FROM INTRODUCTION TO PRACTICE

Lesson 6: Using OceanBase Community Edition for Business Development

Peng Wang
OceanBase Global Technical Evangelist

Agenda

- Lesson 6.1
  - Using MySQL tenants for common database development
- Lesson 6.2
  - Developed through ODC graphical development tools
  - Horizontal Split Using OceanBase Partition Table

Using Mysql Tenants for Common Database Development

Connection Method Introduction

Client

- Command line client： MySQL、OBClient
- Graphical interface tools：ODC、DBeaver、Navicat etc.
- Connection pool configuration introduction

Driver

Support multiple programming language drivers:

- Java Driver (MySQL Connector/J)
- C Driver (OceanBase Connector/C)
- Python Driver (PyMySQL)
- Go Driver (Go-SQL-Driver)

ORM Framework

Object Relational Mapping (ORM for short), is a "virtual object database" that can be used in programming languages. For example: SpringBoot、SpringBatch、SpringJDBC、SpringJPA、Hibernate、MyBatis

Using Mysql Tenants for Common Database Development

Connection Method Introduction - MySQL

Prerequisites

"mysql" is a command-line client for the MySQL database and needs to be installed separately. OceanBase Community Edition only supports MySQL mode tenants.

- Make sure that the MySQL client is correctly installed locally. The current version of OceanBase database supports MySQL client versions including V5.5, V5.6, and V5.7.
- Make sure the PATH environment variable contains the directory where the MySQL client command is located.
- Before connecting to a tenant, please confirm that the current client is in the tenant whitelist. For details on how to query and set the tenant whitelist, see View and Set the Tenant Whitelist.

Connection Operation

1.Open a command line terminal.
2.Connect to the MySQL tenant using the MySQL command

- Connection via ODP
  - $mysql -h10.10.10.1 -**uusername@obmysql#obdemo** -P2883 -p****** -c -A -Doceanbase
  - $mysql -h10.10.10.1 -**uobdemo:obmysql:username** -P2883 -p****** -c -A -Doceanbase
- Directly connect to OBServer
  - $mysql -h10.10.10.1 -uusername@obmysql -P2881 -p****** -c -A -Doceanbase

Connection Parameters

- -h: Provide the OceanBase database connection IP, usually the IP address of an OBServer node
- -u: Provide the tenant's connection account in the format of user_name@tenant_name. Using the MySQL client only supports connecting to MySQL tenants. The default administrator user name of the MySQL tenant is root
- -P: Provides the OceanBase database connection port, the default is 2881, which can be customized when deploying the OceanBase database
- -p: Provide the account password. For security reasons, you do not need to provide it. Instead, enter it in the following prompt. The password text is not visible
- -c: Indicates that comments should not be ignored in the MySQL runtime environment. Hint is a special comment and is not affected by -c
- -A: Indicates that statistics are not automatically obtained when MySQL connects to the database
- -D: The name of the database being accessed, which can be changed to a business database
- When a common tenant connects via direct connection, it is necessary to ensure that the tenant's resources are distributed on the OBServer node. If the tenant's resources are not distributed on the OBServer node, it is not possible to connect to the tenant via direct connection to the OBServer node

Using Mysql Tenants for Common Database Development

Connection Method Introduction - OBClient

Prerequisites

OBClient is a standalone interactive and batch query tool that requires separate installation. It provides a command-line interface for connecting to OceanBase databases, serving as a client for both MySQL and Oracle tenants. After connecting to the OceanBase database, you can run some database commands (including common MySQL commands), SQL statements, and PL statements through OBClient

- Please make sure that you have downloaded and installed the OBClient application. If you have not downloaded the OBClient application, you can visit the Software Center to download the corresponding version of OBClient.
- Before connecting to a tenant, please confirm that the current client is in the tenant whitelist. For details on how to query and set the tenant whitelist, see View and Set the Tenant Whitelist.

Connection Operation

1.Open a command line terminal.
2.Use the OBClient command to connect to the MySQL tenant

- Connection via ODP
  - $obclient -h10.10.10.1 -**uusername@obmysql#obdemