 10 ms.
TIME_WAITED_MICRO	bigint(20)	The total amount of time that was spent on waiting for the event. Unit: microseconds.

5.20.29. gv$sesstat

Shows information about statistical events based on sessions for all OBServers in a cluster.

View definition

```
view_definition = "select session_id as SID, \
                 tenant_id as CON_ID, \
                 svr_ip as SVR_IP, \
                 svr_port as SVR_PORT, \
                 `statistic#` as `STATISTIC#`, \
                 value as VALUE\
                 from oceanbase.__all_virtual_sesstat \
                 where is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and can_visible = true and \
                 (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)"
```

Fields

Name	Data type	Description
SID	bigint(20)	The ID of the session.
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
STATISTIC#	bigint(20)	The number that represents the statistical event.
VALUE	bigint(20)	

5.20.30. gv$sysstat

Shows information about statistical events based on tenants for all OBServers in a cluster.

View definition

```
view_definition = "select tenant_id as CON_ID, \
                 svr_ip as SVR_IP, \
                 svr_port as SVR_PORT, \
                 `statistic#` as `STATISTIC#`, \
                 value as VALUE, \
                 stat_id as STAT_ID, \
                 name as NAME, \
                 class as CLASS \
                 from oceanbase.__all_virtual_sysstat \
                 where is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and can_visible = true and \
                (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)"
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
STATISTIC#	bigint(20)	The number that represents the statistical event.
VALUE	bigint(20)	
STAT_ID	bigint(20)	The ID of the statistical event.
NAME	varchar(64)	The name of the statistical event.
CLASS	bigint(20)	The alias of the class of the statistical event.

5.20.31. gv$sql_audit

Shows an SQL audit table for all OBServers in a cluster.

View definition

```
 view_definition = 'select \
                        svr_ip as SVR_IP,      \
                        svr_port as SVR_PORT,      \
                        request_id as REQUEST_ID,      \
                        execution_id as SQL_EXEC_ID,  \
                        trace_id as TRACE_ID,      \
                        client_ip as CLIENT_IP,      \
                        client_port as CLIENT_PORT,      \
                        tenant_id as TENANT_ID,      \
                        tenant_name as TENANT_NAME,      \
                        user_id as USER_ID,      \
                        user_name as USER_NAME,      \
                        sql_id as SQL_ID,      \
                        query_sql as QUERY_SQL,      \
                        affected_rows as AFFECTED_ROWS,      \
                        return_rows as RETURN_ROWS,      \
                        ret_code as RET_CODE,      \
                        event as EVENT,      \
                        p1text as P1TEXT,      \
                        p1 as P1,      \
                        p2text as P2TEXT,      \
                        p2 as P2,      \
                        p3text as P3TEXT,      \
                        p3 as P3,      \
                        level as LEVEL,      \
                        wait_class_id as WAIT_CLASS_ID,      \
                        `wait_class#` as `WAIT_CLASS#`,      \
                        wait_class as WAIT_CLASS,      \
                        state as STATE,      \
                        wait_time_micro as WAIT_TIME_MICRO,      \
                        total_wait_time_micro as TOTAL_WAIT_TIME_MICRO,      \
                        total_waits as TOTAL_WAITS,      \
                        rpc_count as RPC_COUNT,      \
                        plan_type as PLAN_TYPE,      \
                        is_inner_sql as IS_INNER_SQL,      \
                        is_executor_rpc as IS_EXECUTOR_RPC,      \
                        is_hit_plan as IS_HIT_PLAN,      \
                        request_time as REQUEST_TIME,      \
                        elapsed_time as ELAPSED_TIME,      \
                        net_time as NET_TIME,      \
                        net_wait_time as NET_WAIT_TIME,      \
                        queue_time as QUEUE_TIME,      \
                        decode_time as DECODE_TIME,      \
                        get_plan_time as GET_PLAN_TIME,      \
                        execute_time as EXECUTE_TIME,      \
                        application_wait_time as APPLICATION_WAIT_TIME,      \
                        concurrency_wait_time as CONCURRENCY_WAIT_TIME,      \
                        user_io_wait_time as USER_IO_WAIT_TIME,      \
                        schedule_time as SCHEDULE_TIME,      \
                        disk_reads as DISK_READS      \
                    from oceanbase.__all_virtual_sql_audit  \
                    where is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and (tenant_id = effective_tenant_i
d() or effective_tenant_id() = 1)'
```

Fields

Name	Data type	Description
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.

Name	Data type	Description
REQUEST_ID	bigint(20)	The ID of the request.
EXECUTION_ID	bigint(20)	The unique ID of the SQL statement. This ID is the MD5 hash value of the parameterized string of the SQL statement.
TRACE_ID	varchar(128)	The trace ID of the SQL statement.
CLIENT_IP	varchar(32)	The IP address of the client that sent the request.
CLIENT_PORT	bigint(20)	The port number of the client that sent the request.
TENANT_ID	bigint(20)	The ID of the tenant that sent the request.
TENANT_NAME	varchar(64)	The name of the tenant that sent the request.
USER_ID	bigint(20)	The ID of the user that sent the request.
USER_NAME	varchar(64)	The name of the user that sent the request.
SQL_ID	varchar(32)	The identifier of the SQL statement.
QUERY_SQL	varchar(65536)	The text of the SQL statement.
PLAN_ID	bigint(20)	The ID of the execution plan.
AFFECTED_ROWS	bigint(20)	The number of affected rows.
RETURN_ROWS	bigint(20)	The number of returned rows.
RET_CODE	bigint(20)	The return code of the execution result.
EVENT	varchar(64)	The description of the wait event.
P1TEXT	varchar(64)	The name of the first parameter of the wait event.
P1	bigint(20) unsigned	The value of the first parameter of the wait event.
P2TEXT	varchar(64)	The name of the second parameter of the wait event.
P2	bigint(20) unsigned	The value of the second parameter of the wait event.
P3TEXT	varchar(64)	The name of the third parameter of the wait event.
P3	bigint(20) unsigned	The value of the third parameter of the wait event.
LEVEL	bigint(20)	The level of the wait event.
WAIT_CLASS_ID	bigint(20)	The ID of the class of the wait event.
WAIT_CLASS#	bigint(20)	The number that represents the class of the wait event.
WAIT_CLASS	varchar(64)	The name of the class of the wait event.
STATE	varchar(19)	The state of the wait event.
WAIT_TIME_MICRO	bigint(20)	The amount of time that was spent on waiting for the event. Unit: microseconds.
TOTAL_WAIT_TIME_MICRO	bigint(20)	The total amount of time for all wait events during the execution. Unit: microseconds.
TOTAL_WAITS	bigint(20)	The total number of waits during the execution.
RPC_COUNT	bigint(20)	The number of RPC requests that were sent.

Name	Data type	Description
PLAN_TYPE	bigint(20)	The type of the execution plan. Valid values: local, remote, and distribute.
IS_INNER_SQL	tinyint(4)	Indicates whether the request is an internal SQL query.
IS_EXECUTOR_RPC	tinyint(4)	Indicates whether the request is an RPC request.
IS_HIT_PLAN	tinyint(4)	Indicates whether the plan cache is hit.
REQUEST_TIME	bigint(20)	The time when the execution started.
ELAPSED_TIME	bigint(20)	The amount of time that was spent on processing the request after the request was received.
NET_TIME	bigint(20)	The amount of time that elapsed from the time when the RPC request was sent to the time when the RPC request was received.
NET_WAIT_TIME	bigint(20)	The amount of time that elapsed from the time when the request was received to the time when the request was added to the queue.
QUEUE_TIME	bigint(20)	The amount of time for which the request waited in the queue.
DECODE_TIME	bigint(20)	The amount of time that was spent on decoding the request after the request exited the queue.
GET_PLAN_TIME	bigint(20)	The amount of time that elapsed from the time when the SQL statement was processed to the time when an execution plan was generated.
EXECUTE_TIME	bigint(20)	The amount of time that was spent on executing the execution plan.
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
SCHEDULE_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all schedule events.
DISK_READS	bigint(20)	The number of physical reads from disks.

5.20.32. gv$latch

Shows latch information for all OBServers in a cluster.

View definition

```
view_definition = ' select tenant_id as CON_ID, \
                  svr_ip as SVR_IP,    \
                     svr_port as SVR_PORT,   \
                     addr as ADDR,     \
                     latch_id as `LATCH#`,   \
                     level as `LEVEL#`,    \
                     name as NAME,     \
                     hash as HASH,     \
                     gets as GETS,     \
                    misses as MISSES,      \
                    sleeps as SLEEPS,      \
                    immediate_gets as IMMEDIATE_GETS,    \
                    immediate_misses as IMMEDIATE_MISSES,   \
                    spin_gets as SPIN_GETS,     \
                    wait_time as WAIT_TIME from oceanbase.__all_virtual_latch where        \
                    is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and        \
                    (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)'
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
ADDR	varchar(256)	The value is displayed as NULL.
LATCH_ID#	bigint(20)	The number that represents the class of the latch.
LEVEL#	bigint(20)	The number that represents the level of the latch.
NAME	varchar(256)	The name of the latch.
HASH	bigint(20)	The value is displayed as 0.
GETS	bigint(20)	The number of successful low_lock operations.
MISSES	bigint(20)	The number of waits after the low_lock operations are performed more than the yield operations.
SLEEPS	bigint(20)	The total number of yield operations.
IMMEDIATE_GETS	bigint(20)	The number of successful try_lock operations.
IMMEDIATE_MISSES	bigint(20)	The number of failed try_lock operations.
SPIN_GETS	bigint(20)	The total number of spin operations.
WAIT_TIME	bigint(20)	The amount of time that was spent on waiting for the latch.

5.20.33. gv$memory

Shows memory statistics based on tenants for all OBServers in a cluster.

View definition

```
view_definition = ' SELECT      \
     tenant_id as TENANT_ID,      \
     svr_ip AS IP,       \
     svr_port AS PORT,      \
 mod_id as MOD_ID,       \
 mod_type as MOD_TYPE,      \
     mod_name AS CONTEXT,       \
     count as COUNT,        \
 zone as ZONE,       \
     used as USED,        \
     alloc_count as ALLOC_COUNT,             \
     free_count as FREE_COUNT      \
FROM                                  \
    oceanbase.__all_virtual_memory_info     \
WHERE             \
        (         \
            effective_tenant_id()=1     \
        OR          \
            tenant_id=effective_tenant_id()    \
        )           \
    AND           \
        mod_type='user'        \
    AND            \
        is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) \
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
MOD_ID	bigint(20)	The ID of the mod.
MOD_TYPE	varchar(256)	The type of the mod.
CONTEXT	varchar(256)	The name of the mod to which the memory belongs.
COUNT	bigint(20)	The number of memory units that are used in the mod, which is the difference between the number of allocated units and the number of available units.
ZONE	varchar(256)	
USED	bigint(20)	The memory size that is used in the mod.
ALLOC_COUNT	bigint(20)	The total number of memory units that are allocated to the mod.
FREE_COUNT	bigint(20)	The total number of available memory units in the mod.

5.20.34. gv$memstore

Shows MemStore statistics based on tenants for all OBServers in a cluster.

View definition

```
 view_definition = ' SELECT                                               \
    TENANT_ID,                                    \
    SVR_IP AS IP,                                 \
    SVR_PORT AS PORT,                             \
    ACTIVE_MEMSTORE_USED AS ACTIVE,               \
    TOTAL_MEMSTORE_USED AS TOTAL,                 \
    MAJOR_FREEZE_TRIGGER AS `FREEZE_TRIGGER`,     \
    MEMSTORE_LIMIT AS `MEM_LIMIT`                 \
FROM                                              \
    oceanbase.__all_virtual_tenant_memstore_info  \
WHERE                                             \
        (EFFECTIVE_TENANT_ID()=1                  \
    OR                                            \
        TENANT_ID=EFFECTIVE_TENANT_ID())          \
    AND                                           \
     is_serving_tenant(svr_ip, svr_port, effective_tenant_id())
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
ACTIVE	bigint(20)	The active MemStore size of the cluster.
TOTAL	bigint(20)	The total MemStore size of the cluster.
FREEZE_TRIGGER	bigint(20)	The MemStore size that triggers a major freeze action.
MEM_LIMIT	bigint(20)	The maximum memory size that can be used in the cluster.

5.20.35. gv$memstore_info

Shows MemStore statistics based on tenants for all OBServers in a cluster. The statistics include the details about all partitions.

View definition

```
view_definition=
SELECT                                         \
    TENANT_ID,                                 \
    SVR_IP AS IP,                              \
    SVR_PORT AS PORT,                          \
    table_id AS TABLE_ID,                 \
    partition_idx AS PARTITION_ID,             \
    VERSION,                                   \
    IS_ACTIVE,                                 \
    MEM_USED as USED,                          \
    hash_item_count as HASH_ITEMS,             \
    btree_item_count as BTREE_ITEMS            \
FROM                                           \
    oceanbase.__all_virtual_memstore_info      \
WHERE                                          \
        (EFFECTIVE_TENANT_ID()=1               \
    OR                                         \
        TENANT_ID=EFFECTIVE_TENANT_ID())       \
    AND                                        \
      is_serving_tenant(svr_ip, svr_port, effective_tenant_id())
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
TABLE_ID	bigint(20)	The ID of the table.
PARTITION_ID	bigint(20)	The ID of the partition.
PARTITION_CNT	bigint(20)	The number of partitions.
VERSION	varchar(128)	The version number.
IS_ACTIVE	bigint(20)	Indicates whether the MemStore is active.
USED	bigint(20)	The memory size that is used by the MemStore.
HASH_ITEMS	bigint(20)	The hash indexes that are contained in the MemStore.
BTREE_ITEMS	bigint(20)	The B-tree indexes that are contained in the MemStore.

5.20.36. gv$plan_cache_plan_explain

View definition

```
    view_definition='SELECT TENANT_ID,\
                        SVR_IP as IP, \
                        SVR_PORT as PORT, \
                        PLAN_ID, \
                        OPERATOR, \
                        NAME,\
                        ROWS,\
                        COST,\
                        PROPERTY \
                        FROM oceanbase.__all_virtual_plan_cache_plan_explain \
                        WHERE is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and (EFFECTIVE_TENANT_ID()=1
or TENANT_ID=EFFECTIVE_TENANT_ID())'
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
PLAN_ID	bigint(20)	The ID of the execution plan.
OPERATOR	varchar(128)	The name of the operator.
NAME	varchar(256)	The name of the table.
ROWS	bigint(20)	The estimated number of result rows.
COST	bigint(20)	The estimated cost.
PROPERTY	varchar(256)	-

5.20.37. gv$obrpc_outgoing

Shows statistics about RPC requests that were sent from all OBServers in a cluster.

View definition

```
  view_definition = '''                              \
SELECT                                               \
    TENANT_ID,                                       \
    SVR_IP AS IP,                                    \
    SVR_PORT AS PORT,                                \
    PCODE,                                           \
    PCODE_NAME,                                      \
    COUNT,                                           \
    TOTAL_TIME,                                      \
    TOTAL_SIZE,                                      \
    FAILURE,                                         \
    TIMEOUT,                                         \
    SYNC,                                            \
    ASYNC,                                           \
    LAST_TIMESTAMP                                   \
FROM                                                 \
    oceanbase.__all_virtual_obrpc_stat               \
WHERE                                                \
    is_serving_tenant(svr_ip, svr_port, effective_tenant_id())  \
    AND                                              \
        (EFFECTIVE_TENANT_ID()=1                     \
    OR                                               \
        TENANT_ID=EFFECTIVE_TENANT_ID())             \
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
PCODE	bigint(20)	rpc packet code
PCODE_NAME	varchar(256)	The name that corresponds to the RPC packet code.
COUNT	bigint(20)	The total number of times that the RPC packet code was used.
TOTAL_TIME	bigint(20)	The total amount of time that was spent.
TOTAL_SIZE	bigint(20)	The total amount of data that was sent.
FAILURE	bigint(20)	The number of RPC requests that have failed sending.
TIMEOUT	bigint(20)	The number of sending time-outs.
SYNC	bigint(20)	The number of RPC requests that were sent synchronously.
ASYNC	bigint(20)	The number of RPC requests that were sent asynchronously.
LAST_TIMESTAMP	timestamp(6)	The time when the statistics was last updated.

5.20.38. gv$obrpc_incoming

Shows statistics about RPC requests that were received by all OBServers in a cluster.

View definition

```
  view_definition = '''                     \
SELECT                                      \
    TENANT_ID,                              \
    SVR_IP AS IP,                           \
    SVR_PORT AS PORT,                       \
    PCODE,                                  \
    PCODE_NAME,                             \
    ICOUNT AS COUNT,                        \
    ISIZE AS TOTAL_SIZE,                    \
    NET_TIME,                               \
    WAIT_TIME,                              \
    QUEUE_TIME,                             \
    PROCESS_TIME,                           \
    ILAST_TIMESTAMP AS LAST_TIMESTAMP       \
FROM                                        \
    oceanbase.__all_virtual_obrpc_stat      \
WHERE                                       \
        EFFECTIVE_TENANT_ID()=1             \
    OR                                      \
        TENANT_ID=EFFECTIVE_TENANT_ID()     \
```

Fields

Name	Data type	Description
TENANT_ID	bigint(20)	The ID of the tenant.
IP	varchar(32)	The IP address.
PORT	bigint(20)	The port number.
PCODE	bigint(20)	rpc packet code
PCODE_NAME	varchar(256)	The name that corresponds to the RPC packet code.
COUNT	bigint(20)	The number of times that the RPC packet code was used.
TOTAL_SIZE	bigint(20)	The total amount of data that was received.
NET_TIME	bigint(20)	The network time.
WAIT_TIME	bigint(20)	The amount of time that elapsed from the time when the request was received to the time when the request was added to the queue.
QUEUE_TIME	bigint(20)	The amount of time for which the request waited in the queue.
PROCESS_TIME	bigint(20)	The amount of time that was spent on processing the request.
LAST_TIMESTAMP	timestamp(6)	The last update time.

5.20.39. gv$sql

Shows hot-updated SQL statistics for all execution plans of all OBServers in a cluster. Each row in the table records the statistics of a single execution plan. The statistics contain summary information about multiple executions of the plan.

View definition

```
    view_definition='SELECT tenant_id AS CON_ID, \
        svr_ip AS SVR_IP, \
        svr_port AS SVR_PORT, \
        plan_id AS PLAN_ID, \
        sql_id AS SQL_ID, \
        type AS TYPE, \
        statement AS SQL_TEXT, \
        plan_hash AS PLAN_HASH_VALUE, \
        first_load_time AS FIRST_LOAD_TIME, \
        last_active_time AS LAST_ACTIVE_TIME, \
        avg_exe_usec AS AVG_EXE_USEC, \
        slowest_exe_time AS SLOWEST_EXE_TIME, \
        slowest_exe_usec as SLOWEST_EXE_USEC, \
        slow_count as SLOW_COUNT, \
        hit_count as HIT_COUNT, \
        executions as EXECUTIONS, \
        disk_reads as DISK_READS, \
        direct_writes as DIRECT_WRITES, \
        buffer_gets as BUFFER_GETS,\
        application_wait_time as APPLICATION_WAIT_TIME,\
        concurrency_wait_time as CONCURRENCY_WAIT_TIME, \
        user_io_wait_time as USER_IO_WAIT_TIME, \
        rows_processed as ROWS_PROCESSED, \
        elapsed_time as ELAPSED_TIME, \
        cpu_time as CPU_TIME \
        FROM oceanbase.__all_virtual_plan_stat \
        WHERE is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) and (tenant_id = effective_tenant_id() or effect
ive_tenant_id() = 1)'
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
PLAN_ID	bigint(20)	The ID of the execution plan.
SQL_ID	varchar(32)	The identifier of the SQL statement.
TYPE	bigint(20)	The type of the execution plan. Valid values: local, remote, and distribute.
SQL_TEXT	varchar(4096)	The text of the SQL statement.
PLAN_HASH_VALUE	bigint(20) unsigned	The hash value of the execution plan.
FIRST_LOAD_TIME	timestamp(6)	The time of the first execution.
LAST_ACTIVE_TIME	timestamp(6)	The time of the last execution.
AVG_EXE_USEC	bigint(20)	The average amount of execution time.
SLOWEST_EXE_TIME	timestamp(6)	The time when the slowest execution started.
SLOWEST_EXE_USEC	bigint(20)	The amount of time that was spent on the slowest execution.
SLOW_COUNT	bigint(20)	The number of slow queries.
HIT_COUNT	bigint(20)	The number of plan cache hits.
PLAN_SIZE	bigint(20)	

Name	Data type	Description
EXECUTIONS	bigint(20)	The number of executions.
DISK_READS	bigint(20)	The number of physical reads from disks.
DIRECT_WRITES	bigint(20)	The number of physical writes.
BUFFER_GETS	bigint(20)	The number of logical reads.
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
ROWS_PROCESSED	bigint(20)	The number of rows to be processed by the SQL statement. If a SELECT statement is executed, the value indicates the number of rows that are returned. If a DELETE, UPDATED, or INSERT statement is executed, the value indicates the number of rows that are affected.
ELAPSED_TIME	bigint(20) unsigned	The amount of time that was spent on processing the request after the request was received.
CPU_TIME	bigint(20) unsigned	The CPU time.
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
PLAN_ID	bigint(20)	The ID of the execution plan.
SQL_ID	varchar(32)	The identifier of the SQL statement.
TYPE	bigint(20)	The type of the execution plan. Valid values: local, remote, and distribute.
SQL_TEXT	varchar(4096)	The text of the SQL statement.
PLAN_HASH_VALUE	bigint(20) unsigned	The hash value of the execution plan.
FIRST_LOAD_TIME	timestamp(6)	The time of the first execution.
LAST_ACTIVE_TIME	timestamp(6)	The time of the last execution.
AVG_EXE_USEC	bigint(20)	The average amount of execution time.
SLOWEST_EXE_TIME	timestamp(6)	The time when the slowest execution started.
SLOWEST_EXE_USEC	bigint(20)	The amount of time that was spent on the slowest execution.
SLOW_COUNT	bigint(20)	The number of slow queries.
HIT_COUNT	bigint(20)	The number of plan cache hits.
PLAN_SIZE	bigint(20)	
EXECUTIONS	bigint(20)	The number of executions.
DISK_READS	bigint(20)	The number of physical reads from disks.

Name	Data type	Description
DIRECT_WRITES	bigint(20)	The number of physical writes.
BUFFER_GETS	bigint(20)	The number of logical reads.
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
ROWS_PROCESSED	bigint(20)	The number of rows to be processed by the SQL statement. If a SELECT statement is executed, the value indicates the number of rows that are returned. If a DELETE, UPDATED, or INSERT statement is executed, the value indicates the number of rows that are affected.
ELAPSED_TIME	bigint(20) unsigned	The amount of time that was spent on processing the request after the request was received.
CPU_TIME	bigint(20) unsigned	The CPU time.
CON_ID	bigint(20)	The ID of the tenant.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
PLAN_ID	bigint(20)	The ID of the execution plan.
SQL_ID	varchar(32)	The identifier of the SQL statement.
TYPE	bigint(20)	The type of the execution plan. Valid values: local, remote, and distribute.
SQL_TEXT	varchar(4096)	The text of the SQL statement.
PLAN_HASH_VALUE	bigint(20) unsigned	The hash value of the execution plan.
FIRST_LOAD_TIME	timestamp(6)	The time of the first execution.
LAST_ACTIVE_TIME	timestamp(6)	The time of the last execution.
AVG_EXE_USEC	bigint(20)	The average amount of execution time.
SLOWEST_EXE_TIME	timestamp(6)	The time when the slowest execution started.
SLOWEST_EXE_USEC	bigint(20)	The amount of time that was spent on the slowest execution.
SLOW_COUNT	bigint(20)	The number of slow queries.
HIT_COUNT	bigint(20)	The number of plan cache hits.
PLAN_SIZE	bigint(20)	
EXECUTIONS	bigint(20)	The number of executions.
DISK_READS	bigint(20)	The number of physical reads from disks.
DIRECT_WRITES	bigint(20)	The number of physical writes.
BUFFER_GETS	bigint(20)	The number of logical reads.

Name	Data type	Description
APPLICATION_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all application events.
CONCURRENCY_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all concurrency events.
USER_IO_WAIT_TIME	bigint(20) unsigned	The amount of time that was spent on waiting for all user I/O events.
ROWS_PROCESSED	bigint(20)	The number of rows to be processed by the SQL statement. If a SELECT statement is executed, the value indicates the number of rows that are returned. If a DELETE, UPDATED, or INSERT statement is executed, the value indicates the number of rows that are affected.
ELAPSED_TIME	bigint(20) unsigned	The amount of time that was spent on processing the request after the request was received.
CPU_TIME	bigint(20) unsigned	The CPU time.

5.20.40. gv$sql_monitor

Shows statistics about slow SQL queries based on execution plans for all OBServers in a cluster. Each slow SQL query has a statistical record that can be used to trace the execution plan.

View definition

```
view_definition='SELECT tenant_id as CON_ID,\
     request_id as SQL_EXEC_ID,\
     job_id as JOB_ID,\
     task_id as TASK_ID,\
     svr_ip as SVR_IP,\
     svr_port as SVR_PORT,\
     sql_exec_start as SQL_EXEC_START, \
     plan_id as PLAN_ID,\
     scheduler_ip as SCHEDULER_IP, \
     scheduler_port as SCHEDULER_PORT, \
     monitor_info as MONITOR_INFO,\
     extend_info as EXTEND_INFO FROM oceanbase.__all_virtual_sql_monitor \
     WHERE is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) \
     and (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)',
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SQL_EXEC_ID	bigint(20)	The unique ID of the SQL statement. This ID is the MD5 hash value of the parameterized string of the SQL statement.
JOB_ID	bigint(20)	The job ID. The executor uses this ID to identify a segment in the physical execution plan. Job IDs are globally incremental on a single OBServer.
TASK_ID	bigint(20)	The task ID, which uniquely identifies an execution of a segment that corresponds to a job ID in a distributed execution plan.
SVR_IP	varchar(32)	The IP address.
SVR_PORT	bigint(20)	The port number.
SQL_EXEC_START	timestamp(6)	The time when the execution started.

Name	Data type	Description
PLAN_ID	bigint(20)	The plan ID, which uniquely identifies an execution plan in the plan cache on a single OBServer. Plan IDs are incremental values that are managed by the plan cache module. When a new execution plan is added to the plan cache, a plan ID is assigned to the new execution plan.
SCHEDULER_IP	varchar(32)	The IP address of the OBServer that is scheduled to execute the SQL statement.
SCHEDULER_PORT	bigint(20)	The port number of the OBServer that is scheduled to execute the SQL statement.
MONITOR_INFO	varchar(65535)	The relevant information such as the event for which the amount of waited time is the longest and the reception time.
EXTEND_INFO	varchar(65535)	The extended information such as all trace information that was generated in the process of executing the SQL statement.

5.20.41. gv$sql_plan_monitor

Shows statistics about slow SQL queries based on execution plans for all OBServers in a cluster. Each slow SQL query has a statistical record that can be used to trace the execution plan.

View definition

```
view_definition='SELECT tenant_id as CON_ID, \
      request_id as SQL_EXEC_ID, \
      job_id as JOB_ID, \
      task_id as TASK_ID, \
      operation_id as OPERATION_ID, \
      svr_ip as SVR_IP,\
      svr_port as SVR_PORT,\
      sql_exec_start as SQL_EXEC_START, \
      plan_id as PLAN_ID, \
      scheduler_ip as SCHEDULER_IP, \
      scheduler_port as SCHEDULER_PORT, \
      operation as PLAN_OPERATION, \
      monitor_info as MONITOR_INFO, \
      extend_info as EXTEND_INFO FROM oceanbase.__all_virtual_sql_plan_monitor \
      WHERE is_serving_tenant(svr_ip, svr_port, effective_tenant_id()) \
      and (tenant_id = effective_tenant_id() or effective_tenant_id() = 1)'
```

Fields

Name	Data type	Description
CON_ID	bigint(20)	The ID of the tenant.
SQL_EXEC_ID	bigint(20)	The unique ID of the SQL statement. This ID is the MD5 hash value of the parameterized string of the SQL statement.
JOB_ID	bigint(20)	The job ID. The executor uses this ID to identify a segment in the physical execution plan. Job IDs are globally incremental on a single OBServer.
TASK_ID	bigint(20)	The task ID, which uniquely identifies an execution of a segment that corresponds to a job ID in a distributed execution plan.
OPERATION_ID	bigint(20)	The operator ID, which uniquely identifies an operator in a physical execution plan. Operator IDs are generated based on the postorder traversal of the physical execution plan tree.

Name	Data type	Description
SVR_IP	varchar(32)	The IP address of the OBServer.
SVR_PORT	bigint(20)	The port number of the OBServer.
SQL_EXEC_START	timestamp(6)	The time when the execution of the SQL statement started.
PLAN_ID	bigint(20)	The plan ID, which uniquely identifies an execution plan in the plan cache on a single OBServer. Plan IDs are incremental values that are managed by the plan cache module. When a new execution plan is added to the plan cache, a plan ID is assigned to the new execution plan.
SCHEDULER_IP	varchar(32)	The IP address of the OBServer that is scheduled to execute the SQL statement.
SCHEDULER_PORT	bigint(20)	The port number of the OBServer that is scheduled to execute the SQL statement.
PLAN_OPERATION	varchar(32)	The operators that are used in the execution plan.
MONITOR_INFO	varchar(65535)	The execution time statistics for each step in the execution plan.
EXTEND_INFO	varchar(65535)	The extended information.

5.21. Information Schema

5.21.1. Overview

ApsaraDB for OceanBase supports the INFORMATION_SCHEMA database of MySQL. The stored data in the INFORMATION_SCHEMA database depends on the data that is stored in ApsaraDB for OceanBase. If ApsaraDB for OceanBase does not support a field in the database, the field is retained but the field stores empty values.

The INFORMATION_SCHEMA database stores database metadata.

Metadata is data that provides information about other data, such as database names, table names, data types of columns, and access permissions.

The INFORMATION_SCHEMA database stores the information about all the other databases that are maintained by your server.

The INFORMATION_SCHEMA database contains multiple read-only tables. These read-only tables are views instead of base tables. Therefore, you cannot view the files that are associated with these read-only tables.

Each user can access only specific rows in these read-only tables. The specific rows include the objects on which the user has appropriate read permissions.

Advantages of executing SELECT statements to access metadata

`SELECT ... FROM INFORMATION_SCHEMA` statements are an alternative to SHOW statements. SELECT...FROM INFORMATION_SCHEMA statements provide a consistent method for you to access the information that is returned by the supported SHOW statements. The examples of the supported SHOW statements include SHOW DATABASES and SHOW TABLES statements.

SELECT...FROM INFORMATION_SCHEMA statements offer the following advantages over SHOW statements:

- SELECT...FROM INFORMATION_SCHEMA statements comply with Codd's rules. All the read and write operations are performed on tables.
- You need only to familiarize yourself with the syntax of SELECT statements. You do not need to familiarize yourself with other statements.

  If you are familiar with the syntax of SELECT statements, you need only to know the object names.
- You do not need to concern yourself with the issues of adding keywords.
- SELECT...FROM INFORMATION_SCHEMA statements can return millions of results. This helps you meet various metadata requirements of applications.

Permissions

To query metadata, you can execute SHOW statements or SELECT...FROM INFORMATION_SCHEMA statements. The permissions that are required to execute the two types of statements are the same.

You must have permissions on objects to view the information about the objects.

5.21.2. INFORMATION_SCHEMA tables

Table	Description
INFORMATION_SCHEMA.CHARACTER_SETS	Provides the information about available character sets.
INFORMATION_SCHEMA.COLLATIONS	Provides the information about collations for character sets.
INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY	Specifies the character set that is used for the collation.
INFORMATION_SCHEMA.COLUMNS	Provides the information about table columns.
INFORMATION_SCHEMA.COLUMN_PRIVILEGES	Provides the information about column permissions.
INFORMATION_SCHEMA.ENGINES	Provides the information about storage engines.
INFORMATION_SCHEMA.EVENTS	Provides the information about events.
INFORMATION_SCHEMA.FILES	Provides the information about tablespace data.
INFORMATION_SCHEMA.GLOBAL_STATUS	Provides the information about the status of the global server variables.
INFORMATION_SCHEMA.GLOBAL_VARIABLES	Provides the information about global server variables.
INFORMATION_SCHEMA.KEY_COLUMN_USAGE	Describes key columns that have constraints.
INFORMATION_SCHEMA.OPTIMIZER_TRACE	Provides the information that is produced by the optimizer tracing feature.
INFORMATION_SCHEMA.PARAMETERS	Provides the information about function and method parameters.
INFORMATION_SCHEMA.PARTITIONS	Provides the information about table partitions.
INFORMATION_SCHEMA.PLUGINS	Provides the information about plug-ins.
INFORMATION_SCHEMA.PROCESSLIST	Provides the information about running threads.
INFORMATION_SCHEMA.PROFILING	Provides the information about statement profiling.
INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS	Provides the information about foreign key constraints.
INFORMATION_SCHEMA.ROUTINES	Provides the information about stored subprograms: stored procedures and stored functions.
INFORMATION_SCHEMA.SCHEMATA	Provides the information about databases.
INFORMATION_SCHEMA.SCHEMA_PRIVILEGES	Provides the information about database permissions.
INFORMATION_SCHEMA.SESSION_STATUS	Provides the information about the status of local variables.
INFORMATION_SCHEMA.SESSION_VARIABLES	Provides the information about local variables.
INFORMATION_SCHEMA.STATISTICS	Provides the information about table indexes.
INFORMATION_SCHEMA.TABLES	Provides the information about database tables.
INFORMATION_SCHEMA.TABLESPACES	Provides the information about tablespaces.
INFORMATION_SCHEMA.TABLE_CONSTRAINTS	Describes the tables that have constraints.
INFORMATION_SCHEMA.TABLE_PRIVILEGES	Provides the information about table permissions.
INFORMATION_SCHEMA.TRIGGERS	Provides the information about triggers.
INFORMATION_SCHEMA.USER_PRIVILEGES	Provides the information about user permissions.
INFORMATION_SCHEMA.USER_RECYCLEBIN	-

Table	Description
INFORMATION_SCHEMA.VIEWS	Provides the information about views that are stored in databases.

This topic does not provide details about tables and columns in the INFORMATION_SCHEMA database. These details are provided in other topics. For each column of the tables in the INFORMATION_SCHEMA database, the following three types of information are provided:

- Standard name: specifies the name of the column in an INFORMATION_SCHEMA table. This column name is the standard name that is used in SQL statements.
- SHOW name: specifies the equivalent field name in the latest SHOW statement.
- Remarks: provides additional information about the column.

5.21.3. INFORMATION_SCHEMA.SCHEMATA table

This table provides the information about databases.

Standard name	SHOW name	Remarks
CATALOG_NAME	-	def
SCHEMA_NAME	Database	The name of the database.
DEFAULT_CHARACTER_SET_NAME	-	The default character set.
DEFAULT_COLLATION_NAME	-	The default collation name.
SQL_PATH	-	NULL

The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`
    FROM INFORMATION_SCHEMA.SCHEMATA
    [WHERE SCHEMA_NAME LIKE 'wild']
SHOW DATABASES
  [LIKE 'wild']
```

5.21.4. INFORMATION_SCHEMA.TABLES table

This table provides the information about database tables.

Table information

Standard name	SHOW name	Remarks
TABLE_CATALOG	-	def
TABLE_SCHEMA	Table_...	The name of the schema or database where the table is stored.
TABLE_NAME	Table_...	The name of the table.
TABLE_TYPE		The type of the table. The BASE TABLE value indicates a table, the VIEW value indicates a view, and the TEMPORARY value indicates a temporary table.
ENGINE	Engine	The storage engine.
VERSION	Version	The version number.
ROW_FORMAT	Row_format	-
TABLE_ROWS	Rows	-
AVG_ROW_LENGTH	Avg_row_length	The average row size.
DATA_LENGTH	Data_length	The size of the data file. Unit: bytes.

Standard name	SHOW name	Remarks
MAX_DATA_LENGTH	Max_data_length	The maximum size of the data file. Unit: bytes.
INDEX_LENGTH	Index_length	The size of the index file. Unit: bytes.
DATA_FREE	Data_free	-
AUTO_INCREMENT	Auto_increment	Specifies whether the values are auto-incremented.
CREATE_TIME	Create_time	The time when the table was created.
UPDATE_TIME	Update_time	The time when the table was last updated.
CHECK_TIME	Check_time	The time when the table was last checked.
TABLE_COLLATION	Collation	The table collation.
CHECKSUM	Checksum	The checksum value.
CREATE_OPTIONS	Create_options	The options that are used in the CREATE TABLE statement.
TABLE_COMMENT	Comment	The description.

Notes:

- In the outputs of SHOW statements, the values of the TABLE_SCHEMA and TABLE_NAME columns are included in the values of a single field.
- The valid values in the TABLE_TYPE column include BASE TABLE and VIEW. The BASE TABLE value indicates a table and the VIEW value indicates a view. If you are using a temporary table, set TABLE_TYPE to TEMPORARY: `TABLE_TYPE = TEMPORARY` .
- If the table is stored in the INFORMATION_SCHEMA database, the value in the TABLE_ROWS column is NULL.
- The outputs of query statements do not include the information about the default character set of the table. The TABLE_COLLATION column is disabled because the collation name starts with the character set name.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
     [WHERE table_schema = 'db_name']
     [WHERE|AND table_name LIKE 'wild']
SHOW TABLES
[FROM db_name]
[LIKE 'wild']
```

5.21.5. INFORMATION_SCHEMA.COLUMNS table

This table provides the information about table columns.

Table information

Standard name	SHOW name	Remarks
TABLE_CATALOG	-	def
TABLE_SCHEMA	-	The name of the database.
TABLE_NAME	-	The name of the table.
COLUMN_NAME	Field	The name of the column.
ORDINAL_POSITION	-	The position of the column. For more information, see the "Notes" section in this topic.
COLUMN_DEFAULT	Default	The default value for the column.
IS_NULLABLE	Null	Specifies whether NULL values can be stored in the column.

Standard name	SHOW name	Remarks
DATA_TYPE	Type	The data type of the column.
CHARACTER_MAXIMUM_LENGTH	Type	The maximum length of each string. The length is measured in characters.
CHARACTER_OCTET_LENGTH	-	The maximum length of each string. The length is measured in bytes. For more information, see the "Notes" section in this topic.
NUMERIC_PRECISION	Type	The numeric precision.
NUMERIC_SCALE	Type	The range of the numeric values.
DATETIME_PRECISION	Type	The date and time granularity.
CHARACTER_SET_NAME	-	The name of the character set.
COLLATION_NAME	Collation	The collation name.
COLUMN_TYPE	Type	The data type of the column.
COLUMN_KEY	Key	The column key.
EXTRA	Extra	The additional information about the column.
PRIVILEGES	Privileges	The permissions.
COLUMN_COMMENT	Comment	The description of the column.

Notes:

- In the outputs of SHOW statements, the Type column lists the data types of different columns that are stored in the COLUMNS table.
- ORDINAL_POSITION is required in some scenarios. For example, you may need to execute the ORDER BY ORDINAL_POSITION statement to sort data based on the ordinal positions of columns. Unlike SHOW statements, SELECT statements returns data that is not automatically sorted.
- If single-byte character sets are used, the CHARACTER_OCTET_LENGTH value is the same as the CHARACTER_MAXIMUM_LENGTH value. If multi-byte character sets are used, the values are different.
- You can obtain the CHARACTER_SET_NAME column value based on the COLLATION_NAME column value. For example, if you execute a `SHOW FULL COLUMNS FROM t` statement and the latin1_swedish_ci value is displayed in the COLLATION_NAME column, the name of the character set is latin1. The portion that is located before the first underscore (_) specifies the name of the character set.

The following statements are equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_KEY, COLUMN_DEFAULT, EXTRA
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE table_name = 'tbl_name'
    [AND table_schema = 'db_name']
    [AND column_name LIKE 'wild']
SHOW COLUMNS
  FROM tbl_name
  [FROM db_name]
  [LIKE wild]
```

5.21.6. INFORMATION_SCHEMA.STATISTICS table

This table provides the information about table indexes.

Table information

Standard name	SHOW name	Remarks
TABLE_CATALOG	-	def
TABLE_SCHEMA	-	The name of the database.

Standard name	SHOW name	Remarks
TABLE_NAME	Table	The name of the table.
NON_UNIQUE	Non_unique	-
INDEX_SCHEMA	-	The name of the database.
INDEX_NAME	Key_name	The name of the index.
SEQ_IN_INDEX	Seq_in_index	-
COLUMN_NAME	Column_name	The name of the column.
COLLATION	Collation	-
CARDINALITY	Cardinality	-
SUB_PART	Sub_part	-
PACKED	Packed	-
NULLABLE	Null	-
INDEX_TYPE	Index_type	-
COMMENT	Comment	-
INDEX_COMMENT	-	-
IS_VISIBLE	-	-

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
    WHERE table_name = 'tbl_name'
    [AND table_schema = 'db_name']
SHOW INDEX
    FROM tbl_name
    [FROM db_name]
```

5.21.7. INFORMATION_SCHEMA.USER_PRIVILEGES table

This table provides the information about user permissions.

Table information

Standard name	SHOW name	Remarks
GRANTEE	-	The name of the account to which the permission is granted. For example, the value can be user'@'host.
TABLE_CATALOG	-	def
PRIVILEGE_TYPE	-	The permission type.
IS_GRANTABLE	-	Specifies whether the user has the permission to execute the GRANT OPTION statement.

5.21.8. INFORMATION_SCHEMA.SCHEMA_PRIVILEGES table

This table provides the information about database permissions.

Table information

Standard name	SHOW name	Remarks
GRANTEE	-	The name of the account to which the permission is granted. For example, the value can be user'@'host.
TABLE_CATALOG	-	def
TABLE_SCHEMA	-	-
PRIVILEGE_TYPE	-	The permission type.
IS_GRANTABLE	-	Specifies whether the user has the permission to execute the GRANT OPTION statement. If the user has the permission, the value is YES. If the user does not have the permission, the value is NO.

5.21.9. INFORMATION_SCHEMA.TABLE_PRIVILEGES table

This table provides the information about table permissions.

Standard name	SHOW name	Remarks
GRANTEE	-	The name of the account to which the permission is granted. For example, the value can be user'@'host.
TABLE_CATALOG	-	def
TABLE_SCHEMA	-	The name of the database where the table is stored.
TABLE_NAME	-	The name of the table.
PRIVILEGE_TYPE	-	The permission type.
IS_GRANTABLE	-	Specifies whether the user has the permission to execute the GRANT OPTION statement. If the user has the permission, the value is YES. If the user does not have the permission, the value is NO.

The following statements are not equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
SHOW GRANTS ...
```

The PRIVILEGE_TYPE column stores only one of the following values: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, and CREATE VIEW.

5.21.10. INFORMATION_SCHEMA.CHARACTER_SETS table

This table provides the information about available character sets.

Table information

Standard name	SHOW name	Remarks
CHARACTER_SET_NAME	Charset	The name of the character set.
DEFAULT_COLLATE_NAME	Default collation	The default collation for the character set.
DESCRIPION	Description	The description of the character set.
MAXLEN	Maxlen	The maximum length.

Note The DESCRIPTION and MAXLEN columns are not standard columns. These columns are added to align with the following columns that are displayed in the output of the SHOW CHARACTER SET statement: Description and Maxlen.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
      [WHERE name LIKE 'wild']
SHOW CHARACTER SET
       [LIKE 'wild']
```

5.21.11. INFORMATION_SCHEMA.COLLATIONS table

This table provides the information about collations for character sets.

Table information

Standard name	SHOW name	Remarks
COLLATION_NAME	Collation	The collation name.
CHARACTER_SET_NAME	Charset	The name of the character set.
ID	Id	The collation ID.
IS_DEFAULT	Default	Specifies whether the collation is the default collation for the character set.
IS_COMPILED	Compiled	Specifies whether the character set is compiled.
SORTLEN	Sortlen	The number of bytes in the strings when the strings are sorted in the memory. The strings are expressed based on the character set.

Note The CHARACTER_SET_NAME, ID, IS_DEFAULT, IS_COMPILED, and SORTLEN columns are not standard columns. These columns are added to align with the following columns that are displayed in the output of the `SHOW COLLATION` statement: Charset, Id, Default, Compiled, and Sortlen.

The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
    [WHERE collation_name LIKE 'wild']
SHOW COLLATION
    [LIKE 'wild']
```

5.21.12. INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY table

This table specifies the character set that is used for the collation. This table consists of two columns. The two columns correspond to the first two columns that are displayed in the output of the SHOW COLLATION statement.

Table information

Standard name	SHOW name	Remarks
COLLATION_NAME	Collation	The collation name.
CHARACTER_SET_NAME	Charset	The name of the character set.

5.21.13. INFORMATION_SCHEMA.TABLE_CONSTRAINTS table

This table provides the information about tables that have constraints.

Table information

Standard name	SHOW name	Remarks
CONSTRAINT_CATALOG	-	def
CONSTRAINT_SCHEMA	-	The name of the schema or database to which the constraint belongs.
CONSTRAINT_NAME	-	The name of the constraint.
TABLE_SCHEMA	-	The name of the schema or database where the table is stored.
TABLE_NAME	-	The name of the table.
CONSTRAINT_TYPE	-	The type of the constraint.

Notes:

- The valid values in the CONSTRAINT_TYPE column include UNIQUE, PRIMARY KEY, and FOREIGN KEY.
- If the value of the Non_unique field is 0, the UNIQUE and PRIMARY KEY information is the same as the Key_name column values in the output of the SHOW INDEX statement.
- The CONSTRAINT_TYPE column stores only one of the following values: UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK. The data type of the CONSTRAINT_TYPE column is CHAR instead of ENUM.

  The CHECK value is unavailable before the CHECK constraint is supported by ApsaraDB for OceanBase.

5.21.14. INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS table

This table provides the information about foreign key constraint.

Standard name	SHOW name	Remarks
CONSTRAINT_CATALOG	-	def
CONSTRAINT_SCHEMA	-	The name of the schema or database to which the foreign key constraint belongs.
CONSTRAINT_NAME	-	The name of the foreign key constraint.
UNIQUE_CONSTRAINT_CATALOG	-	def
UNIQUE_CONSTRAINT_SCHEMA	-	The name of the database. The database contains the unique constraint or the primary key that the foreign key constraint references.
UNIQUE_CONSTRAINT_NAME	-	The name of the unique constraint or the primary key that the foreign key constraint references.
MATCH_OPTION	-	The match rule of the foreign key constraint. The value in the column can be FULL, PARTIAL, or NONE.
UPDATE_RULE	-	The update rule of the foreign key constraint. The value in the column can be CASCADE, SET NULL, SET DEFAULT, RESTRICT, or NO ACTION.
DELETE_RULE	-	The deletion rule of the foreign key constraint. The value in the column can be CASCADE, SET NULL, SET DEFAULT, RESTRICT, or NO ACTION.
TABLE_NAME	-	The name of the table.
REFERENCED_TABLE_NAME	-	The name of the table that is referenced by the foreign key constraint.

Note The value in the TABLE_NAME column is the same as that in the TABLE_NAME column of the INFORMATION_SCHEMA.TABLE_CONSTRAINTS table. For more information, see INFORMATION_SCHEMA.TABLE_CONSTRAINTS table.

5.21.15. INFORMATION_SCHEMA.KEY_COLUMN_USAGE table

This table describes the key columns that have constraints.

Table information

Standard name	SHOW name	Remarks
CONSTRAINT_CATALOG	-	def
CONSTRAINT_SCHEMA	-	The name of the database to which the constraint belongs.
CONSTRAINT_NAME	-	The name of the constraint.
TABLE_CATALOG	-	The name of the catalog to which the table belongs. The table contains the field on which the constraint is applied.
TABLE_SCHEMA	-	The name of the schema that stores the table. The table contains the field on which the constraint is applied.
TABLE_NAME	-	The name of the table that includes the constraint. The table contains the field on which the constraint is applied.
COLUMN_NAME	-	The name of the column on which the constraint is implemented.
ORDINAL_POSITION	-	The column position in the constraint. The column position is represented by the sequence number.
POSITION_IN_UNIQUE_CONSTRAINT	-	The ordinal position in the key of the table that is referenced.
REFERENCED_TABLE_SCHEMA	-	The name of the database that is referenced by the constraint.
REFERENCED_TABLE_NAME	-	The name of the table that is referenced by the constraint.
REFERENCED_COLUMN_NAME	-	The name of the column that is referenced by the constraint.

Notes:

- If the constraint is a foreign key, the value in the COLUMN_NAME column is the name of the foreign key. The value is not the name of the column that the foreign key references.
- The value in the ORDINAL_POSITION column specifies the column position in the constraint instead of the column position in the table. Column positions are represented by sequence numbers that start from 1.
- For unique and primary key constraints, the value in the POSITION_IN_UNIQUE_CONSTRAINT column is NULL. For foreign key constraints, the value in the POSITION_IN_UNIQUE_CONSTRAINT column is the ordinal position in the key of the table that is referenced.

For example, execute the following statements to create the t1 and t3 tables:

```
CREATE TABLE t1
 (
s1 INT,
s2 INT,
s3 INT,
PRIMARY KEY(s3)
);
CREATE TABLE t3
 (
s1 INT,
s2 INT,
s3 INT,
KEY(s1),
CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
);
```

The KEY_COLUMN_USAGE table stores two rows for the t1 and t3 tables:

- One row contains `CONSTRAINT_NAME='PRIMARY ', TABLE_NAME='t1', COLUMN_NAME='s3 ', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=NULL.`
- The other row contains `CONSTRAINT_NAME='CO ', TABLE_NAME='t3', COLUMN_NAME='s2 ', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=1` .

5.21.16. INFORMATION_SCHEMA.ROUTINES table

This table provides the information about stored subprograms: stored procedures and stored functions.

Table information

Standard name	Description
SPECIFIC_NAME	The name of the function.
ROUTINE_CATALOG	def
ROUTINE_SCHEMA	The name of the schema or database to which the function belongs.
ROUTINE_NAME	The name of the function.
ROUTINE_TYPE	The type of the stored subprogram. Valid values: PROCEDURE and FUNCTION.
DATA_TYPE	The data type of the return value.
CHARACTER_MAXIMUM_LENGTH	The maximum length of the character set.
CHARACTER_OCTET_LENGTH	The maximum length of the value. If single-byte character sets are used, the `CHARACTER_OCTET_LENGTH` value is the same as the `CHARACTER_MAXIMUM_LENGTH` value. If multi-byte character sets are used, the values are different.
NUMERIC_PRECISION	The numerical precision.
NUMERIC_SCALE	The range of numeric values.
DATETIME_PRECISION	The date and time granularity.
CHARACTER_SET_NAME	The name of the character set.
COLLATION_NAME	The collation name for the character set.
DTD_IDENTIFIER	The identifier of the data type descriptor. The data type is returned by the function. The identifier is unique among the data type descriptors that are assigned to the function.
ROUTINE_BODY	The query language that is used to define the function. The value is always SQL.

Standard name	Description
ROUTINE_DEFINITION	The source code text of the function. If the function is not owned by the current user, the value is NULL.
EXTERNAL_NAME	NULL
EXTERNAL_LANGUAGE	NULL
PARAMETER_STYLE	SQL
IS_DETERMINISTIC	Specifies whether the function is declared as immutable. If the function is declared as immutable, the value is YES. In this scenario, the function is a deterministic function in SQL. If the function is not declared as immutable, the value is NO.
SQL_DATA_ACCESS	The setting of data access. The value is always MODIFIES. This means that the function may modify SQL data.
SQL_PATH	NULL
SECURITY_TYPE	If the function runs based on the permissions of the current user, the value is INVOKER. If the function runs based on the permissions of the user who defines the function, the value is DEFINER.
CREATED	The date and time when the function was created. The value is a TIMESTAMP value.
LAST_ALTERED	The date and time when the stored subprogram was last modified. The value is a TIMESTAMP value.
SQL_MODE	-
ROUTINE_COMMENT	The description.
DEFINER	The user who defines the function.
CHARACTER_SET_CLIENT	The session value of the character_set_client system variable when the function was created.
COLLATION_CONNECTION	The session value of the collation_connection system variable when the stored subprogram was created.
DATABASE_COLLATION	The collation of the database with which the stored subprogram is associated.

5.21.17. INFORMATION_SCHEMA.VIEWS table

This table provides the information about the views that are stored in databases.

Table information

Standard name	SHOW name	Remarks
TABLE_CATALOG	-	def
TABLE_SCHEMA	-	The name of the schema or database to which the view belongs.
TABLE_NAME	-	The name of the view.
VIEW_DEFINITION	-	The SELECT statement that defines the view. If the current user is not the owner of the view, the value is NULL.
CHECK_OPTION	-	NONE
IS_UPDATABLE	-	Specifies whether the view can be updated.
DEFINER	-	The user who created the view.

Standard name	SHOW name	Remarks
SECURITY_TYPE	-	Valid values: DEFINER and INVOKER.
CHARACTER_SET_CLIENT	-	The session value of the character_set_client system variable when the view was created.
COLLATION_CONNECTION	-	The session value of the collation_connection system variable when the view was created.

Notes:

- If you do not have the permission to execute the `SHOW VIEW` statement, you cannot access the INFORMATION_SCHEMA.VIEWS table. This permission is newly added.
- The VIEW_DEFINITION column stores most of the information about the fields that you use to create views. You can execute the `SHOW CREATE VIEW` statement to view the statement that creates the specified view. To obtain the SELECT statement that creates the specified view, skip the words that are located before the SELECT keyword and skip the WITH CHECK OPTION words in the SQL statement.

  For example, you can execute the following SQL statement to create a view:

  ```
   CREATE VIEW v AS
    SELECT s2,s1 FROM t
    WHERE s1 > 5
    ORDER BY s1
    WITH CHECK OPTION;
  ```

  In this example, the view is created based on the following SELECT statement:

  ```
  SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
  ```

  Notes:

  - The value in the CHECK_OPTION column is always NONE.
  - If the view can be updated, the value in the IS_UPDATABLE column is YES. If the view cannot be updated, the value in the column is NO.
  - The DEFINER column specifies the user who defined the view. The valid values in the SECURITY_TYPE column are DEFINER or INVOKER.

5.21.18. INFORMATION_SCHEMA.TRIGGERS table

This table provides the information about triggers.

You must have the SUPER permission to view the INFORMATION_SCHEMA.TRIGGERS table.

Table information

Standard name	SHOW name	Remarks
TRIGGER_CATALOG	-	def
TRIGGER_SCHEMA	-	The name of the database to which the trigger belongs.
TRIGGER_NAME	Trigger	The name of the trigger.
EVENT_MANIPULATION	Event	The event that activates the trigger. The value in the column is INSERT, UPDATE, or DELETE.
EVENT_OBJECT_CATALOG	-	def
EVENT_OBJECT_SCHEMA	-	The name of the database. The database stores the table with which the trigger is associated.
EVENT_OBJECT_TABLE	Table	The name of the table with which the trigger is associated.
ACTION_ORDER	-	-
ACTION_CONDITION	-	NULL

Standard name	SHOW name	Remarks
ACTION_STATEMENT	Statement	The statement that is executed if the trigger is activated. The value in the column is always EXECUTE PROCEDURE. *function*(...).
ACTION_ORIENTATION	-	The value is always ROW. The value indicates that the trigger applies to each processed row.
ACTION_TIMING	Timing	Specifies whether the trigger is activated before or after the event. The valid values are BEFORE and AFTER.
ACTION_REFERENCE_OLD_TABLE	-	NULL
ACTION_REFERENCE_NEW_TABLE	-	NULL
ACTION_REFERENCE_OLD_ROW	-	OLD
ACTION_REFERENCE_NEW_ROW	-	NEW
CREATED	Created	NULL
SQL_MODE	sql_mode	
DEFINER	Definer	The user who created the trigger.
CHARACTER_SET_CLIENT	character_set_client	The session value of the character_set_client system variable when the trigger was created.
COLLATION_CONNECTION	collation_connection	The session value of the collation_connection system variable when the trigger was created.
DATABASE_COLLATION	Database Collation	The collation of the database with which the trigger is associated.

Notes:

- The TRIGGER_NAME column specifies the name of the trigger. The TRIGGER_SCHEMA column specifies the name of the database where the trigger occurs.
- The valid values in the EVENT_MANIPULATION column are INSERT, DELETE, and UPDATE.
- Each trigger is associated with only one table. The EVENT_OBJECT_SCHEMA column specifies the name of the database. The EVENT_OBJECT_TABLE column specifies the name of the table. The database stores the table with which the trigger is associated.
- The ACTION_ORDER column specifies the ordinal position of the trigger action in the list of similar triggers that are applied on the same table. The value in the ACTION_ORDER column is always 0. The system does not allow two or more triggers that have the same EVENT_MANIPULATION and ACTION_TIMING values to take effect on the same table.
- The ACTION_STATEMENT column specifies the statement that is executed if the trigger is activated. The text in this column is the same as the text in the Statement column that is displayed in the output of the `SHOW TRIGGERS` statement.

  Note `SHOW The text in the Statement column in the output of SHOW TRIGGERS` statement uses UTF-8 encoding.

- The value in the ACTION_ORIENTATION column is always ROW.
- The valid values in the ACTION_TIMING column are BEFORE and AFTER.
- The ACTION_REFERENCE_OLD_ROW column specifies the previous column identifier, and the ACTION_REFERENCE_NEW_ROW column specifies the new column identifier. Therefore, the value in the ACTION_REFERENCE_OLD_ROW column is always OLD, and the value in the ACTION_REFERENCE_NEW_ROW column is always NEW.
- The SQL_MODE column specifies the valid server SQL mode that was used when the trigger was created.

  Note The trigger remains valid after the trigger is activated, regardless of the current server SQL mode.

  The value range for the SQL_MODE column is the same as that of the sql_mode system variable.

- The values in the following columns are always NULL: TRIGGER_CATALOG, EVENT_OBJECT_CATALOG, ACTION_CONDITION, ACTION_REFERENCE_OLD_TABLE, and CREATED.

5.21.19. INFORMATION_SCHEMA.TABLESPACE table

This table provides the information about tablespaces. However, ApsaraDB for OceanBase does not use tablespaces. Therefore, the INFORMATION_SCHEMA.TABLESPACE table is not required in ApsaraDB for OceanBase. The INFORMATION_SCHEMA.TABLESPACE table in ApsaraDB for OceanBase is an empty table.

Table information

Standard name	SHOW name	Remarks
TABLE_CATALOG	-	`def`
TABLESPACE_NAME	-	-
ENGINE	-	-
TABLESPACE _TYPE	-	-
LOGFILE_GROUP_NAME	-	-
EXTENT_SIZE	-	-
AUTOEXTEND_SIZE	-	-
MAXIMUM_SIZE	-	-
NODEGROUP_ID	-	-
TABLESPACE _COMMENT	-	-

5.21.20. INFORMATION_SCHEMA.PARTITIONS table

Table information

Standard name	SHOW name	Remarks
TABLE_CATALOG	-	`def`
TABLE_SCHEMA	-	The name of the database.
TABLE_NAME	-	The name of the table.
PARTITION_NAME	-	The name of the partition.
SUBPARTITION_NAME	-	The name of the subpartition.
PARTITION_ORDINAL_POSITION	-	The original ordinal position of the partition.
SUBPARTITION_ORDINAL_POSITION	-	The original ordinal position of the subpartition.
PARTITION_METHOD	-	The partitioning type. For example, hash partitioning or list partitioning is used. The default partitioning type is hash partitioning.
SUBPARTITION_METHOD	-	The subpartitioning type. The default subpartitioning type is hash partitioning.
PARTITION_EXPRESSION	-	The partition expression.
SUBPARTITION_EXPRESSION	-	The subpartition expression.
PARTITION_DESCRIPTION	-	The partition description.
TABLE_ROWS	-	The number of table rows in the partition.
AVG_ROW_LENGTH	-	The average length of the data in the table rows.
DATA_LENGTH	-	The data length. Unit: bytes.

Standard name	SHOW name	Remarks
MAX_DATA_LENGTH	-	The maximum length of the data in the table rows that are stored in the partition or subpartition. Unit: bytes.
INDEX_LENGTH	-	The size of the index file. Unit: bytes.
DATA_FREE	-	
CREATE_TIME	-	The time when the partition or subpartition was created.
UPDATE_TIME	-	The time when the partition or subpartition was last modified.
CHECK_TIME	-	The time when the table that includes the partition or subpartition was last checked.
CHECKSUM	-	The checksum value.
PARTITION_COMMENT	-	The additional information about the partition.
NODEGROUP	-	-
TABLESPACE_NAME	-	The name of the tablespace.

Notes:

- The PARTITIONS table is not a standard SQL table in the INFORMATION_SCHEMA database.
- One row is stored in the PARTITIONS table for each non-partitioned table. The values in the following columns are NULL: `PARTITION_NAME, SUBPARTITION_NAME, PARTITION_ORDINAL_POSITION, SUBPARTITION_ORDINAL_POSITION, PARTITION_METHOD, SUBPARTITION_METHOD, PARTITION_EXPRESSION, SUBPARTITION_EXPRESSION, and PARTITION_DESCRIPTION` . The columns store no data.

5.21.21. INFORMATION_SCHEMA.EVENTS table

This table provides the information about events.

Standard name	SHOW name	Remarks
EVENT_CATALOG	-	def
EVENT_SCHEMA	Db	The name of the database to which the event belongs.
EVENT_NAME	Name	The name of the event.
DEFINER	Definer	user_name'@'host_name'
TIME_ZONE	Time zone	The time zone.
EVENT_BODY	-	The query language.
EVENT_DEFINITION	-	The statement that is executed if the event occurs.
EVENT_TYPE	Type	The repetition type of the event. Events are divided into one-time events and recurring events.
EXECUTE_AT	Execute at	The time when the event occurred.
INTERVAL_VALUE	Interval value	The time interval at which the event recurs.
INTERVAL_FIELD	Interval field	-
SQL_MODE	-	-
STARTS	Starts	The start date and time of a recurring event.
ENDS	Ends	The end date and time of a recurring event.

Standard name	SHOW name	Remarks
STATUS	Status	The event status. Valid values: ENABLED, DISABLED, and SLAVESIDE_DISABLED.
ON_COMPLETION	-	Valid values: PRESERVE and NOT PRESERVE.
CREATED	-	The date and time when the event was created. The value is a TIMESTAMP value.
LAST_ALTERED	-	The date and time when the event was last modified.
LAST_EXECUTED	-	The date and time when the event was last run.
EVENT_COMMENT	-	The description about the event.
ORIGINATOR	Originator	The ID of the server on which the event was created.
CHARACTER_SET_CLIENT	character_set_client	The session value of the character_set_client system variable when the event was created.
COLLATION_CONNECTION	collation_connection	The session value of the collation_connection system variable when the event was created.
DATABASE_COLLATION	Database Collation	The collation of the database with which the event is associated.

5.21.22. INFORMATION_SCHEMA.FILES table

This table provides the information about tablespace data.

Standard name	SHOW name	Remarks
FILE_ID	-	-
FILE_NAME	-	-
FILE_TYPE	-	-
TABLESPACE_NAME	-	-
TABLE_CATALOG	-	-
TABLE_SCHEMA	-	-
TABLE_NAME	-	-
LOGFILE_GROUP_NAME	-	-
LOGFILE_GROUP_NUMBER	-	-
ENGINE	-	-
FULLTEXT_KEYS	-	-
DELETED_ROWS	-	-
UPDATE_COUNT	-	-
FREE_EXTENTS	-	-
TOTAL_EXTENTS	-	-
EXTENT_SIZE	-	-
INITIAL_SIZE	-	-

Standard name	SHOW name	Remarks
MAXIMUM_SIZE	-	-
AUTOEXTEND_SIZE	-	-
CREATION_TIME	-	-
LAST_UPDATE_TIME	-	-
LAST_ACCESS_TIME	-	-
RECOVER_TIME	-	-
TRANSACTION_COUNTER	-	-
VERSION	-	-
ROW_FORMAT	-	-
TABLE_ROWS	-	-
AVG_ROW_LENGTH	-	-
DATA_LENGTH	-	-
MAX_DATA_LENGTH	-	-
INDEX_LENGTH	-	-
DATA_FREE	-	-
CREATE_TIME	-	-
UPDATE_TIME	-	-
CHECK_TIME	-	-
CHECKSUM	-	-
STATUS	-	-
EXTRA	-	-

Notes:

- The values in the FILE_ID column are auto-incremented.
- The FILE_NAME column specifies the name of the data file that is created by executing the `CREATE TABLESPACE` or `ALTER TABLESPACE` statement.
- The FILE_TYPE column specifies the type of the tablespace file.
- The TABLESPACE_NAME column specifies the name of the tablespace.
- The value in the TABLESPACE_CATALOG column is always NULL.
- The TABLE_NAME column specifies the name of the table.
- The value in the EXTENT_SIZE column is always 0.
- You cannot execute SHOW statements to view the data that is stored in the INFORMATION_SCHEMA.FILES table.

5.21.23. INFORMATION_SCHEMA.GLOBAL_STATUS table

This table provides the information about the status of global variables. To view the global variables, execute the `SHOW GLOBAL STATUS` statement.

Standard name	SHOW name	Remarks
VARIABLE_NAME	Variable_name	The name of the variable.
VARIABLE_VALUE |	Value	The value of the variable.

The data type of the VARIABLE_VALUE column is VARCHAR(1024).

5.21.24. INFORMATION_SCHEMA.GLOBAL_VARIABLES table

This table provides the information about global variables. To view the global variables, execute the `SHOW GLOBAL VARIABLES` statement.

Standard name	SHOW name	Remarks
VARIABLE_NAME	Variable_name	The name of the variable.
VARIABLE_VALUE |	Value	The value of the variable.

The data type of the VARIABLE_VALUE column is VARCHAR(1024).

5.21.25. INFORMATION_SCHEMA.PROCESSLIST table

This table provides the information about running threads.

Standard name	SHOW name	Remarks
ID	Id	The ID of the thread.
USER	User	The user.
HOST	Host	The name of the host.
DB	db	The name of the database.
COMMAND	Command	The command.
TIME	Time	The time.
STATE	State	The current status of the thread.
INFO	Info	The statement that is being executed on the thread.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
SHOW FULL PROCESSLIST
```

5.21.26. INFORMATION_SCHEMA.SESSION_STATUS table

This table provides the information about local variables. To view the local variables, execute the `SHOW SESSION STATUS` statement.

Standard name	SHOW name	Remarks
VARIABLE_NAME	Variable_name	The name of the variable.
VARIABLE_VALUE |	Value	The value of the variable.

The data type of the VARIABLE_VALUE column is VARCHAR(1024).

5.21.27. INFORMATION_SCHEMA.SESSION_VARIABLES table

This table provides the information about local variables. To view the local variables, execute the `SHOW SESSION VARIABLES` statement.

Standard name	SHOW name	Remarks
VARIABLE_NAME	Variable_name	The name of the variable.
VARIABLE_VALUE |	Value	The value of the variable.

The data type of the VARIABLE_VALUE column is VARCHAR(1024).

5.21.28. INFORMATION_SCHEMA.PROFILING table

This table provides the information about statement profiling. The profiling information in this table is the same as that in the output of the SHOW PROFILES or SHOW PROFILE statement.

Standard name	SHOW name	Remarks
QUERY_ID	Query_ID	-
SEQ	-	-
STATE	Status	-
DURATION	Duration	-
CPU_USER	CPU_user	-
CPU_SYSTEM	CPU_system	-
CONTEXT_VOLUNTARY	Context_voluntary	-
CONTEXT_INVOLUNTARY	Context_involuntary	-
BLOCK_OPS_IN	Block_ops_in	-
BLOCK_OPS_OUT	Block_ops_out	-
MESSAGES_SENT	Messages_sent	-
MESSAGES_RECEIVED	Messages_received	-
PAGE_FAULTS_MAJOR	Page_faults_major	-
PAGE_FAULTS_MINOR	Page_faults_minor	-
SWAPS	Swaps	-
SOURCE_FUNCTION	Source_function	-
SOURCE_FILE	Source_file	-
SOURCE_LINE	Source_line	-

5.21.29. INFORMATION_SCHEMA.PARAMETERS table

This table provides the information about the parameters for functions and methods.

Standard name	SHOW name	Remarks
SPECIFIC_CATALOG	-	def
SPECIFIC_SCHEMA	db	The name of the database that stores the function.
SPECIFIC_NAME	name	The name of the function.
ORDINAL_POSITION	-	The ordinal position of the parameter in the parameter list of the function. For example, the value can be 1, 2, or 3. The value 0 indicates the ordinal position of the row that describes the return value of the function.

Standard name	SHOW name	Remarks
PARAMETER_MODE	-	The type of the parameter. Valid values: • IN. • OUT. • INOUT. If the value of the parameter is returned by a function, the PARAMETER_MODE value is NULL.
PARAMETER_NAME	-	The name of the parameter. NULL is returned for a response parameter.
DATA_TYPE	-	The data type of the parameter.
CHARACTER_MAXIMUM_LENGTH	-	The maximum length of the parameter value. The length is measured in characters.
CHARACTER_OCTET_LENGTH	-	The maximum length of the value. If single-byte character sets are used, the CHARACTER_OCTET_LENGTH value is the same as the CHARACTER_MAXIMUM_LENGTH value. If multi-byte character sets are used, the values are different.
NUMERIC_PRECISION	-	The numeric precision.
NUMERIC_SCALE	-	The range of numeric values.
DATETIME_PRECISION	-	The time granularity.
CHARACTER_SET_NAME	-	The name of the character set.
COLLATION_NAME	-	The collation name.
DTD_IDENTIFIER	-	The identifier of the data type descriptor for the parameter.
ROUTINE_TYPE	type	{PROCEDURE|FUNCTION}

5.21.30. INFORMATION_SCHEMA.OPTIMIZER_TRACE table

Standard name	SHOW name	Remarks
QUERY	-	The query statement.
TRACE	-	The trace record.
MISSING_BYTES_BEYOND_MAX_MEM_SIZE	-	Reserved.
MISSING_PRIVILEGES	-	Reserved.

5.21.31. INFORMATION_SCHEMA.ENGINES table

Standard name	SHOW name	Remarks
ENGINE	-	-
SUPPORT	-	-
COMMENT	-	-
TRANSACTIONS	-	-
XA	-	-

Standard name	SHOW name	Remarks
SAVEPOINTS	-	-

5.21.32. INFORMATION_SCHEMA.PLUGINS table

Standard name	SHOW name	Remarks
NAME	-	-
OWNER|	-	-
DB_NAME	-	-
USED	-	-
TIMESTAMP	-	-
VERSION	-	-
SQL_TEXT	-	-
SIGNATURE	-	-
COMPATIBLE	-	-
ENABLED	-	-
FORMAT	-	-
OUTLINE_CONTENT	-	-
OUTLINE_TARGET	-	-

5.22. MySQL dictionary tables

To ensure compatibility with MySQL, ApsaraDB for OceanBase provides a MySQL database for each tenant. You can execute the `SHOW DATABASES` statement to view the MySQL database.

The MySQL database stores a set of dictionary tables that have the same schema as MySQL tables. describes these dictionary tables.

MySQL dictionary tables

Dictionary table	Description
db	Provides the information about databases and permissions.
user	Provides the information about users and permissions.
time_zone	Provides the information about time zones.
time_zone_name	Provides the information about time zones.
time_zone_name	
time_zone_transition_type	Provides the information about time zones.

5.23. Error codes

5.23.1. ApsaraDB for OceanBase error codes

```
DEFINE_ERROR(OB_SUCCESS, 0, 0, "00000", "Success");
////////////////////////////////////////////////////////////////
//error code for common -4000 ---- -4500
////////////////////////////////////////////////////////////////
DEFINE_ERROR(OB_ERROR, -4000, -1, "HY000", "Common error");
DEFINE_ERROR(OB_OBJ_TYPE_ERROR, -4001, -1, "HY004", "Object type error");
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT, -4002, ER_WRONG_ARGUMENTS, "HY000", "Invalid argument", "Incorrect arguments to %s"
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT, -4002, ER_WRONG_ARGUMENTS, "HY000", "Invalid argument", "Incorrect arguments to %s"
);
DEFINE_ERROR(OB_ARRAY_OUT_OF_RANGE, -4003, -1, "42000", "Array index out of range");
DEFINE_ERROR(OB_SERVER_LISTEN_ERROR, -4004, -1, "08S01", "Failed to listen to the port");
DEFINE_ERROR(OB_INIT_TWICE, -4005, -1, "HY000", "The object is initialized twice");
DEFINE_ERROR(OB_NOT_INIT, -4006, -1, "HY000", "The object is not initialized");
DEFINE_ERROR_EXT(OB_NOT_SUPPORTED, -4007, ER_NOT_SUPPORTED_YET, "0A000", "Not supported feature or function", "%s not sup
ported");
DEFINE_ERROR(OB_ITER_END, -4008, -1, "HY000", "End of iteration");
DEFINE_ERROR(OB_IO_ERROR, -4009, -1, "58030", "IO error");
DEFINE_ERROR(OB_ERROR_FUNC_VERSION, -4010, -1, "HY000", "Wrong RPC command version");
DEFINE_ERROR(OB_PACKET_NOT_SENT, -4011, -1, "HY000", "Can not send packet");
DEFINE_ERROR(OB_TIMEOUT, -4012, -1, "HY000", "Timeout");
DEFINE_ERROR(OB_ALLOCATE_MEMORY_FAILED, -4013, -1, "HY001", "No memory or reach tenant memory limit");
DEFINE_ERROR(OB_INNER_STAT_ERROR, -4014, -1, "HY000", "Inner state error");
DEFINE_ERROR(OB_ERR_SYS, -4015, -1, "HY000", "System error");
DEFINE_ERROR_EXT(OB_ERR_UNEXPECTED, -4016, -1, "HY000", "Ooooooooooooops", "%s");
DEFINE_ERROR(OB_ENTRY_EXIST, -4017, -1, "HY000", "Entry already exist");
DEFINE_ERROR(OB_ENTRY_NOT_EXIST, -4018, -1, "HY000", "Entry not exist");
DEFINE_ERROR(OB_SIZE_OVERFLOW, -4019, -1, "HY000", "Size overflow");
DEFINE_ERROR(OB_REF_NUM_NOT_ZERO, -4020, -1, "HY000", "Reference count is not zero");
DEFINE_ERROR(OB_CONFLICT_VALUE, -4021, -1, "HY000", "Conflict value");
DEFINE_ERROR(OB_ITEM_NOT_SETTED, -4022, -1, "HY000", "Item not set");
DEFINE_ERROR(OB_EAGAIN, -4023, -1, "HY000", "Try again");
DEFINE_ERROR(OB_BUF_NOT_ENOUGH, -4024, -1, "HY000", "Buffer not enough");
DEFINE_ERROR(OB_PARTIAL_FAILED, -4025, -1, "HY000", "Partial failed");
DEFINE_ERROR(OB_READ_NOTHING, -4026, -1, "02000", "Nothing to read");
DEFINE_ERROR(OB_FILE_NOT_EXIST, -4027, ER_FILE_NOT_FOUND, "HY000", "File not exist");
DEFINE_ERROR(OB_DISCONTINUOUS_LOG, -4028, -1, "HY000", "Log entry not continuous");
DEFINE_ERROR(OB_SCHEMA_ERROR, -4029, -1, "HY000", "Schema error");
DEFINE_ERROR(OB_TENANT_OUT_OF_MEM, -4030, -1, "HY000", "Over tenant memory limits");
DEFINE_ERROR(OB_UNKNOWN_OBJ, -4031, -1, "HY004", "Unknown object");
DEFINE_ERROR(OB_NO_MONITOR_DATA, -4032, -1, "02000", "No monitor data");
DEFINE_ERROR(OB_SERIALIZE_ERROR, -4033, -1, "HY000", "Serialize error");
DEFINE_ERROR(OB_DESERIALIZE_ERROR, -4034, -1, "HY000", "Deserialize error");
DEFINE_ERROR(OB_AIO_TIMEOUT, -4035, -1, "HY000", "Asynchronous IO error");
DEFINE_ERROR(OB_NEED_RETRY, -4036, -1, "HY000", "Need retry");
DEFINE_ERROR(OB_TOO_MANY_SSTABLE, -4037, -1, "HY000", "Too many sstable");
DEFINE_ERROR(OB_NOT_MASTER, -4038, -1, "HY000", "The observer or zone is not the master");
DEFINE_ERROR(OB_DECRYPT_FAILED, -4041, -1, "HY000", "Decrypt error");
DEFINE_ERROR(OB_USER_NOT_EXIST, -4042, ER_PASSWORD_NO_MATCH, "42000", "Can not find any matching row in the user table");
DEFINE_ERROR_EXT(OB_PASSWORD_WRONG, -4043, ER_ACCESS_DENIED_ERROR, "42000", "Access denied for user", "Access denied for
user '%.*s'@'%.*s' (using password: %s)");
DEFINE_ERROR(OB_SKEY_VERSION_WRONG, -4044, -1, "HY000", "Wrong skey version");
DEFINE_ERROR(OB_NOT_REGISTERED, -4048, -1, "HY000", "Not registered");
DEFINE_ERROR(OB_WAITQUEUE_TIMEOUT, -4049, 4012, "HY000", "Task timeout and not executed");
DEFINE_ERROR(OB_NOT_THE_OBJECT, -4050, -1, "HY000", "Not the object");
DEFINE_ERROR(OB_ALREADY_REGISTERED, -4051, -1, "HY000", "Already registered");
DEFINE_ERROR(OB_LAST_LOG_RUINNED, -4052, -1, "HY000", "Corrupted log entry");
DEFINE_ERROR(OB_NO_CS_SELECTED, -4053, -1, "HY000", "No ChunkServer selected");
DEFINE_ERROR(OB_NO_TABLETS_CREATED, -4054, -1, "HY000", "No tablets created");
DEFINE_ERROR(OB_INVALID_ERROR, -4055, -1, "HY000", "Invalid entry");
DEFINE_ERROR(OB_DECIMAL_OVERFLOW_WARN, -4057, -1, "HY000", "Decimal overflow warning");
DEFINE_ERROR(OB_DECIMAL_UNLEGAL_ERROR, -4058, -1, "HY000", "Decimal overflow error");
DEFINE_ERROR(OB_OBJ_DIVIDE_ERROR, -4060, -1, "HY000", "Divide error");
DEFINE_ERROR(OB_NOT_A_DECIMAL, -4061, -1, "HY000", "Not a decimal");
DEFINE_ERROR(OB_DECIMAL_PRECISION_NOT_EQUAL, -4062, -1, "HY104", "Decimal precision error");
DEFINE_ERROR(OB_EMPTY_RANGE, -4063, -1, "HY000", "Empty range");
DEFINE_ERROR(OB_SESSION_KILLED, -4064, -1, "HY000", "Session killed");
DEFINE_ERROR(OB_LOG_NOT_SYNC, -4065, -1, "HY000", "Log not sync");
DEFINE_ERROR(OB_DIR_NOT_EXIST, -4066, ER_CANT_READ_DIR, "HY000", "Directory not exist");
DEFINE_ERROR(OB_SESSION_NOT_FOUND, -4067, 4012, "HY000", "RPC session not found");
DEFINE_ERROR(OB_INVALID_LOG, -4068, -1, "HY000", "Invalid log");
DEFINE_ERROR(OB_INVALID_DATA, -4070, -1, "HY000", "Invalid data");
DEFINE_ERROR(OB_ALREADY_DONE, -4071, -1, "HY000", "Already done");
DEFINE_ERROR(OB_CANCELED, -4072, -1, "HY000", "Operation canceled");
DEFINE_ERROR(OB_LOG_SRC_CHANGED, -4073, -1, "HY000", "Log source changed");
DEFINE_ERROR(OB_LOG_NOT_ALIGN, -4074, -1, "HY000", "Log not aligned");
DEFINE_ERROR(OB_LOG_MISSING, -4075, -1, "HY000", "Log entry missed");
DEFINE_ERROR(OB_NEED_WAIT, -4076, -1, "HY000", "Need wait");
DEFINE_ERROR(OB_NOT_IMPLEMENT, -4077, -1, "0A000", "Not implemented feature");
DEFINE_ERROR(OB_DIVISION_BY_ZERO, -4078, ER_DIVISION_BY_ZERO, "42000", "Divided by zero");
DEFINE_ERROR(OB_DIVISION_BY_ZERO, -4078, ER_DIVISION_BY_ZERO, "42000", "Divided by zero");
DEFINE_ERROR(OB_EXCEED_MEM_LIMIT, -4080, -1, "HY013", "exceed memory limit");
DEFINE_ERROR(OB_RESULT_UNKNOWN, -4081, -1, "HY000", "Unknown result");
DEFINE_ERROR(OB_NO_RESULT, -4084, -1, "02000", "No result");
DEFINE_ERROR(OB_QUEUE_OVERFLOW, -4085, -1, "HY000", "Queue overflow");
DEFINE_ERROR(OB_TERM_LAGGED, -4097, -1, "HY000", "Term lagged");
DEFINE_ERROR(OB_TERM_NOT_MATCH, -4098, -1, "HY000", "Term not match");
DEFINE_ERROR(OB_START_LOG_CURSOR_INVALID, -4099, -1, "HY000", "Invalid log cursor");
DEFINE_ERROR(OB_LOCK_NOT_MATCH, -4100, -1, "HY000", "Lock not match");
DEFINE_ERROR(OB_DEAD_LOCK, -4101, ER_LOCK_DEADLOCK, "HY000", "Deadlock");
DEFINE_ERROR(OB_PARTIAL_LOG, -4102, -1, "HY000", "Incomplete log entry");
DEFINE_ERROR(OB_CHECKSUM_ERROR, -4103, -1, "42000", "Data checksum error");
DEFINE_ERROR(OB_INIT_FAIL, -4104, -1, "HY000", "Initialize error");
DEFINE_ERROR(OB_NOT_ENOUGH_STORE, -4106, -1, "HY000", "not enough commitlog store");
DEFINE_ERROR(OB_BLOCK_SWITCHED, -4107, -1, "HY000", "block switched when fill commitlog");
DEFINE_ERROR(OB_STATE_NOT_MATCH, -4109, -1, "HY000", "Server state or role not the same as expected");
DEFINE_ERROR(OB_READ_ZERO_LOG, -4110, -1, "HY000", "Read zero log");
DEFINE_ERROR(OB_BLOCK_NEED_FREEZE, -4111, -1, "HY000", "block need freeze");
DEFINE_ERROR(OB_BLOCK_FROZEN, -4112, -1, "HY000", "block frozen");
DEFINE_ERROR(OB_IN_FATAL_STATE, -4113, -1, "HY000", "In FATAL state");
DEFINE_ERROR(OB_IN_STOP_STATE, -4114, -1, "08S01", "In STOP state");
DEFINE_ERROR(OB_UPS_MASTER_EXISTS, -4115, -1, "HY000", "Master UpdateServer already exists");
DEFINE_ERROR(OB_LOG_NOT_CLEAR, -4116, -1, "42000", "Log not clear");
DEFINE_ERROR(OB_FILE_ALREADY_EXIST, -4117, ER_FILE_EXISTS_ERROR, "58000", "File already exist");
DEFINE_ERROR(OB_UNKNOWN_PACKET, -4118, ER_UNKNOWN_COM_ERROR, "HY001", "Unknown packet");
DEFINE_ERROR(OB_RPC_PACKET_TOO_LONG, -4119, -1, "08000", "RPC packet to send too long");
DEFINE_ERROR(OB_LOG_TOO_LARGE, -4120, -1, "HY000", "Log too large");
DEFINE_ERROR(OB_RPC_SEND_ERROR, -4121, -1, "08000", "RPC send error");
DEFINE_ERROR(OB_RPC_POST_ERROR, -4122, -1, "08000", "RPC post error");
DEFINE_ERROR(OB_LIBEASY_ERROR, -4123, -1, "08000", "Libeasy error");
DEFINE_ERROR(OB_CONNECT_ERROR, -4124, -1, "HY000", "Connect error");
DEFINE_ERROR(OB_NOT_FREE, -4125, -1, "HY000", "Not free");
DEFINE_ERROR(OB_INIT_SQL_CONTEXT_ERROR, -4126, -1, "HY000", "Init SQL context error");
DEFINE_ERROR(OB_SKIP_INVALID_ROW, -4127, -1, "42000", "Skip invalid row");
DEFINE_ERROR(OB_RPC_PACKET_INVALID, -4128, -1, "HY000", "RPC packet is invalid");
DEFINE_ERROR(OB_NO_TABLET, -4133, -1, "HY000", "No tablets");
DEFINE_ERROR(OB_SNAPSHOT_DISCARDED, -4138, -1, "HY000", "Request to read too old versioned data");
DEFINE_ERROR(OB_DATA_NOT_UPTODATE, -4139, -1, "HY000", "State is stale");
DEFINE_ERROR(OB_ROW_MODIFIED, -4142, -1, "HY000", "Row modified");
DEFINE_ERROR(OB_VERSION_NOT_MATCH, -4143, -1, "42000", "Version not match");
DEFINE_ERROR(OB_BAD_ADDRESS, -4144, -1, "42000", "Bad address");
DEFINE_ERROR(OB_ENQUEUE_FAILED, -4146, -1, "HY000", "Enqueue error");
DEFINE_ERROR(OB_INVALID_CONFIG, -4147, -1, "HY000", "Invalid config");
DEFINE_ERROR(OB_STMT_EXPIRED, -4149, -1, "HY000", "Expired statement");
DEFINE_ERROR(OB_ERR_MIN_VALUE, -4150, -1, "42000", "Min value");
DEFINE_ERROR(OB_ERR_MAX_VALUE, -4151, -1, "42000", "Max value");
DEFINE_ERROR_EXT(OB_ERR_NULL_VALUE, -4152, -1, "42000", "Null value", "%s");
DEFINE_ERROR(OB_RESOURCE_OUT, -4153, ER_OUT_OF_RESOURCES, "53000", "Out of resource");
DEFINE_ERROR(OB_ERR_SQL_CLIENT, -4154, -1, "HY000", "Internal SQL client error");
DEFINE_ERROR(OB_META_TABLE_WITHOUT_USE_TABLE, -4155, -1, "HY000", "Meta table without use table");
DEFINE_ERROR(OB_DISCARD_PACKET, -4156, -1, "HY000", "Discard packet");
DEFINE_ERROR_EXT(OB_OPERATE_OVERFLOW, -4157, ER_DATA_OUT_OF_RANGE, "22003", "value is out of range", "%s value is out of
range in '%s'");
DEFINE_ERROR_EXT(OB_INVALID_DATE_FORMAT, -4158, ER_TRUNCATED_WRONG_VALUE, "22007", "Incorrect value", "%s=%d must between
%d and %d");
DEFINE_ERROR(OB_POOL_REGISTERED_FAILED, -4159, -1, "HY000", "register pool failed");
DEFINE_ERROR(OB_POOL_UNREGISTERED_FAILED, -4160, -1, "HY000", "unregister pool failed");
DEFINE_ERROR(OB_INVALID_ARGUMENT_NUM, -4161, -1, "42000", "Invalid argument num");
DEFINE_ERROR(OB_LEASE_NOT_ENOUGH, -4162, -1, "HY000", "reserved lease not enough");
DEFINE_ERROR(OB_LEASE_NOT_MATCH, -4163, -1, "HY000", "ups lease not match with rs");
DEFINE_ERROR(OB_UPS_SWITCH_NOT_HAPPEN, -4164, -1, "HY000", "ups switch not happen");
DEFINE_ERROR(OB_EMPTY_RESULT, -4165, -1, "HY000", "Empty result");
DEFINE_ERROR(OB_CACHE_NOT_HIT, -4166, -1, "HY000", "Cache not hit");
DEFINE_ERROR(OB_NESTED_LOOP_NOT_SUPPORT, -4167, -1, "HY000", "Nested loop not support");
DEFINE_ERROR(OB_LOG_INVALID_MOD_ID, -4168, -1, "HY000", "Invalid log module id");
DEFINE_ERROR_EXT(OB_LOG_MODULE_UNKNOWN, -4169, -1, "HY000", "Unknown module name", "Unknown module name. Invalid Setting:
'%.*s'. Syntax:parMod.subMod:level, parMod.subMod:level");
DEFINE_ERROR_EXT(OB_LOG_LEVEL_INVALID, -4170, -1, "HY000", "Invalid level", "Invalid level. Invalid setting:'%.*s'. Synta
x:parMod.subMod:level, parMod.subMod:level");
DEFINE_ERROR_EXT(OB_LOG_PARSER_SYNTAX_ERR, -4171, -1, "HY000", "Syntax to set log_level error", "Syntax to set log_level
error. Invalid setting:'%.*s'. Syntax:parMod.subMod:level, parMod.subMod:level");
DEFINE_ERROR(OB_INDEX_OUT_OF_RANGE, -4172, -1, "HY000", "Index out of range");
DEFINE_ERROR(OB_INDEX_OUT_OF_RANGE, -4172, -1, "HY000", "Index out of range");
DEFINE_ERROR(OB_INT_UNDERFLOW, -4173, -1, "HY000", "Int underflow");
DEFINE_ERROR_EXT(OB_UNKNOWN_CONNECTION, -4174, ER_NO_SUCH_THREAD, "HY000", "Unknown thread id", "Unknown thread id: %lu")
;
DEFINE_ERROR(OB_ERROR_OUT_OF_RANGE, -4175, -1, "42000", "Out of range");
DEFINE_ERROR(OB_CACHE_SHRINK_FAILED, -4176, -1, "HY001", "shrink cache failed, no available cache");
DEFINE_ERROR(OB_OLD_SCHEMA_VERSION, -4177, -1, "42000", "Schema version too old");
DEFINE_ERROR(OB_RELEASE_SCHEMA_ERROR, -4178, -1, "HY000", "Release schema error");
DEFINE_ERROR_EXT(OB_OP_NOT_ALLOW, -4179, -1, "HY000", "Operation not allowed now", "%s not allowed");
DEFINE_ERROR(OB_NO_EMPTY_ENTRY, -4180, -1, "HY000", "No empty entry");
DEFINE_ERROR(OB_ERR_ALREADY_EXISTS, -4181, -1, "42S01", "Already exist");
DEFINE_ERROR(OB_SEARCH_NOT_FOUND, -4182, -1, "HY000", "Value not found");
DEFINE_ERROR(OB_BEYOND_THE_RANGE, -4183, -1, "HY000", "Key out of range");
DEFINE_ERROR(OB_CS_OUTOF_DISK_SPACE, -4184, -1, "53100", "ChunkServer out of disk space");
DEFINE_ERROR(OB_COLUMN_GROUP_NOT_FOUND, -4185, -1, "HY000", "Column group not found");
DEFINE_ERROR(OB_CS_COMPRESS_LIB_ERROR, -4186, -1, "HY000", "ChunkServer failed to get compress library");
DEFINE_ERROR(OB_ITEM_NOT_MATCH, -4187, -1, "HY000", "Item not match");
DEFINE_ERROR(OB_SCHEDULER_TASK_CNT_MISMATCH, -4188, -1, "HY000", "Running task cnt and unfinished task cnt not consistent
");
DEFINE_ERROR(OB_HASH_EXIST, -4200, -1, "HY000", "hash map/set entry exist");
DEFINE_ERROR(OB_HASH_NOT_EXIST, -4201, -1, "HY000", "hash map/set entry not exist");
DEFINE_ERROR(OB_HASH_GET_TIMEOUT, -4204, -1, "HY000", "hash map/set get timeout");
DEFINE_ERROR(OB_HASH_PLACEMENT_RETRY, -4205, -1, "HY000", "hash map/set retry");
DEFINE_ERROR(OB_HASH_FULL, -4206, -1, "HY000", "hash map/set full");
DEFINE_ERROR(OB_PACKET_PROCESSED, -4207, -1, "HY000", "packet processed");
DEFINE_ERROR(OB_WAIT_NEXT_TIMEOUT, -4208, -1, "HY000", "wait next packet timeout");
DEFINE_ERROR(OB_LEADER_NOT_EXIST, -4209, -1, "HY000", "partition has not leader");
DEFINE_ERROR(OB_PREPARE_MAJOR_FREEZE_FAILED, -4210, -1, "HY000", "prepare major freeze failed");
DEFINE_ERROR(OB_COMMIT_MAJOR_FREEZE_FAILED, -4211, -1, "HY000", "commit major freeze failed");
DEFINE_ERROR(OB_ABORT_MAJOR_FREEZE_FAILED, -4212, -1, "HY000", "abort major freeze failed");
DEFINE_ERROR(OB_MAJOR_FREEZE_NOT_FINISHED, -4213, -1, "HY000", "last major freeze not finish");
DEFINE_ERROR(OB_PARTITION_NOT_LEADER, -4214, -1, "HY000", "partition is not leader partition");
DEFINE_ERROR(OB_WAIT_MAJOR_FREEZE_RESPONSE_TIMEOUT, -4215, -1, "HY000", "wait major freeze response timeout");
DEFINE_ERROR(OB_CURL_ERROR, -4216, -1, "HY000", "curl error");
DEFINE_ERROR_EXT(OB_MAJOR_FREEZE_NOT_ALLOW, -4217, -1, "HY000", "Major freeze not allowed now", "%s");
DEFINE_ERROR(OB_PREPARE_FREEZE_FAILED, -4218, -1, "HY000", "prepare freeze failed");
DEFINE_ERROR_EXT(OB_INVALID_DATE_VALUE, -4219, ER_TRUNCATED_WRONG_VALUE, "22007", "Incorrect value", "Incorrect datetime
value: '%s' for column '%s' at row %d");
DEFINE_ERROR(OB_INACTIVE_SQL_CLIENT, -4220, -1, "HY000", "Inactive sql client, only read allowed");
DEFINE_ERROR(OB_INACTIVE_RPC_PROXY, -4221, -1, "HY000", "Inactive rpc proxy, can not send RPC request");
DEFINE_ERROR(OB_INTERVAL_WITH_MONTH, -4222, -1, "42000", "Interval with year or month can not be converted to microsecond
s");
DEFINE_ERROR(OB_TOO_MANY_DATETIME_PARTS, -4223, -1, "42000", "Interval has too many datetime parts");
DEFINE_ERROR_EXT(OB_DATA_OUT_OF_RANGE, -4224, ER_WARN_DATA_OUT_OF_RANGE, "22003", "Out of range value for column", "Out o
f range value for column '%.*s' at row %lld");
DEFINE_ERROR(OB_PARTITION_NOT_EXIST, -4225, -1, "HY000", "Partition entry not exists");
DEFINE_ERROR_EXT(OB_ERR_TRUNCATED_WRONG_VALUE_FOR_FIELD, -4226, ER_TRUNCATED_WRONG_VALUE_FOR_FIELD, "HY000", "Incorrect i
nteger value", "Incorrect integer value: '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_NO_DEFAULT_FOR_FIELD, -4227, ER_NO_DEFAULT_FOR_FIELD, "HY000", "Field doesn't have a default valu
e", "Field \'%s\' doesn't have a default value");
DEFINE_ERROR_EXT(OB_ERR_FIELD_SPECIFIED_TWICE, -4228, ER_FIELD_SPECIFIED_TWICE, "42000", "Column specified twice", "Colum
n \'%s\' specified twice");
DEFINE_ERROR_EXT(OB_ERR_TOO_LONG_TABLE_COMMENT, -4229, ER_TOO_LONG_TABLE_COMMENT, "HY000", "Comment for table is too long
", "Comment for table is too long (max = %ld)");
DEFINE_ERROR_EXT(OB_ERR_TOO_LONG_FIELD_COMMENT, -4230, ER_TOO_LONG_FIELD_COMMENT, "HY000", "Comment for field is too long
", "Comment for field is too long (max = %ld)");
DEFINE_ERROR_EXT(OB_ERR_TOO_LONG_INDEX_COMMENT, -4231, ER_TOO_LONG_INDEX_COMMENT, "HY000", "Comment for index is too long
", "Comment for index is too long (max = %ld)");
DEFINE_ERROR(OB_NOT_FOLLOWER, -4232, -1, "HY000", "The observer or zone is not a follower");
DEFINE_ERROR(OB_ERR_OUT_OF_LOWER_BOUND, -4233, -1, "HY000", "smaller than container lower bound");
DEFINE_ERROR(OB_ERR_OUT_OF_UPPER_BOUND, -4234, -1, "HY000", "bigger than container upper bound");
DEFINE_ERROR_EXT(OB_BAD_NULL_ERROR, -4235, ER_BAD_NULL_ERROR, "23000", "Column cannot be null", "Column '%.*s' cannot be
null");
DEFINE_ERROR(OB_OBCONFIG_RETURN_ERROR, -4236, -1, "HY000", "ObConfig return error code");
DEFINE_ERROR(OB_OBCONFIG_APPNAME_MISMATCH, -4237, -1, "HY000", "Appname mismatch with obconfig result");
DEFINE_ERROR(OB_ERR_VIEW_SELECT_DERIVED, -4238, ER_VIEW_SELECT_DERIVED, "HY000", "View's SELECT contains a subquery in th
e FROM clause");
DEFINE_ERROR(OB_CANT_MJ_PATH, -4239, -1, "HY000", "Can not use merge-join to join the tables without join conditions");
DEFINE_ERROR(OB_ERR_NO_JOIN_ORDER_GENERATED, -4240, -1, "HY000", "No join order generated");
DEFINE_ERROR(OB_ERR_NO_PATH_GENERATED, -4241, -1, "HY000", "No join path generated");
DEFINE_ERROR(OB_ERR_WAIT_REMOTE_SCHEMA_REFRESH, -4242, -1, "HY000", "Schema error");
DEFINE_ERROR(OB_FILE_NOT_OPENED, -4243, -1, "HY000", "file not opened");
DEFINE_ERROR(OB_TIMER_TASK_HAS_SCHEDULED, -4244, -1, "HY000", "Timer task has been scheduled");
DEFINE_ERROR(OB_TIMER_TASK_HAS_NOT_SCHEDULED, -4245, -1, "HY000", "Timer task has not been scheduled");
DEFINE_ERROR(OB_PARSE_DEBUG_SYNC_ERROR, -4246, -1, "HY000", "parse debug sync string error");
DEFINE_ERROR(OB_UNKNOWN_DEBUG_SYNC_POINT, -4247, -1, "HY000", "unknown debug sync point");
DEFINE_ERROR(OB_ERR_INTERRUPTED, -4248, -1, "HY000", "task is interrupted while running");
DEFINE_ERROR(OB_ERR_DATA_TRUNCATED, -4249, WARN_DATA_TRUNCATED, "01000", "Data truncated for argument");
// used by modules in partition service only, and not returned to client
DEFINE_ERROR(OB_NOT_RUNNING, -4250, -1, "HY000", "module is not running");
DEFINE_ERROR(OB_INVALID_PARTITION, -4251, -1, "HY000", "partition not valid");
DEFINE_ERROR(OB_ERR_TIMEOUT_TRUNCATED, -4252, WARN_DATA_TRUNCATED, "01000", "Timeout value truncated to 102 years");
DEFINE_ERROR_EXT(OB_ERR_TOO_LONG_TENANT_COMMENT, -4253, -1, "HY000", "Comment for tenant is too long", "Comment for tenan
t is too long (max = %ld)");
DEFINE_ERROR(OB_ERR_NET_PACKET_TOO_LARGE, -4254, ER_NET_PACKET_TOO_LARGE, "08S01", "Got a packet bigger than \'max_allowe
d_packet\' bytes");
DEFINE_ERROR(OB_TRACE_DESC_NOT_EXIST, -4255, -1, "HY000", "trace log title or key not exist describle");
DEFINE_ERROR_EXT(OB_ERR_NO_DEFAULT, -4256, ER_NO_DEFAULT, "42000", "Variable doesn't have a default value", "Variable '%.
*s' doesn't have a default value");
DEFINE_ERROR(OB_ERR_COMPRESS_DECOMPRESS_DATA, -4257, -1, "HY000", "compress data or decompress data failed");
DEFINE_ERROR_EXT(OB_ERR_INCORRECT_STRING_VALUE, -4258, ER_TRUNCATED_WRONG_VALUE_FOR_FIELD, "HY000", "Incorrect string val
ue", "Incorrect string value for column '%.*s' at row %lld");
DEFINE_ERROR_EXT(OB_ERR_DISTRIBUTED_NOT_SUPPORTED, -4259, ER_NOT_SUPPORTED_YET, "0A000", "Not supported feature or functi
on", "%s not supported");
DEFINE_ERROR(OB_IS_CHANGING_LEADER, -4260, -1, "HY000", "the partition is changing leader");
DEFINE_ERROR(OB_DATETIME_FUNCTION_OVERFLOW, -4261, ER_DATETIME_FUNCTION_OVERFLOW, "22008", "Datetime overflow");
DEFINE_ERROR_EXT(OB_ERR_DOUBLE_TRUNCATED, -4262, ER_TRUNCATED_WRONG_VALUE, "01000", "Truncated incorrect DOUBLE value", "
Truncated incorrect DOUBLE value: '%.*s'");
DEFINE_ERROR_EXT(OB_MINOR_FREEZE_NOT_ALLOW, -4263, -1, "HY000", "Minor freeze not allowed now", "%s");
DEFINE_ERROR(OB_LOG_OUTOF_DISK_SPACE, -4264, -1, "HY000", "Log out of disk space");
DEFINE_ERROR(OB_RPC_CONNECT_ERROR, -4265, -1, "HY000", "Rpc connect error");
DEFINE_ERROR(OB_MINOR_MERGE_NOT_ALLOW, -4266, -1, "HY000", "minor merge not allow");
DEFINE_ERROR(OB_CACHE_INVALID, -4267, -1, "HY000", "Cache invalid");
DEFINE_ERROR(OB_REACH_SERVER_DATA_COPY_IN_CONCURRENCY_LIMIT, -4268, -1, "HY000", "reach server data copy in concurrency")
;
DEFINE_ERROR(OB_WORKING_PARTITION_EXIST, -4269, -1, "HY000", "Working partition entry already exists");
DEFINE_ERROR(OB_WORKING_PARTITION_NOT_EXIST, -4270, -1, "HY000", "Working partition entry does not exists");
DEFINE_ERROR(OB_LIBEASY_REACH_MEM_LIMIT, -4271, -1, "HY000", "LIBEASY reach memory limit");
DEFINE_ERROR_EXT(OB_MISS_ARGUMENT, -4272, ER_WRONG_ARGUMENTS, "HY000", "Miss argument", "Miss argument for %s");
////////////////////////////////////////////////////////////////
//error code for root server & server management -4500 ---- -5000
////////////////////////////////////////////////////////////////
DEFINE_ERROR(OB_IMPORT_NOT_IN_SERVER, -4505, -1, "HY000", "Import not in service");
DEFINE_ERROR(OB_CONVERT_ERROR, -4507, -1, "42000", "Convert error");
DEFINE_ERROR(OB_BYPASS_TIMEOUT, -4510, -1, "HY000", "Bypass timeout");
DEFINE_ERROR(OB_RS_STATE_NOT_ALLOW, -4512, -1, "HY000", "RootServer state error");
DEFINE_ERROR(OB_NO_REPLICA_VALID, -4515, -1, "HY000", "No replica is valid");
DEFINE_ERROR(OB_NO_NEED_UPDATE, -4517, -1, "HY000", "No need to update");
DEFINE_ERROR(OB_CACHE_TIMEOUT, -4518, -1, "HY000", "Cache timeout");
DEFINE_ERROR(OB_ITER_STOP, -4519, -1, "HY000", "Iteration was stopped");
DEFINE_ERROR(OB_ZONE_ALREADY_MASTER, -4523, -1, "HY000", "The zone is the master already");
DEFINE_ERROR(OB_IP_PORT_IS_NOT_SLAVE_ZONE, -4524, -1, "HY000", "Not slave zone");
DEFINE_ERROR(OB_ZONE_IS_NOT_SLAVE, -4525, -1, "HY000", "Not slave zone");
DEFINE_ERROR(OB_ZONE_IS_NOT_MASTER, -4526, -1, "HY000", "Not master zone");
DEFINE_ERROR(OB_CONFIG_NOT_SYNC, -4527, -1, "F0000", "Configuration not sync");
DEFINE_ERROR(OB_IP_PORT_IS_NOT_ZONE, -4528, -1, "42000", "Not a zone address");
DEFINE_ERROR(OB_MASTER_ZONE_NOT_EXIST, -4529, -1, "HY000", "Master zone not exist");
DEFINE_ERROR_EXT(OB_ZONE_INFO_NOT_EXIST, -4530, -1, "HY000", "Zone info not exist", "Zone info \'%s\' not exist");
DEFINE_ERROR(OB_GET_ZONE_MASTER_UPS_FAILED, -4531, -1, "HY000", "Failed to get master UpdateServer");
DEFINE_ERROR(OB_MULTIPLE_MASTER_ZONES_EXIST, -4532, -1, "HY000", "Multiple master zones");
DEFINE_ERROR(OB_INDEXING_ZONE_INVALID, -4533, -1, "HY000", "indexing zone is not exist anymore or not active");
DEFINE_ERROR(OB_ROOT_TABLE_RANGE_NOT_EXIST, -4537, -1, "HY000", "Tablet range not exist");
DEFINE_ERROR(OB_ROOT_MIGRATE_CONCURRENCY_FULL, -4538, -1, "HY000", "Migrate concurrency full");
DEFINE_ERROR(OB_ROOT_MIGRATE_INFO_NOT_FOUND, -4539, -1, "HY000", "Migrate info not found");
DEFINE_ERROR(OB_NOT_DATA_LOAD_TABLE, -4540, -1, "HY000", "No data to load");
DEFINE_ERROR(OB_DATA_LOAD_TABLE_DUPLICATED, -4541, -1, "HY000", "Duplicated table data to load");
DEFINE_ERROR(OB_ROOT_TABLE_ID_EXIST, -4542, -1, "HY000", "Table ID exist");
DEFINE_ERROR(OB_INDEX_TIMEOUT, -4543, -1, "HY000", "Building index timeout");
DEFINE_ERROR(OB_ROOT_NOT_INTEGRATED, -4544, -1, "42000", "Root not integrated");
DEFINE_ERROR(OB_INDEX_INELIGIBLE, -4545, -1, "HY000", "index data not unique");
DEFINE_ERROR(OB_REBALANCE_EXEC_TIMEOUT, -4546, -1, "HY000", "execute replication or migration task timeout");
DEFINE_ERROR(OB_MERGE_NOT_STARTED, -4547, -1, "HY000", "global merge not started");
DEFINE_ERROR(OB_MERGE_ALREADY_STARTED, -4548, -1, "HY000", "merge already started");
DEFINE_ERROR(OB_ROOTSERVICE_EXIST, -4549, -1, "HY000", "rootservice already exist");
DEFINE_ERROR(OB_RS_SHUTDOWN, -4550, -1, "HY000", "rootservice is shutdown");
DEFINE_ERROR(OB_SERVER_MIGRATE_IN_DENIED, -4551, -1, "HY000", "server migrate in denied");
DEFINE_ERROR(OB_REBALANCE_TASK_CANT_EXEC, -4552, -1, "HY000", "rebalance task can not executing now");
DEFINE_ERROR(OB_PARTITION_CNT_REACH_ROOTSERVER_LIMIT, -4553, -1, "HY000", "rootserver can not hold more partition");
DEFINE_ERROR(OB_DATA_SOURCE_NOT_EXIST, -4600, -1, "HY000", "Data source not exist");
DEFINE_ERROR(OB_DATA_SOURCE_TABLE_NOT_EXIST, -4601, -1, "HY000", "Data source table not exist");
DEFINE_ERROR(OB_DATA_SOURCE_RANGE_NOT_EXIST, -4602, -1, "HY000", "Data source range not exist");
DEFINE_ERROR(OB_DATA_SOURCE_DATA_NOT_EXIST, -4603, -1, "HY000", "Data source data not exist");
DEFINE_ERROR(OB_DATA_SOURCE_SYS_ERROR, -4604, -1, "HY000", "Data source sys error");
DEFINE_ERROR(OB_DATA_SOURCE_TIMEOUT, -4605, -1, "HY000", "Data source timeout");
DEFINE_ERROR(OB_DATA_SOURCE_CONCURRENCY_FULL, -4606, -1, "53000", "Data source concurrency full");
DEFINE_ERROR(OB_DATA_SOURCE_WRONG_URI_FORMAT, -4607, -1, "42000", "Data source wrong URI format");
DEFINE_ERROR(OB_SSTABLE_VERSION_UNEQUAL, -4608, -1, "42000", "SSTable version not equal");
DEFINE_ERROR(OB_UPS_RENEW_LEASE_NOT_ALLOWED, -4609, -1, "HY000", "ups should not renew its lease");
DEFINE_ERROR(OB_UPS_COUNT_OVER_LIMIT, -4610, -1, "HY000", "ups count over limit");
DEFINE_ERROR(OB_NO_UPS_MAJORITY, -4611, -1, "HY000", "ups not form a majority");
DEFINE_ERROR(OB_INDEX_COUNT_REACH_THE_LIMIT, -4613, -1, "HY000", "created index tables count has reach the limit:128");
DEFINE_ERROR(OB_TASK_EXPIRED, -4614, -1, "HY000", "task expired");
DEFINE_ERROR(OB_TABLEGROUP_NOT_EMPTY, -4615, -1, "HY000", "tablegroup is not empty");
DEFINE_ERROR(OB_INVALID_SERVER_STATUS, -4620, -1, "HY000", "server status is not valid");
DEFINE_ERROR(OB_WAIT_ELEC_LEADER_TIMEOUT, -4621, -1, "HY000", "wait elect partition leader timeout");
DEFINE_ERROR(OB_WAIT_ALL_RS_ONLINE_TIMEOUT, -4622, -1, "HY000", "wait all rs online timeout");
DEFINE_ERROR(OB_ALL_REPLICAS_ON_MERGE_ZONE, -4623, -1, "HY000", "all replicas of partition group are on zones to merge");
DEFINE_ERROR(OB_MACHINE_RESOURCE_NOT_ENOUGH, -4624, -1, "HY000", "machine resource is not enough to hold a new unit");
DEFINE_ERROR(OB_NOT_SERVER_CAN_HOLD_SOFTLY, -4625, -1, "HY000", "not server can hole the unit and not over soft limit");
DEFINE_ERROR_EXT(OB_RESOURCE_POOL_ALREADY_GRANTED, -4626, -1, "HY000", "resource pool has already been granted to a tenan
t", "resource pool \'%s\' has already been granted to a tenant");
DEFINE_ERROR(OB_SERVER_ALREADY_DELETED, -4628, -1, "HY000", "server has already been deleted");
DEFINE_ERROR(OB_SERVER_NOT_DELETING, -4629, -1, "HY000", "server is not in deleting status");
DEFINE_ERROR(OB_SERVER_NOT_IN_WHITE_LIST, -4630, -1, "HY000", "server not in server white list");
DEFINE_ERROR(OB_SERVER_ZONE_NOT_MATCH, -4631, -1, "HY000", "server zone not match");
DEFINE_ERROR(OB_OVER_ZONE_NUM_LIMIT, -4632, -1, "HY000", "zone num has reach max zone num");
DEFINE_ERROR(OB_ZONE_STATUS_NOT_MATCH, -4633, -1, "HY000", "zone status not match");
DEFINE_ERROR_EXT(OB_RESOURCE_UNIT_IS_REFERENCED, -4634, -1, "HY000", "resource unit is referenced by resource pool", "res
ource unit \'%s\' is referenced by some resource pool");
DEFINE_ERROR(OB_DIFFERENT_PRIMARY_ZONE, -4636, -1, "HY000", "table schema primary zone different with other table in samp
e tablegroup");
DEFINE_ERROR(OB_SERVER_NOT_ACTIVE, -4637, -1, "HY000", "server is not active");
DEFINE_ERROR(OB_RS_NOT_MASTER, -4638, -1, "HY000", "The RootServer is not the master");
DEFINE_ERROR(OB_CANDIDATE_LIST_ERROR, -4639, -1, "HY000", "The candidate list is invalid");
DEFINE_ERROR(OB_PARTITION_ZONE_DUPLICATED, -4640, -1, "HY000", "The chosen partition servers belong to same zone.") ;
DEFINE_ERROR_EXT(OB_ZONE_DUPLICATED, -4641, -1, "HY000", "Duplicated zone in zone list", "Duplicated zone \'%s\' in zone
list %s");
DEFINE_ERROR(OB_NOT_ALL_ZONE_ACTIVE, -4642, -1, "HY000", "Not all zone in zone list are active");
DEFINE_ERROR_EXT(OB_PRIMARY_ZONE_NOT_IN_ZONE_LIST, -4643, -1, "HY000", "primary zone not in zone list", "primary zone \'%
s\' not in zone list %s");
DEFINE_ERROR(OB_REPLICA_NUM_NOT_MATCH, -4644, -1, "HY000", "replica num not same with zone count");
DEFINE_ERROR_EXT(OB_ZONE_LIST_POOL_LIST_NOT_MATCH, -4645, -1, "HY000", "zone list not a subset of  resource pool list", "
zone list %s not a subset of resource pool zone list %s");
DEFINE_ERROR_EXT(OB_INVALID_TENANT_NAME, -4646, -1, "HY000", "tenant name is too long", "tenant name \'%s\' over max_tena
nt_name_length %ld");
DEFINE_ERROR(OB_EMPTY_RESOURCE_POOL_LIST, -4647, -1, "HY000", "resource pool list is empty");
DEFINE_ERROR_EXT(OB_RESOURCE_UNIT_NOT_EXIST, -4648, -1, "HY000", "resource unit not exist", "resource unit \'%s\' not exi
st");
DEFINE_ERROR_EXT(OB_RESOURCE_UNIT_EXIST, -4649, -1, "HY000", "resource unit already exist", "resource unit \'%s\' already
exist");
DEFINE_ERROR_EXT(OB_RESOURCE_POOL_NOT_EXIST, -4650, -1, "HY000", "resource pool not exist", "resource pool \'%s\' not exi
st");
DEFINE_ERROR_EXT(OB_RESOURCE_POOL_EXIST, -4651, -1, "HY000", "resource pool already exist", "resource pool \'%s\' exist")
;
DEFINE_ERROR(OB_WAIT_LEADER_SWITCH_TIMEOUT, -4652, -1, "HY000", "wait leader switch timeout");
DEFINE_ERROR(OB_LOCATION_NOT_EXIST, -4653, -1, "HY000", "location not exist");
DEFINE_ERROR(OB_LOCATION_LEADER_NOT_EXIST, -4654, -1, "HY000", "location leader not exist");
DEFINE_ERROR(OB_ZONE_NOT_ACTIVE, -4655, -1, "HY000", "zone not active");
DEFINE_ERROR(OB_UNIT_NUM_OVER_SERVER_COUNT, -4656, -1, "HY000", "resource pool unit num is bigger than zone server count"
);
DEFINE_ERROR_EXT(OB_POOL_SERVER_INTERSECT, -4657, -1, "HY000", "resource pool list unit server intersect", "resource pool
list %s unit servers intersect");
DEFINE_ERROR_EXT(OB_NOT_SINGLE_RESOURCE_POOL, -4658, -1, "HY000", "create tenant only support single resource pool now",
"create tenant only support single resource pool now, but pool list is %s");
DEFINE_ERROR_EXT(OB_INVALID_RESOURCE_UNIT, -4659, -1, "HY000", "invalid resource unit", "invalid resource unit, %s\'s min
value is %s");
DEFINE_ERROR_EXT(OB_STOP_SERVER_IN_MULTIPLE_ZONES, -4660, -1, "HY000", "Can not stop server in multiple zones", "Can not
stop server in multiple zones, there are already servers stopped in zone:%s");
DEFINE_ERROR(OB_SESSION_ENTRY_EXIST, -4661, -1, "HY000", "Session already exist");
DEFINE_ERROR_EXT(OB_GOT_SIGNAL_ABORTING, -4662, ER_GOT_SIGNAL, "01000", "Got signal. Aborting!", "%s: Got signal %d. Abor
ting!") ;
DEFINE_ERROR(OB_SERVER_NOT_ALIVE, -4663, -1, "HY000", "server is not alive");
DEFINE_ERROR(OB_GET_LOCATION_TIME_OUT, -4664, 4012, "HY000", "Timeout");
DEFINE_ERROR(OB_UNIT_IS_MIGRATING, -4665, -1, "HY000", "Unit is migrating, can not migrate again");
DEFINE_ERROR_EXT(OB_CLUSTER_NO_MATCH, -4666, -1, "HY000", "cluster name is not match", "cluster name is not match to \'%s
\'");
DEFINE_ERROR(OB_CHECK_ZONE_MERGE_ORDER, -4667, -1, "HY000", "Please check new zone in zone_merge_order. You can show para
meters like 'zone_merge_order'");
DEFINE_ERROR_EXT(OB_ERR_ZONE_NOT_EMPTY, -4668, -1, "HY000", "zone not empty", "The zone is not empty and can not be delet
ed. You should delete the servers of the zone. There are %ld servers alive and %ld not alive.") ;
////////////////////////////////////////////////////////////////
// SQL & Schema specific error code, -5000 ~ -6000
////////////////////////////////////////////////////////////////
DEFINE_ERROR(OB_ERR_PARSER_INIT, -5000, ER_PARSE_ERROR, "0B000", "Failed to init SQL parser");
DEFINE_ERROR_EXT(OB_ERR_PARSE_SQL, -5001, ER_PARSE_ERROR, "42000", "Parse error", "%s near \'%.*s\' at line %d");
DEFINE_ERROR(OB_ERR_RESOLVE_SQL, -5002, -1, "HY000", "Resolve error");
DEFINE_ERROR(OB_ERR_GEN_PLAN, -5003, -1, "HY000", "Generate plan error");
DEFINE_ERROR(OB_ERR_PARSER_SYNTAX, -5006, ER_SYNTAX_ERROR, "42000", "You have an error in your SQL syntax; check the manu
al that corresponds to your MySQL server version for the right syntax to use");
DEFINE_ERROR(OB_ERR_COLUMN_SIZE, -5007, ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT, "21000", "The used SELECT statements have a
different number of columns");
// xiyu@TODO: will be replaced by OB_NON_UNIQ_ERROR
DEFINE_ERROR_EXT(OB_ERR_COLUMN_DUPLICATE, -5008, ER_DUP_FIELDNAME, "42S21", "Duplicate column name", "Duplicate column na
me '%.*s'");
DEFINE_ERROR(OB_ERR_OPERATOR_UNKNOWN, -5010, -1, "21000", "Unknown operator");
DEFINE_ERROR(OB_ERR_STAR_DUPLICATE, -5011, -1, "42000", "Duplicated star");
DEFINE_ERROR_EXT(OB_ERR_ILLEGAL_ID, -5012, -1, "HY000", "Illegal ID", "%s");
DEFINE_ERROR(OB_ERR_ILLEGAL_VALUE, -5014, -1, "HY000", "Illegal value");
DEFINE_ERROR(OB_ERR_COLUMN_AMBIGUOUS, -5015, ER_AMBIGUOUS_FIELD_TERM, "42000", "Ambiguous column");
DEFINE_ERROR(OB_ERR_LOGICAL_PLAN_FAILD, -5016, -1, "HY000", "Generate logical plan error");
DEFINE_ERROR(OB_ERR_SCHEMA_UNSET, -5017, -1, "HY000", "Schema not set");
DEFINE_ERROR(OB_ERR_ILLEGAL_NAME, -5018, -1, "42000", "Illegal name");
DEFINE_ERROR_EXT(OB_ERR_TABLE_EXIST, -5020, ER_TABLE_EXISTS_ERROR, "42S01", "Table already exists", "Table '%.*s' already
exists");
DEFINE_ERROR_EXT(OB_TABLE_NOT_EXIST, -5019, ER_NO_SUCH_TABLE, "42S02", "Table doesn\'t exist", "Table \'%s.%s\' doesn\'t
exist");
DEFINE_ERROR(OB_ERR_EXPR_UNKNOWN, -5022, -1, "42000", "Unknown expression");
DEFINE_ERROR_EXT(OB_ERR_ILLEGAL_TYPE, -5023, -1, "S1004", "Illegal type", "unsupport MySQL type %d. Maybe you should use
java.sql.Timestamp instead of java.util.Date.") ;
DEFINE_ERROR_EXT(OB_ERR_KEY_NAME_DUPLICATE, -5025, ER_DUP_KEYNAME, "42000", "Duplicated key name", "Duplicate key name \'
%.*s\'");
DEFINE_ERROR_EXT(OB_ERR_PRIMARY_KEY_DUPLICATE, -5024, ER_DUP_ENTRY, "23000", "Duplicated primary key", "Duplicate entry \
'%s\' for key \'%.*s\'");
DEFINE_ERROR(OB_ERR_CREATETIME_DUPLICATE, -5026, -1, "42000", "Duplicated createtime");
DEFINE_ERROR(OB_ERR_MODIFYTIME_DUPLICATE, -5027, -1, "42000", "Duplicated modifytime");
DEFINE_ERROR(OB_ERR_ILLEGAL_INDEX, -5028, ER_NO_SUCH_INDEX, "42S12", "Illegal index");
DEFINE_ERROR(OB_ERR_INVALID_SCHEMA, -5029, -1, "HY000", "Invalid schema");
DEFINE_ERROR(OB_ERR_INSERT_NULL_ROWKEY, -5030, ER_PRIMARY_CANT_HAVE_NULL, "42000", "Insert null rowkey");
DEFINE_ERROR(OB_ERR_COLUMN_NOT_FOUND, -5031, -1, "HY000", "Column not found");
DEFINE_ERROR(OB_ERR_DELETE_NULL_ROWKEY, -5032, -1, "23000", "Delete null rowkey");
DEFINE_ERROR(OB_ERR_USER_EMPTY, -5034, -1, "01007", "No user");
DEFINE_ERROR(OB_ERR_USER_NOT_EXIST, -5035, ER_NO_SUCH_USER, "01007", "User not exist");
DEFINE_ERROR_EXT(OB_ERR_NO_PRIVILEGE, -5036, ER_SPECIFIC_ACCESS_DENIED_ERROR, "42501", "Access denied", "Access denied; y
ou need (at least one of) the %s privilege(s) for this operation");
DEFINE_ERROR(OB_ERR_NO_AVAILABLE_PRIVILEGE_ENTRY, -5037, -1, "HY000", "No privilege entry");
DEFINE_ERROR(OB_ERR_WRONG_PASSWORD, -5038, ER_PASSWORD_NO_MATCH, "42000", "Incorrect password");
DEFINE_ERROR(OB_ERR_USER_IS_LOCKED, -5039, -1, "01007", "User locked");
DEFINE_ERROR(OB_ERR_UPDATE_ROWKEY_COLUMN, -5040, -1, "42000", "Can not update rowkey column");
DEFINE_ERROR(OB_ERR_UPDATE_JOIN_COLUMN, -5041, -1, "42000", "Can not update join column");
DEFINE_ERROR_EXT(OB_ERR_INVALID_COLUMN_NUM, -5042, ER_OPERAND_COLUMNS, "21000", "Invalid column number", "Operand should
contain %d column(s)");
DEFINE_ERROR_EXT(OB_ERR_PREPARE_STMT_NOT_FOUND, -5043, ER_UNKNOWN_STMT_HANDLER, "HY007", "Unknown prepared statement", "s
tatement not prepared, stmt_id=%u");
DEFINE_ERROR_EXT(OB_ERR_SYS_VARIABLE_UNKNOWN, -5044, ER_UNKNOWN_SYSTEM_VARIABLE, "HY000", "Unknown system variable", "Unk
nown system variable '%.*s'");
DEFINE_ERROR(OB_ERR_OLDER_PRIVILEGE_VERSION, -5046, -1, "HY000", "Older privilege version");
DEFINE_ERROR_EXT(OB_ERR_LACK_OF_ROWKEY_COL, -5047, ER_REQUIRES_PRIMARY_KEY, "42000", "No rowkey column specified", "Prima
ry key column(s) not specified in the WHERE clause");
DEFINE_ERROR(OB_ERR_USER_EXIST, -5050, -1, "42710", "User exists");
DEFINE_ERROR(OB_ERR_PASSWORD_EMPTY, -5051, -1, "HY000", "Empty password");
DEFINE_ERROR(OB_ERR_GRANT_PRIVILEGES_TO_CREATE_TABLE, -5052, -1, "42000", "Failed to grant privelege");
DEFINE_ERROR_EXT(OB_ERR_WRONG_DYNAMIC_PARAM, -5053, -1, "HY093", "Wrong dynamic parameters", "Incorrect arguments number
to EXECUTE, need %ld arguments but give %ld");
DEFINE_ERROR_EXT(OB_ERR_PARAM_SIZE, -5054, ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT, "42000", "Incorrect parameter count", "Inco
rrect parameter count in the call to native function '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_FUNCTION_UNKNOWN, -5055, ER_SP_DOES_NOT_EXIST, "42000", "FUNCTION does not exist", "%s %s does no
t exist");
DEFINE_ERROR(OB_ERR_CREAT_MODIFY_TIME_COLUMN, -5056, -1, "23000", "CreateTime or ModifyTime column cannot be modified");
DEFINE_ERROR(OB_ERR_MODIFY_PRIMARY_KEY, -5057, -1, "23000", "Primary key cannot be modified");
DEFINE_ERROR(OB_ERR_PARAM_DUPLICATE, -5058, -1, "42000", "Duplicated parameters");
DEFINE_ERROR(OB_ERR_TOO_MANY_SESSIONS, -5059, ER_TOO_MANY_USER_CONNECTIONS, "42000", "Too many sessions");
DEFINE_ERROR(OB_ERR_TOO_MANY_PS, -5061, -1, "54023", "Too many prepared statements");
DEFINE_ERROR(OB_ERR_HINT_UNKNOWN, -5063, -1, "42000", "Unknown hint");
DEFINE_ERROR(OB_ERR_WHEN_UNSATISFIED, -5064, -1, "23000", "When condition not satisfied");
DEFINE_ERROR(OB_ERR_QUERY_INTERRUPTED, -5065, ER_QUERY_INTERRUPTED, "70100", "Query execution was interrupted");
DEFINE_ERROR(OB_ERR_SESSION_INTERRUPTED, -5066, -1, "HY000", "Session interrupted");
DEFINE_ERROR(OB_ERR_UNKNOWN_SESSION_ID, -5067, -1, "HY000", "Unknown session ID");
DEFINE_ERROR(OB_ERR_PROTOCOL_NOT_RECOGNIZE, -5068, -1, "HY000", "Incorrect protocol");
DEFINE_ERROR(OB_ERR_WRITE_AUTH_ERROR, -5069, -1, "HY000", "Write auth packet error");
DEFINE_ERROR(OB_ERR_PARSE_JOIN_INFO, -5070, -1, "42000", "Wrong join info")
DEFINE_ERROR(OB_ERR_ALTER_INDEX_COLUMN, -5071, -1, "42000", "Cannot alter index column");
DEFINE_ERROR(OB_ERR_MODIFY_INDEX_TABLE, -5072, -1, "42000", "Cannot modify index table");
DEFINE_ERROR(OB_ERR_INDEX_UNAVAILABLE, -5073, ER_NO_SUCH_INDEX, "42000", "Index unavailable");
DEFINE_ERROR(OB_ERR_NOP_VALUE, -5074, -1, "23000", "NOP cannot be used here");
DEFINE_ERROR(OB_ERR_PS_TOO_MANY_PARAM, -5080, ER_PS_MANY_PARAM, "54000", "Prepared statement contains too many placeholde
rs");
DEFINE_ERROR(OB_ERR_READ_ONLY, -5081, -1, "25000", "The server is read only now");
DEFINE_ERROR_EXT(OB_ERR_INVALID_TYPE_FOR_OP, -5083, -1, "22000", "Invalid data type for the operation", "invalid obj type
for type promotion, left_type=%d right_type=%d");
DEFINE_ERROR(OB_ERR_CAST_VARCHAR_TO_BOOL, -5084, -1, "22000", "Can not cast varchar value to bool type");
DEFINE_ERROR(OB_ERR_CAST_VARCHAR_TO_NUMBER, -5085, -1, "22000", "Not a number Can not cast varchar value to number type")
;
DEFINE_ERROR(OB_ERR_CAST_VARCHAR_TO_TIME, -5086, -1, "22000", "Not timestamp Can not cast varchar value to timestamp type
");
DEFINE_ERROR(OB_ERR_CAST_NUMBER_OVERFLOW, -5087, -1, "22000", "Result value was out of range when cast to number");
DEFINE_ERROR_EXT(OB_INTEGER_PRECISION_OVERFLOW, -5088, -1, "22000", "Result value was out of range when cast varchar to n
umber", "value larger than specified precision(%ld,%ld) allowed for this column");
DEFINE_ERROR_EXT(OB_DECIMAL_PRECISION_OVERFLOW, -5089, -1, "22000", "Result value was out of range when cast varchar to n
umber", "value(%s) larger than specified precision(%ld,%ld) allowed for this column");
DEFINE_ERROR(OB_SCHEMA_NUMBER_PRECISION_OVERFLOW, -5090, -1, "22000", "Precision was out of range");
DEFINE_ERROR(OB_SCHEMA_NUMBER_SCALE_OVERFLOW, -5091, -1, "22000", "Scale value was out of range");
DEFINE_ERROR(OB_ERR_INDEX_UNKNOWN, -5092, -1, "42000", "Unknown index");
DEFINE_ERROR(OB_NUMERIC_OVERFLOW, -5093, -1, "22000", "numeric overflow");
DEFINE_ERROR(OB_ERR_TOO_MANY_JOIN_TABLES, -5094, -1, "HY000", "too many joined tables");
DEFINE_ERROR_EXT(OB_ERR_VARCHAR_TOO_LONG, -5098, -1, "22001", "Varchar value is too long for the column", "Data too long(
%d>%ld) for column '%s'");
DEFINE_ERROR(OB_ERR_SYS_CONFIG_UNKNOWN, -5099, -1, "42000", "System config unknown");
DEFINE_ERROR_EXT(OB_ERR_LOCAL_VARIABLE, -5100, ER_LOCAL_VARIABLE, "HY000", "Local variable", "Variable \'%.*s\' is a SESS
ION variable and can't be used with SET GLOBAL");
DEFINE_ERROR_EXT(OB_ERR_GLOBAL_VARIABLE, -5101, ER_GLOBAL_VARIABLE, "HY000", "Global variable", "Variable \'%.*s\' is a G
LOBAL variable and should be set with SET GLOBAL");
DEFINE_ERROR_EXT(OB_ERR_VARIABLE_IS_READONLY, -5102, ER_VARIABLE_IS_READONLY, "HY000", "variable is read only", "%.*s var
iable '%.*s' is read-only. Use SET %.*s to assign the value");
DEFINE_ERROR_EXT(OB_ERR_INCORRECT_GLOBAL_LOCAL_VAR, -5103, ER_INCORRECT_GLOBAL_LOCAL_VAR, "HY000", "incorrect global or l
ocal variable", "Variable '%.*s' is a %.*s variable");
DEFINE_ERROR_EXT(OB_ERR_EXPIRE_INFO_TOO_LONG, -5104, -1, "42000", "Expire expression too long", "length(%d) of expire_inf
o is larger than the max allowed(%ld)");
DEFINE_ERROR_EXT(OB_ERR_EXPIRE_COND_TOO_LONG, -5105, -1, "42000", "Expire condition too long", "total length(%ld) of expi
re_info and its expression is larger than the max allowed(%ld)");
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT_FOR_EXTRACT, -5106, -1, "42000", "Invalid argument for extract()", "EXTRACT() expect
ed timestamp or a string as date argument");
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT_FOR_IS, -5107, -1, "42000", "Invalid argument for IS operator", "Invalid operand typ
e for IS operator, lval=%s");
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT_FOR_LENGTH, -5108, -1, "42000", "Invalid argument for length()", "function LENGTH()
expected a varchar argument");
```

```
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT_FOR_SUBSTR, -5109, -1, "42000", "Invalid argument for substr()", "invalid input form
at. ret=%d text=%s start=%s length=%s");
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT_FOR_TIME_TO_USEC, -5110, -1, "42000", "Invalid argument for time_to_usec()", "TIME_T
O_USEC() expected timestamp or a string as date argument");
DEFINE_ERROR_EXT(OB_INVALID_ARGUMENT_FOR_USEC_TO_TIME, -5111, -1, "42000", "Invalid argument for usec_to_time()", "USEC_T
O_TIME expected a interger number as usec argument");
DEFINE_ERROR_EXT(OB_ERR_USER_VARIABLE_UNKNOWN, -5112, -1, "42P01", "Unknown user variable", "Variable %.*s does not exist
s");
DEFINE_ERROR_EXT(OB_ILLEGAL_USAGE_OF_MERGING_FROZEN_TIME, -5113, -1, "42000", "Illegal usage of merging_frozen_time()", "
MERGING_FROZEN_TIME() system function only be used in daily merging.") ;
DEFINE_ERROR_EXT(OB_INVALID_NUMERIC, -5114, -1, "42000", "Invalid numeric", "Invalid numeric char '%c'");
DEFINE_ERROR(OB_ERR_REGEXP_ERROR, -5115, ER_REGEXP_ERROR, "42000", "Got error 'empty (sub)expression' from regexp");
DEFINE_ERROR(OB_SQL_LOG_OP_SETCHILD_OVERFLOW, -5116, -1, "HY000", "Logical operator child index overflow");
DEFINE_ERROR(OB_SQL_EXPLAIN_FAILED, -5117, -1, "HY000", "fail to explain plan");
DEFINE_ERROR(OB_SQL_OPT_COPY_OP_FAILED, -5118, -1, "HY000", "fail to copy logical operator");
DEFINE_ERROR(OB_SQL_OPT_GEN_PLAN_FALIED, -5119, -1, "HY000", "fail to generate plan");;
DEFINE_ERROR(OB_SQL_OPT_CREATE_RAWEXPR_FAILED, -5120, -1,  "HY000", "fail to create raw expr");
DEFINE_ERROR(OB_SQL_OPT_JOIN_ORDER_FAILED, -5121, -1,  "HY000", "fail to generate join order");
DEFINE_ERROR(OB_SQL_OPT_ERROR, -5122, -1,  "HY000", "optimizer general error");
DEFINE_ERROR(OB_SQL_RESOLVER_NO_MEMORY, -5130, -1, "HY000", "sql resolver no memory");
DEFINE_ERROR(OB_SQL_DML_ONLY, -5131, -1, "HY000", "plan cache support dml only");
DEFINE_ERROR(OB_ERR_NO_GRANT, -5133, -1, "42000", "No such grant defined");
DEFINE_ERROR(OB_ERR_NO_DB_SELECTED, -5134, ER_NO_DB_ERROR, "3D000", "No database selected");
DEFINE_ERROR(OB_SQL_PC_OVERFLOW, -5135, -1, "HY000", "plan cache is overflow");
DEFINE_ERROR(OB_SQL_PC_PLAN_DUPLICATE, -5136, -1, "HY000", "plan exists in plan cache already");
DEFINE_ERROR(OB_SQL_PC_PLAN_EXPIRE, -5137, -1, "HY000", "plan is expired");
DEFINE_ERROR(OB_SQL_PC_NOT_EXIST, -5138, -1, "HY000", "no plan exist");
DEFINE_ERROR(OB_SQL_PARAMS_LIMIT, -5139, -1, "HY000", "too many params, plan cache not support" );
DEFINE_ERROR(OB_SQL_PC_PLAN_SIZE_LIMIT, -5140, -1, "HY000", "plan is too big to add to plan cache");
DEFINE_ERROR_EXT(OB_ERR_UNKNOWN_CHARSET, -5142, ER_UNKNOWN_CHARACTER_SET, "42000", "Unknown character set", "Unknown char
acter set: '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_UNKNOWN_COLLATION, -5143, ER_UNKNOWN_COLLATION, "HY000", "Unknown collation", "Unknown collation:
'%.*s'");
DEFINE_ERROR_EXT(OB_ERR_COLLATION_MISMATCH, -5144, ER_COLLATION_CHARSET_MISMATCH, "42000", "The collation is not valid fo
r the character set", "COLLATION '%.*s' is not valid for CHARACTER SET '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_WRONG_VALUE_FOR_VAR, -5145, ER_WRONG_VALUE_FOR_VAR, "42000", "Variable can't be set to the value"
, "Variable \'%.*s\' can't be set to the value of \'%.*s\'");
DEFINE_ERROR_EXT(OB_UNKNOWN_PARTITION, -5146, ER_UNKNOWN_PARTITION, "HY000", "Unknown partition", "Unkown partition '%.*s
' in table '%.*s'");
DEFINE_ERROR(OB_PARTITION_NOT_MATCH, -5147, ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET, "HY000", "Found a row not matching
the given partition set");
DEFINE_ERROR(OB_ER_PASSWD_LENGTH, -5148, -1, "HY000", " Password hash should be a 40-digit hexadecimal number");
DEFINE_ERROR(OB_ERR_INSERT_INNER_JOIN_COLUMN, -5149, -1, "07000", "Insert inner join column error");
DEFINE_ERROR(OB_TENANT_NOT_IN_SERVER, -5150, -1, "HY000", "Tenant not in this server");
DEFINE_ERROR(OB_TABLEGROUP_NOT_EXIST, -5151, -1, "42P01", "tablegroup not exist");
DEFINE_ERROR(OB_SUBQUERY_TOO_MANY_ROW, -5153, ER_SUBQUERY_NO_1_ROW, "21000", "Subquery returns more than 1 row");
DEFINE_ERROR_EXT(OB_ERR_BAD_DATABASE, -5154, ER_BAD_DB_ERROR, "42000", "Unknown database", "Unknown database '%.*s'");
DEFINE_ERROR_EXT(OB_CANNOT_USER, -5155, ER_CANNOT_USER, "HY000", "User operation failed", "Operation %.*s failed for %.*s
");
DEFINE_ERROR_EXT(OB_TENANT_EXIST, -5156, -1, "HY000", "tenant already exist", "tenant \'%s\' already exist");
DEFINE_ERROR_EXT(OB_TENANT_NOT_EXIST, -5157, -1, "HY000", "Unknown tenant", "Unknown tenant '%.*s'");
DEFINE_ERROR_EXT(OB_DATABASE_EXIST, -5158, ER_DB_CREATE_EXISTS, "HY000", "Can't create database;database exists", "Can't
create database '%.*s'; database exists");
DEFINE_ERROR(OB_TABLEGROUP_EXIST, -5159, -1, "HY000", "tablegroup already exist");
DEFINE_ERROR(OB_ERR_INVALID_TENANT_NAME, -5160, -1, "HY000", "invalid tenant name specified in connection string");
DEFINE_ERROR(OB_EMPTY_TENANT, -5161, -1, "HY000", "tenant is empty");
DEFINE_ERROR_EXT(OB_WRONG_DB_NAME, -5162, ER_WRONG_DB_NAME, "42000", "Incorrect database name", "Incorrect database name
'%.*s'");
DEFINE_ERROR_EXT(OB_WRONG_TABLE_NAME, -5163, ER_WRONG_TABLE_NAME, "42000", "Incorrect table name", "Incorrect table name
'%.*s'");
DEFINE_ERROR_EXT(OB_WRONG_COLUMN_NAME, -5164, ER_WRONG_COLUMN_NAME, "42000", "Incorrect column name", "Incorrect column n
ame '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_COLUMN_SPEC, -5165, ER_WRONG_FIELD_SPEC, "42000", "Incorrect column specifier", "Incorrect column
specifier for column '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_DB_DROP_EXISTS, -5166, ER_DB_DROP_EXISTS, "HY000", "Can't drop database;database doesn't exist",
"Can't drop database '%.*s'; database doesn't exist");
DEFINE_ERROR_EXT(OB_ERR_DATA_TOO_LONG, -5167, ER_DATA_TOO_LONG, "22001", "Data too long for column", "Data too long for c
olumn '%.*s' at row %lld");
DEFINE_ERROR_EXT(OB_ERR_WRONG_VALUE_COUNT_ON_ROW, -5168, ER_WRONG_VALUE_COUNT_ON_ROW, "21S01", "column count does not mat
ch value count", "column count does not match value count at row '%d'");
DEFINE_ERROR(OB_ERR_CREATE_USER_WITH_GRANT, -5169, ER_CANT_CREATE_USER_WITH_GRANT, "42000", "You are not allowed to creat
e a user with GRANT");
DEFINE_ERROR_EXT(OB_ERR_NO_DB_PRIVILEGE, -5170, ER_DBACCESS_DENIED_ERROR, "42000", "Access denied for user to database",
"Access denied for user '%.*s'@'%.*s' to database '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_NO_TABLE_PRIVILEGE, -5171, ER_TABLEACCESS_DENIED_ERROR, "42000", "Command denied to user for tabl
e", "%.*s command denied to user '%.*s'@'%.*s' for table '%.*s'");
DEFINE_ERROR_EXT(OB_INVALID_ON_UPDATE, -5172, ER_INVALID_ON_UPDATE, "HY000", "Invalid ON UPDATE clause", "Invalid ON UPDA
TE clause for \'%s\' column");
DEFINE_ERROR_EXT(OB_INVALID_DEFAULT, -5173, ER_INVALID_DEFAULT, "42000", "Invalid default value", "Invalid default value
for \'%.*s\'");
DEFINE_ERROR_EXT(OB_ERR_UPDATE_TABLE_USED, -5174, ER_UPDATE_TABLE_USED, "HY000", "Update table used", "You can\'t specify
target table \'%s\' for update in FROM clause");
DEFINE_ERROR_EXT(OB_ERR_COULUMN_VALUE_NOT_MATCH, -5175, ER_WRONG_VALUE_COUNT_ON_ROW, "21S01", "Column count doesn\'t matc
h value count", "Column count doesn\'t match value count at row %ld");
DEFINE_ERROR(OB_ERR_INVALID_GROUP_FUNC_USE, -5176, ER_INVALID_GROUP_FUNC_USE, "HY000", "Invalid use of group function");
DEFINE_ERROR_EXT(OB_CANT_AGGREGATE_2COLLATIONS, -5177, ER_CANT_AGGREGATE_2COLLATIONS, "HY000", "Illegal mix of collations
", "Illegal mix of collations");
DEFINE_ERROR_EXT(OB_ERR_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD, -5178, ER_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD, "
HY000", "Field is of a not allowed type for this type of partitioning", "Field \'%.*s\' is of a not allowed type for this
type of partitioning");
DEFINE_ERROR_EXT(OB_ERR_TOO_LONG_IDENT, -5179, ER_TOO_LONG_IDENT, "42000", "Identifier name is too long", "Identifier nam
e \'%.*s\' is too long");
DEFINE_ERROR_EXT(OB_ERR_WRONG_TYPE_FOR_VAR, -5180, ER_WRONG_TYPE_FOR_VAR, "42000", "Incorrect argument type to variable",
"Incorrect argument type to variable '%.*s'");
DEFINE_ERROR_EXT(OB_WRONG_USER_NAME_LENGTH, -5181, ER_WRONG_STRING_LENGTH, "HY000", "String is too long for user_name (sh
ould be no longer than 16)", "String '%.*s' is too long for user name (should be no longer than 16)");
DEFINE_ERROR(OB_ERR_PRIV_USAGE, -5182, ER_WRONG_USAGE, "HY000", "Incorrect usage of DB GRANT and GLOBAL PRIVILEGES");
DEFINE_ERROR(OB_ILLEGAL_GRANT_FOR_TABLE, -5183, ER_ILLEGAL_GRANT_FOR_TABLE, "42000", "Illegal GRANT/REVOKE command; pleas
e consult the manual to see which privileges can be used");
DEFINE_ERROR(OB_ERR_REACH_AUTOINC_MAX, -5184, ER_AUTOINC_READ_FAILED, "HY000", "Failed to read auto-increment value from
storage engine");
DEFINE_ERROR(OB_ERR_NO_TABLES_USED, -5185, ER_NO_TABLES_USED, "HY000", "No tables used");
DEFINE_ERROR(OB_CANT_REMOVE_ALL_FIELDS, -5187, ER_CANT_REMOVE_ALL_FIELDS, "42000", "You can't delete all columns with ALT
ER TABLE; use DROP TABLE instead");
DEFINE_ERROR(OB_TOO_MANY_PARTITIONS_ERROR, -5188, ER_TOO_MANY_PARTITIONS_ERROR, "HY000", "Too many partitions (including
subpartitions) were defined", "Too many partitions (including subpartitions) were defined")
DEFINE_ERROR(OB_NO_PARTS_ERROR, -5189, ER_NO_PARTS_ERROR, "HY000", "Number of partitions = 0 is not an allowed value", "N
umber of partitions = 0 is not an allowed value");
DEFINE_ERROR(OB_WRONG_SUB_KEY, -5190, ER_WRONG_SUB_KEY, "HY000", "Incorrect prefix key; the used key part isn't a string,
the used length is longer than the key part, or the storage engine doesn't support unique prefix keys");
DEFINE_ERROR_EXT(OB_KEY_PART_0, -5191, ER_KEY_PART_0, "HY000", "Key part length cannot be 0", "Key part \'%.*s\' length c
annot be 0");
DEFINE_ERROR_EXT(OB_ERR_UNKNOWN_TIME_ZONE, -5192, ER_UNKNOWN_TIME_ZONE, "HY000", "Unknown or incorrect time zone", "Unkno
wn or incorrect time zone: \'%.*s\'");
DEFINE_ERROR(OB_ERR_WRONG_AUTO_KEY, -5193, ER_WRONG_AUTO_KEY, "42000", "Incorrect table definition; there can be only one
auto column");
DEFINE_ERROR_EXT(OB_ERR_TOO_MANY_KEYS, -5194, ER_TOO_MANY_KEYS, "42000","Too many keys specified", "Too many keys specifi
ed; max %d keys allowed");
DEFINE_ERROR_EXT(OB_ERR_TOO_MANY_ROWKEY_COLUMNS, -5195, ER_TOO_MANY_KEY_PARTS, "42000","Too many key parts specified", "T
oo many key parts specified; max %d parts allowed");
DEFINE_ERROR_EXT(OB_ERR_TOO_LONG_KEY_LENGTH, -5196, ER_TOO_LONG_KEY, "42000", "Specified key was too long", "Specified ke
y was too long; max key length is %d bytes");
DEFINE_ERROR(OB_ERR_TOO_MANY_COLUMNS, -5197, ER_TOO_MANY_FIELDS, "42000", "Too many columns");
DEFINE_ERROR_EXT(OB_ERR_TOO_LONG_COLUMN_LENGTH, -5198, ER_TOO_BIG_FIELDLENGTH, "42000", "Column length too big", "Column
length too big for column '%s' (max = %d)");
DEFINE_ERROR(OB_ERR_TOO_BIG_ROWSIZE, -5199, ER_TOO_BIG_ROWSIZE, "42000", "Row size too large");
DEFINE_ERROR_EXT(OB_ERR_UNKNOWN_TABLE, -5200, ER_UNKNOWN_TABLE, "42S02", "Unknown table", "Unknown table '%.*s' in %.*s")
;
DEFINE_ERROR_EXT(OB_ERR_BAD_TABLE, -5201, ER_BAD_TABLE_ERROR, "42S02", "Unknown table", "Unknown table '%.*s'");
DEFINE_ERROR(OB_ERR_TOO_BIG_SCALE, -5202, ER_TOO_BIG_SCALE, "42000", "Too big scale %d specified for column '%s'. Maximum
is %ld.") ;
DEFINE_ERROR(OB_ERR_TOO_BIG_PRECISION, -5203, ER_TOO_BIG_PRECISION, "42000", "Too big precision %d specified for column '
%s'. Maximum is %ld.") ;
DEFINE_ERROR(OB_ERR_M_BIGGER_THAN_D, -5204, ER_M_BIGGER_THAN_D, "42000", "For float(M,D), double(M,D) or decimal(M,D), M
must be >= D (column '%s').") ;
DEFINE_ERROR(OB_ERR_TOO_BIG_DISPLAYWIDTH, -5205, ER_TOO_BIG_DISPLAYWIDTH, "42000", "Display width out of range for column
'%s' (max = %ld)");
DEFINE_ERROR(OB_WRONG_GROUP_FIELD, -5206, ER_WRONG_GROUP_FIELD, "42000", "Can't group on '%.*s'");
DEFINE_ERROR_EXT(OB_NON_UNIQ_ERROR, -5207, ER_NON_UNIQ_ERROR, "23000", "Column is ambiguous", "Column '%.*s' in %.*s is a
mbiguous");
DEFINE_ERROR_EXT(OB_ERR_NONUNIQ_TABLE, -5208, ER_NONUNIQ_TABLE, "42000", "Not unique table/alias", "Not unique table/alia
s: \'%.*s\'");
```

```
DEFINE_ERROR_EXT(OB_ERR_CANT_DROP_FIELD_OR_KEY, -5209, ER_CANT_DROP_FIELD_OR_KEY, "42000", "Can't DROP Column; check that
column/key exists", "Can't DROP '%.*s'; check that column/key exists");
DEFINE_ERROR(OB_ERR_MULTIPLE_PRI_KEY, -5210, ER_MULTIPLE_PRI_KEY, "42000", "Multiple primary key defined");
DEFINE_ERROR_EXT(OB_ERR_KEY_COLUMN_DOES_NOT_EXITS, -5211, ER_KEY_COLUMN_DOES_NOT_EXITS, "42000", "Key column doesn't exis
t in table", "Key column '%.*s' doesn't exist in table");
DEFINE_ERROR_EXT(OB_ERR_AUTO_PARTITION_KEY, -5212, -1, "42000", "auto-increment column should not be part of partition ke
y", "auto-increment column '%.*s' should not be part of partition key");
DEFINE_ERROR_EXT(OB_ERR_CANT_USE_OPTION_HERE, -5213, ER_CANT_USE_OPTION_HERE, "42000", "Incorrect usage/placement", "Inco
rrect usage/placement of '%s'");
DEFINE_ERROR_EXT(OB_ERR_WRONG_OBJECT, -5214, ER_WRONG_OBJECT, "HY000", "Wrong object", "\'%s.%s\' is not %s");
DEFINE_ERROR_EXT(OB_ERR_ON_RENAME, -5215, ER_ERROR_ON_RENAME, "HY000", "Error on rename table", "Error on rename of \'%s.
%s\' to \'%s.%s\'");
DEFINE_ERROR_EXT(OB_ERR_WRONG_KEY_COLUMN, -5216, ER_WRONG_KEY_COLUMN, "42000", "The used storage engine can't index colum
n", "The used storage engine can't index column '%.*s'");
DEFINE_ERROR_EXT(OB_ERR_BAD_FIELD_ERROR, -5217, ER_BAD_FIELD_ERROR, "42S22", "Unknown column", "Unknown column '%.*s' in
'%.*s'");
DEFINE_ERROR_EXT(OB_ERR_WRONG_FIELD_WITH_GROUP, -5218, ER_WRONG_FIELD_WITH_GROUP, "42000", "column is not in GROUP BY", "
\'%.*s\' is not in GROUP BY");
DEFINE_ERROR(OB_ERR_CANT_CHANGE_TX_CHARACTERISTICS, -5219, ER_CANT_CHANGE_TX_CHARACTERISTICS, "25001", "Transaction chara
cteristics can't be changed while a transaction is in progress");
DEFINE_ERROR(OB_ERR_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION, -5220, ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION, "25006", "Can
not execute statement in a READ ONLY transaction.") ;
DEFINE_ERROR(OB_ERR_MIX_OF_GROUP_FUNC_AND_FIELDS, -5221, ER_MIX_OF_GROUP_FUNC_AND_FIELDS, "42000", "Mixing of GROUP colum
ns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause");
DEFINE_ERROR_EXT(OB_ERR_TRUNCATED_WRONG_VALUE, -5222, ER_TRUNCATED_WRONG_VALUE, "22007", "Incorrect value", "Truncated in
correct %.*s value: '%.*s'");
DEFINE_ERROR(OB_ERR_WRONG_IDENT_NAME, -5223, -1, "42000", "wrong ident name");
DEFINE_ERROR_EXT(OB_WRONG_NAME_FOR_INDEX, -5224, ER_WRONG_NAME_FOR_INDEX, "42000", "Incorrect index name", "Incorrect ind
ex name '%.*s'");
DEFINE_ERROR_EXT(OB_ILLEGAL_REFERENCE, -5225, ER_ILLEGAL_REFERENCE, "42S22", "Reference not supported (reference to group
function)", "Reference '%.*s' not supported (reference to group function)")
DEFINE_ERROR(OB_REACH_MEMORY_LIMIT, -5226, -1, "42000", "plan cache memory used reach the high water mark.") ;
DEFINE_ERROR(OB_ERR_PASSWORD_FORMAT, -5227, ER_PASSWORD_FORMAT, "42000", "The password hash doesn't have the expected for
mat. Check if the correct password algorithm is being used with the PASSWORD() function.") ;
DEFINE_ERROR(OB_ERR_NON_UPDATABLE_TABLE, -5228, ER_NON_UPDATABLE_TABLE, "HY000", "The target table tt of the UPDATE is no
t updatable");
DEFINE_ERROR_EXT(OB_ERR_WARN_DATA_OUT_OF_RANGE, -5229, ER_WARN_DATA_OUT_OF_RANGE, "22003", "Out of range value for column
", "Out of range value for column '%.*s' at row %ld")
DEFINE_ERROR(OB_ERR_WRONG_EXPR_IN_PARTITION_FUNC_ERROR, -5230, ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR, "HY000", "Constant
or random or timezone-dependent expressions in (sub)partitioning function are not allowed");
DEFINE_ERROR_EXT(OB_ERR_VIEW_INVALID, -5231, ER_VIEW_INVALID, "42S22", "view invalid", "View \'%.*s. %.*s\' references in
valid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them");
DEFINE_ERROR(OB_ERR_OPTION_PREVENTS_STATEMENT, -5233, ER_OPTION_PREVENTS_STATEMENT, "HY000", "The MySQL server is running
with the --read-only option so it cannot execute this statement");
DEFINE_ERROR(OB_ERR_DB_READ_ONLY, -5234, -1, "HY000", "The database is read only so it cannot execute this statement", "T
he database \'%.*s\' is read only so it cannot execute this statement");
DEFINE_ERROR(OB_ERR_TABLE_READ_ONLY, -5235, -1, "HY000", "The table is read only so it cannot execute this statement", "T
he table \'%.*s. %.*s\' is read only so it cannot execute this statement");
DEFINE_ERROR(OB_ERR_LOCK_OR_ACTIVE_TRANSACTION, -5236, ER_LOCK_OR_ACTIVE_TRANSACTION, "HY000", "Can't execute the given c
ommand because you have active locked tables or an active transaction");
DEFINE_ERROR_EXT(OB_ERR_SAME_NAME_PARTITION_FIELD, -5237, ER_SAME_NAME_PARTITION_FIELD, "HY000", "Duplicate partition fie
ld name", "Duplicate partition field name '%.*s'")
DEFINE_ERROR_EXT(OB_ERR_TABLENAME_NOT_ALLOWED_HERE, -5238, ER_TABLENAME_NOT_ALLOWED_HERE, "42000", "Table from one of the
SELECTs cannot be used in global ORDER clause", "Table \'%.*s\' from one of the SELECTs cannot be used in global ORDER cl
ause");
DEFINE_ERROR_EXT(OB_ERR_VIEW_RECURSIVE, -5239, ER_VIEW_RECURSIVE, "42S22", "view contains recursion", "\'%.*s'.' %.*s\' c
ontains view recursion");
DEFINE_ERROR(OB_ERR_QUALIFIER, -5240, -1, "HY000", "Column part of USING clause cannot have qualifier");
DEFINE_ERROR(OB_ERR_WRONG_VALUE, -5241, ER_WRONG_VALUE, "HY000", "Incorrect %s value: '%s'");
DEFINE_ERROR(OB_ERR_VIEW_WRONG_LIST, -5242, ER_VIEW_WRONG_LIST,  "HY000", "View's SELECT and view's field list have diffe
rent column counts");
DEFINE_ERROR(OB_SYS_VARS_MAYBE_DIFF_VERSION, -5243, -1,  "HY000", "system variables' version maybe different");
DEFINE_ERROR(OB_ERR_AUTO_INCREMENT_CONFLICT, -5244, ER_AUTO_INCREMENT_CONFLICT, "HY000", "Auto-increment value in UPDATE
conflicts with internally generated values");
DEFINE_ERROR_EXT(OB_ERR_TASK_SKIPPED, -5245, -1, "HY000", "some tasks are skipped", "some tasks are skipped, skipped serv
er addr is '%s', the orginal error code is %d");
DEFINE_ERROR_EXT(OB_ERR_NAME_BECOMES_EMPTY, -5246, ER_NAME_BECOMES_EMPTY, "HY000", "Name has become ''", "Name \'%.*s\' h
as become ''");
DEFINE_ERROR_EXT(OB_ERR_REMOVED_SPACES, -5247, ER_REMOVED_SPACES, "HY000", "Leading spaces are removed from name ", "Lead
ing spaces are removed from name \'%.*s\'");
DEFINE_ERROR_EXT(OB_WARN_ADD_AUTOINCREMENT_COLUMN, -5248, -1, "HY000", "Alter table add auto increment column is dangerou
DEFINE_ERROR_EXT(OB_WARN_ADD_AUTOINCREMENT_COLUMN, -5248, -1, "HY000", "Alter table add auto_increment column is dangerou
s", "Alter table add auto_increment column is dangerous, table_name=\'%.*s\', column_name=\'%s\'");
DEFINE_ERROR_EXT(OB_WARN_CHAMGE_NULL_ATTRIBUTE, -5249, -1, "HY000", "Alter table change nullable column to not nullable i
s dangerous", "Alter table change nullable column to not nullable is dangerous, table_name=\'%.*s\', column_name=\'%.*s\'
");
DEFINE_ERROR_EXT(OB_ERR_INVALID_CHARACTER_STRING, -5250, ER_INVALID_CHARACTER_STRING, "HY000", "Invalid character string"
, "Invalid %.*s character string: \'%.*s\'");
DEFINE_ERROR_EXT(OB_ERR_KILL_DENIED, -5251, ER_KILL_DENIED_ERROR, "HY000", "You are not owner of thread", "You are not ow
ner of thread %lu");
DEFINE_ERROR_EXT(OB_ERR_COLUMN_DEFINITION_AMBIGUOUS, -5252, -1, "HY000", "Column definition is ambiguous. Column has both
NULL and NOT NULL attributes", "Column definition is ambiguous. Column has both NULL and NOT NULL attributes");
DEFINE_ERROR(OB_ERR_EMPTY_QUERY, -5253, ER_EMPTY_QUERY, "42000", "Query was empty");
DEFINE_ERROR_EXT(OB_ERR_CUT_VALUE_GROUP_CONCAT, -5254, ER_CUT_VALUE_GROUP_CONCAT, "42000", "Row was cut by GROUP_CONCAT()
", "Row %ld was cut by GROUP_CONCAT()");
DEFINE_ERROR(OB_ERR_FILED_NOT_FOUND_PART, -5255, ER_FIELD_NOT_FOUND_PART_ERROR, "HY000", "Field in list of fields for par
tition function not found in table");
DEFINE_ERROR(OB_ERR_PRIMARY_CANT_HAVE_NULL, -5256, ER_PRIMARY_CANT_HAVE_NULL, "42000", "All parts of a PRIMARY KEY must b
e NOT NULL; if you need NULL in a key, use UNIQUE instead");
DEFINE_ERROR(OB_ERR_PARTITION_FUNC_NOT_ALLOWED_ERROR, -5257, ER_PARTITION_FUNC_NOT_ALLOWED_ERROR, "HY000", "The PARTITION
function returns the wrong type");
DEFINE_ERROR(OB_ERR_INVALID_BLOCK_SIZE, -5258, -1, "HY000", "Invalid block size, block size should between 16384 and 1048
576");
DEFINE_ERROR_EXT(OB_ERR_UNKNOWN_STORAGE_ENGINE, -5259, ER_UNKNOWN_STORAGE_ENGINE, "42000", "Unknown storage engine", "Unk
nown storage engine \'%.*s\'");
DEFINE_ERROR_EXT(OB_ERR_TENANT_IS_LOCKED, -5260, -1, "HY000", "Tenant is locked", "Tenant \'%.*s\' is locked");
DEFINE_ERROR_EXT(OB_EER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF, -5261, ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF, "HY000", "A UNIQUE
INDEX/PRIMARY KEY must include all columns in the table's partitioning function", "A %s must include all columns in the t
able's partitioning function";
DEFINE_ERROR(OB_ERR_PARTITION_FUNCTION_IS_NOT_ALLOWED, -5262, ER_PARTITION_FUNCTION_IS_NOT_ALLOWED, "HY000", "This partit
ion function is not allowed");
DEFINE_ERROR_EXT(OB_ERR_AGGREGATE_ORDER_FOR_UNION, -5263, ER_AGGREGATE_ORDER_FOR_UNION, "HY000", "aggregate order for uni
on", "Expression #%d of ORDER BY contains aggregate function and applies to a UNION");
DEFINE_ERROR_EXT(OB_ERR_OUTLINE_EXIST, -5264, -1, "HY000", "Outline exists", "Outline '%.*s' already exists");
DEFINE_ERROR_EXT(OB_OUTLINE_NOT_EXIST, -5265, -1, "HY000", "Outline not exists", "Outline \'%.*s. %.*s\' doesn\'t exist")
;
DEFINE_ERROR_EXT(OB_WARN_OPTION_BELOW_LIMIT, -5266, WARN_OPTION_BELOW_LIMIT, "HY000", "The value should be no less than t
he limit", "The value of \'%s\' should be no less than the value of \'%s\'");
DEFINE_ERROR_EXT(OB_INVALID_OUTLINE, -5267, -1, "HY000", "invalid outline", "invalid outline ,error info:%s");
DEFINE_ERROR_EXT(OB_REACH_MAX_CONCURRENT_NUM, -5268, -1, "HY000", "SQL reach max concurrent num", "SQL reach max concurre
nt num %ld");
DEFINE_ERROR(OB_ERR_OPERATION_ON_RECYCLE_OBJECT, -5269, -1, "HY000", "can not perform DDL/DML over objects in Recycle Bin
");
DEFINE_ERROR(OB_ERR_OBJECT_NOT_IN_RECYCLEBIN, -5270, -1, "HY000", "object not in RECYCLE BIN");
DEFINE_ERROR(OB_ERR_CON_COUNT_ERROR, -5271, ER_CON_COUNT_ERROR, "08004", "Too many connections");
DEFINE_ERROR_EXT(OB_ERR_OUTLINE_CONTENT_EXIST, -5272, -1, "HY000", "Outline content already exists when added", "Outline
content '%.*s' of outline '%.*s' already exists when added");
DEFINE_ERROR_EXT(OB_ERR_OUTLINE_MAX_CONCURRENT_EXIST, -5273, -1, "HY000", "Max concurrent already exists when added", "Ma
x concurrent in outline '%.*s' already exists when added");
DEFINE_ERROR_EXT(OB_ERR_VALUES_IS_NOT_INT_TYPE_ERROR, -5274, ER_VALUES_IS_NOT_INT_TYPE_ERROR, "HY000", "VALUES value for
partition must have type INT", "VALUES value for partition \'%.*s\' must have type INT");
DEFINE_ERROR(OB_ERR_WRONG_TYPE_COLUMN_VALUE_ERROR, -5275, ER_WRONG_TYPE_COLUMN_VALUE_ERROR, "HY000", "Partition column va
lues of incorrect type");
DEFINE_ERROR(OB_ERR_PARTITION_COLUMN_LIST_ERROR, -5276, ER_PARTITION_COLUMN_LIST_ERROR, "HY000", "Inconsistency in usage
of column lists for partitioning");
DEFINE_ERROR(OB_ERR_TOO_MANY_VALUES_ERROR, -5277, ER_TOO_MANY_VALUES_ERROR, "HY000", "Cannot have more than one value for
this type of RANGE partitioning");
DEFINE_ERROR(OB_ERR_PARTITION_FUNCTION_IS_NOT_ALLOWED, -5278, ER_PARTITION_FUNCTION_IS_NOT_ALLOWED, "HY000", "This partit
ion function is not allowed");
DEFINE_ERROR(OB_ERR_PARTITION_INTERVAL_ERROR, -5279, -1, "HY000", "Partition interval must have type INT");
DEFINE_ERROR_EXT(OB_ERR_SAME_NAME_PARTITION, -5280, ER_SAME_NAME_PARTITION, "HY000", "Duplicate partition name", "Duplica
te partition name \'%.*s\'");
DEFINE_ERROR(OB_ERR_RANGE_NOT_INCREASING_ERROR, -5281, ER_RANGE_NOT_INCREASING_ERROR, "HY000", "VALUES LESS THAN value mu
st be strictly increasing for each partition");
DEFINE_ERROR(OB_ERR_PARSE_PARTITION_RANGE, -5282, ER_PARSE_ERROR, "42000", "Wrong number of partitions defined, mismatch
with previous setting");
DEFINE_ERROR(OB_ERR_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF, -5283, ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF, "HY000", "A PRIMARY KEY
must include all columns in the table\'s partitioning function");
DEFINE_ERROR(OB_NO_PARTITION_FOR_GIVEN_VALUE, -5284, ER_NO_PARTITION_FOR_GIVEN_VALUE, "HY000", "Table has no partition fo
r value");
DEFINE_ERROR(OB_EER_NULL_IN_VALUES_LESS_THAN, -5285, ER_NULL_IN_VALUES_LESS_THAN, "HY000", "Not allowed to use NULL value
in VALUES LESS THAN");
DEFINE_ERROR(OB_ERR_PARTITION_CONST_DOMAIN_ERROR, -5286, ER_PARTITION_CONST_DOMAIN_ERROR, "HY000", "Partition constant is
out of partition function domain");
DEFINE_ERROR(OB_ERR_TOO_MANY_PARTITION_FUNC_FIELDS, -5287, ER_TOO_MANY_PARTITION_FUNC_FIELDS_ERROR, "HY000", "Too many fi
elds in \'list of partition fields\'");
DEFINE_ERROR_EXT(OB_ERR_BAD_FT_COLUMN, -5288, ER_BAD_FT_COLUMN, "HY000", "Column cannot be part of FULLTEXT index", "Colu
mn '%.*s' cannot be part of FULLTEXT index");
DEFINE_ERROR_EXT(OB_ERR_KEY_DOES_NOT_EXISTS, -5289, ER_KEY_DOES_NOT_EXITS, "42000", "key does not exist in table", "Key '
%.*s' doesn't exist in table '%.*s'");
DEFINE_ERROR_EXT(OB_NON_DEFAULT_VALUE_FOR_GENERATED_COLUMN, -5290, ER_NON_DEFAULT_VALUE_FOR_GENERATED_COLUMN, "HY000", "n
on-default value for generated column is not allowed", "The value specified for generated column '%.*s' in table '%.*s' i
s not allowed");
DEFINE_ERROR(OB_ERR_BAD_CTXCAT_COLUMN, -5291, -1, "HY000", "The CTXCAT column must be contiguous in the index column list
");
DEFINE_ERROR_EXT(OB_ERR_UNSUPPORTED_ACTION_ON_GENERATED_COLUMN, -5292, ER_UNSUPPORTED_ACTION_ON_GENERATED_COLUMN, "HY000"
, "not supported for generated columns", "'%s' is not supported for generated columns.") ;
DEFINE_ERROR_EXT(OB_ERR_DEPENDENT_BY_GENERATED_COLUMN, -5293, ER_DEPENDENT_BY_GENERATED_COLUMN, "HY000", "Column has a ge
nerated column dependency", "Column '%.*s' has a generated column dependency");
DEFINE_ERROR(OB_ERR_TOO_MANY_ROWS, -5294, ER_TOO_MANY_ROWS, "42000", "Result consisted of more than one row");
DEFINE_ERROR(OB_WRONG_FIELD_TERMINATORS, -5295, ER_WRONG_FIELD_TERMINATORS, "42000", "Field separator argument is not wha
t is expected; check the manual");
DEFINE_ERROR(OB_NO_READABLE_REPLICA, -5296, -1, "42000", "there has no readable replica");
DEFINE_ERROR(OB_ERR_PARTITION_MGMT_ON_NONPARTITIONED, -5302, ER_PARTITION_MGMT_ON_NONPARTITIONED, "HY000", "Partition man
agement on a not partitioned table is not possible");
DEFINE_ERROR_EXT(OB_ERR_DROP_PARTITION_NON_EXISTENT, -5303, ER_DROP_PARTITION_NON_EXISTENT, "HY000", "Error in list of pa
rtitions", "Error in list of partitions to %s");
DEFINE_ERROR(OB_ERR_PARTITION_MGMT_ON_TWOPART_TABLE, -5304, -1, "HY000", "Partition management on a two-part table is not
possible");
DEFINE_ERROR(OB_ERR_ONLY_ON_RANGE_LIST_PARTITION, -5305, ER_ONLY_ON_RANGE_LIST_PARTITION, "HY000", "can only be used on R
ANGE/LIST partitions", "%s PARTITION can only be used on RANGE/LIST partitions");
DEFINE_ERROR(OB_ERR_DROP_LAST_PARTITION, -5306, ER_DROP_LAST_PARTITION, "HY000", "Cannot remove all partitions, use DROP
TABLE instead");
////////////////////////////////////////////////////////////////
//error code for transaction, mvcc and commitlog -6001 ---- -7000
////////////////////////////////////////////////////////////////
DEFINE_ERROR(OB_TRANSACTION_SET_VIOLATION, -6001, -1, "25000", "Transaction set changed during the execution");
DEFINE_ERROR_EXT(OB_TRANS_ROLLBACKED, -6002, -1, "40000", "Transaction rollbacked", "transaction is rolled back");
DEFINE_ERROR(OB_ERR_EXCLUSIVE_LOCK_CONFLICT, -6003, ER_LOCK_WAIT_TIMEOUT, "HY000", "Lock wait timeout exceeded; try resta
rting transaction");
DEFINE_ERROR(OB_ERR_SHARED_LOCK_CONFLICT, -6004, -1, "HY000", "Shared lock conflict");
DEFINE_ERROR(OB_TRY_LOCK_ROW_CONFLICT, -6005, -1, "HY000", "Try lock row conflict");
DEFINE_ERROR(OB_CLOCK_OUT_OF_ORDER, -6201, -1, "25000", "Clock out of order");
DEFINE_ERROR(OB_MASK_SET_NO_NODE, -6203, -1, "25000", "Mask set has no node");
DEFINE_ERROR(OB_TRANS_HAS_DECIDED, -6204, -1, "HY000", "Transaction has been decided");
DEFINE_ERROR(OB_TRANS_INVALID_STATE, -6205, -1, "HY000", "Transaction state invalid");
DEFINE_ERROR(OB_TRANS_STATE_NOT_CHANGE, -6206, -1, "HY000", "Transaction state not changed");
DEFINE_ERROR(OB_TRANS_PROTOCOL_ERROR, -6207, -1, "HY000", "Transaction protocol error");
DEFINE_ERROR(OB_TRANS_INVALID_MESSAGE, -6208, -1, "HY000", "Transaction message invalid");
DEFINE_ERROR(OB_TRANS_INVALID_MESSAGE_TYPE, -6209, -1, "HY000", "Transaction message type invalid");
DEFINE_ERROR(OB_TRANS_TIMEOUT, -6210, 4012, "25000", "Transaction is timeout");
DEFINE_ERROR(OB_TRANS_KILLED, -6211, 6002, "25000", "Transaction is killed");
DEFINE_ERROR(OB_TRANS_STMT_TIMEOUT, -6212, 4012, "25000", "Statement is timeout");
DEFINE_ERROR(OB_TRANS_CTX_NOT_EXIST, -6213, 6002, "HY000", "Transaction context does not exist");
DEFINE_ERROR(OB_PARTITION_IS_FROZEN, -6214, 6002, "25000", "Partition is frozen");
DEFINE_ERROR(OB_PARTITION_IS_NOT_FROZEN, -6215, -1, "HY000", "Partition is not frozen");
DEFINE_ERROR(OB_TRANS_INVALID_LOG_TYPE, -6219, -1, "HY000", "Transaction invalid log type");
DEFINE_ERROR(OB_TRANS_SQL_SEQUENCE_ILLEGAL, -6220, -1, "HY000", "SQL sequence illegal");
DEFINE_ERROR(OB_TRANS_CANNOT_BE_KILLED, -6221, -1, "HY000", "Transaction context cannot be killed");
DEFINE_ERROR(OB_TRANS_STATE_UNKNOWN, -6222, -1, "HY000", "Transaction state unknown");
DEFINE_ERROR(OB_TRANS_IS_EXITING, -6223, 6002, "25000", "Transaction exiting");
DEFINE_ERROR(OB_TRANS_NEED_ROLLBACK, -6224, 6002, "25000", "transaction need rollback");
DEFINE_ERROR(OB_TRANS_UNKNOWN, -6225, 4012, "25000", "Transaction result is unknown");
DEFINE_ERROR(OB_ERR_READ_ONLY_TRANSACTION, -6226, ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION, "25006", "Cannot execute stat
ement in a READ ONLY transaction");
DEFINE_ERROR(OB_PARTITION_IS_NOT_STOPPED, -6227, -1, "HY000", "Partition is not stopped");
DEFINE_ERROR(OB_PARTITION_IS_STOPPED, -6228, -1, "HY000", "Partition has been stopped");
DEFINE_ERROR(OB_PARTITION_IS_BLOCKED, -6229, -1, "HY000", "Partition has been blocked");
DEFINE_ERROR(OB_TRANS_RPC_TIMEOUT, -6230, 4012, "25000", "transaction rpc timeout");
DEFINE_ERROR(OB_REPLICA_NOT_READABLE, -6231, -1, "HY000", "replica is not readable");
// for clog
DEFINE_ERROR(OB_LOG_ID_NOT_FOUND, -6301, -1, "HY000", "log id not found");
DEFINE_ERROR(OB_LSR_THREAD_STOPPED, -6302, -1, "HY000", "log scan runnable thread stop");
DEFINE_ERROR(OB_NO_LOG, -6303, -1, "HY000", "no log ever scanned");
DEFINE_ERROR(OB_LOG_ID_RANGE_ERROR, -6304, -1, "HY000", "log id range error");
DEFINE_ERROR(OB_LOG_ITER_ENOUGH, -6305, -1, "HY000", "iter scans enough files");
DEFINE_ERROR(OB_CLOG_INVALID_ACK, -6306, -1, "HY000", "invalid ack msg");
DEFINE_ERROR(OB_CLOG_CACHE_INVALID, -6307, -1, "HY000", "clog cache invalid");
DEFINE_ERROR(OB_EXT_HANDLE_UNFINISH, -6308, -1, "HY000", "external executor handle do not finish");
DEFINE_ERROR(OB_CURSOR_NOT_EXIST, -6309, -1, "HY000", "cursor not exist");
DEFINE_ERROR(OB_STREAM_NOT_EXIST, -6310, -1, "HY000", "stream not exist");
DEFINE_ERROR(OB_STREAM_BUSY, -6311, -1, "HY000", "stream busy");
DEFINE_ERROR(OB_FILE_RECYCLED, -6312, -1, "HY000", "file recycled");
DEFINE_ERROR(OB_REPLAY_EAGAIN_TOO_MUCH_TIME, -6313, -1, "HY000", "replay eagain cost too much time");
DEFINE_ERROR(OB_MEMBER_CHANGE_FAILED, -6314, -1, "HY000", "member change log sync failed");
////////////////////////////////////////////////////////////////
//error code for election -7000 ---- -7100
////////////////////////////////////////////////////////////////
DEFINE_ERROR(OB_ELECTION_WARN_LOGBUF_FULL, -7000, -1, "HY000", "The log buffer is full");
DEFINE_ERROR(OB_ELECTION_WARN_LOGBUF_EMPTY, -7001, -1, "HY000", "The log buffer is empty");
DEFINE_ERROR(OB_ELECTION_WARN_NOT_RUNNING, -7002, -1, "HY000", "The object is not running");
DEFINE_ERROR(OB_ELECTION_WARN_IS_RUNNING, -7003, -1, "HY000", "The object is running");
DEFINE_ERROR(OB_ELECTION_WARN_NOT_REACH_MAJORITY, -7004, -1, "HY000", "Election does not reach majority");
DEFINE_ERROR(OB_ELECTION_WARN_INVALID_SERVER, -7005, -1, "HY000", "The server is not valid");
DEFINE_ERROR(OB_ELECTION_WARN_INVALID_LEADER, -7006, -1, "HY000", "The leader is not valid");
DEFINE_ERROR(OB_ELECTION_WARN_LEADER_LEASE_EXPIRED, -7007, -1, "HY000", "The leader lease is expired");
DEFINE_ERROR(OB_ELECTION_WARN_INVALID_MESSAGE, -7010, -1, "HY000", "The message is not valid");
DEFINE_ERROR(OB_ELECTION_WARN_MESSAGE_NOT_INTIME, -7011, -1, "HY000", "The message is not intime");
DEFINE_ERROR(OB_ELECTION_WARN_NOT_CANDIDATE, -7012, -1, "HY000", "The server is not candidate");
DEFINE_ERROR(OB_ELECTION_WARN_NOT_CANDIDATE_OR_VOTER, -7013, -1, "HY000", "The server is not candidate or voter");
DEFINE_ERROR(OB_ELECTION_WARN_PROTOCOL_ERROR, -7014, -1, "HY000", "Election protocol error");
DEFINE_ERROR(OB_ELECTION_WARN_RUNTIME_OUT_OF_RANGE, -7015, -1, "HY000", "The task run time out of range");
DEFINE_ERROR(OB_ELECTION_WARN_LAST_OPERATION_NOT_DONE, -7021, -1, "HY000", "Last operation has not done");
DEFINE_ERROR(OB_ELECTION_WARN_CURRENT_SERVER_NOT_LEADER, -7022, -1, "HY000", "Current server is not leader");
DEFINE_ERROR(OB_ELECTION_WARN_NO_PREPARE_MESSAGE, -7024, -1, "HY000", "There is not prepare message");
DEFINE_ERROR(OB_ELECTION_ERROR_MULTI_PREPARE_MESSAGE, -7025, -1, "HY000", "There is more than one prepare message");
DEFINE_ERROR(OB_ELECTION_NOT_EXIST, -7026, -1, "HY000", "Election does not exist");
DEFINE_ERROR(OB_ELECTION_MGR_IS_RUNNING, -7027, -1, "HY000", "Election manager is running");
DEFINE_ERROR(OB_ELECTION_WARN_NO_MAJORITY_PREPARE_MESSAGE, -7029, -1, "HY000", "Election msg pool not have majority prepa
re message");
DEFINE_ERROR(OB_ELECTION_ASYNC_LOG_WARN_INIT, -7030, -1, "HY000", "Election async log init error");
DEFINE_ERROR(OB_ELECTION_WAIT_LEADER_MESSAGE, -7031, -1, "HY000", "Election waiting leader message");
////////////////////////////////////////////////////////////////
// !!! Fatal errors and the client should close the connection, -8000 ~ -8999
////////////////////////////////////////////////////////////////
DEFINE_ERROR(OB_SERVER_IS_INIT, -8001, -1, "08004", "Server is initializing");
DEFINE_ERROR(OB_SERVER_IS_STOPPING, -8002, -1, "08004", "Server is stopping");
DEFINE_ERROR(OB_PACKET_CHECKSUM_ERROR, -8003, -1, "08004", "Packet checksum error");
DEFINE_ERROR(OB_PACKET_CLUSTER_ID_NOT_MATCH, -8004, -1, "08004", "Packet cluster_id not match");
```

5.24. Logs

Log levels

The following log levels in ApsaraDB for OceanBase are listed based on severity in descending order: ERROR, USER_ERROR, WARN, INFO, TRACE, and DEBUG.

describes each of the log levels.

Log levels and descriptions

Log level	Description
ERROR	The unexpected errors that must be manually handled.
USER_ERROR	The errors that are caused by invalid input data.
WARN	The expected errors that can be resolved by applications.
INFO	The messages that describe the changes of the system status.

Log level	Description
TRACE	The fine-grained debugging messages that allow you to trace the details of each request. For example, the execution of an SQL statement is divided into different stages and a trace log entry is generated for each stage.
DEBUG	The detailed debugging messages.

Log print settings

Logs are displayed by program module. Program modules consist of parent modules and child modules.

Examples:

- SQL_OPT_LOG records the information about the *SQL.OPT* child module. The SQL_OPT_LOG log entries are generated based on the log level settings of the *SQL.OPT* child module.
- SQL_LOG records the information about the SQL module. The SQL_LOG log entries are generated based on the log level settings of the SQL module.
- OB_LOG records the information about all the modules. The OB_LOG log entries are generated based on global settings of log levels.

Program logs divided into statement, session, and system logs.

Program logs are generated based on priorities in the following descending order: statement logs, session logs, and system logs.

Use the following methods to specify log print settings:

- System logs
  - Method 1: Execute the `ALTER SYSTEM SET OB_LOG_LEVEL` statement

    Execute the `ALTER SYSTEM SET OB_LOG_LEVEL` statement to change the log level of system logs.

    Example:

    ```
    alter system set ob_log_level='sql.*:debug, common.*:error';
    ```

    `alter SYSTEM SET` allows you to specify the server or zone. For more information, see the `ALTER SYSTEM SET` syntax.
  - Method 2: Run the `kill -41` and kill -42 commands

    You can run the `kill -41` command to increase the log level. For example, if the previous log level is INFO, you can run the `kill -41` command to change the log level to TRACE.

    `kill You can run the kill -42` command to decrease the log level. For example, if the previous log level is INFO, you can run the `kill -42` command to change the log level to WARN.

    `kill You can run the kill -41` command to increase the log level to the highest level: DEBUG. You can run the kill -42 command to decrease the log level to the lowest level: ERROR.

  We recommend that you use the first method. If the server updates the configuration file, the ob_log_level configurations in the file are imported. In this scenario, the kill commands fail to be run.
- Session logs

  Run the `set @@ob_log_level='par_mod.sub_mod:level, par_mod.sub_mod:level'` command.

  Example:

  ```
  set @@ob_log_level='sql.*:debug, common.*:info';
  ```
- Statement logs

  Use hints to set the log levels of statement logs.

  Example:

  ```
  select /*+log_level('debug')*/ * from t;
  select /*+log_level('sql.*:debug, common.*:info')*/ * from t;
  ```

  To view the log details in an easy way when you debug OBServers, you can execute the `ALTER SYSTEM SET ob_log_level ='error'` statement to set the log level to `error`. You can run the `set @ @ ob_log_level='debug'` command to specify the log level of the session log as `debug`. If you execute a statement to retrieve data from multiple OBServers, the global settings of log levels apply to each of the OBServers.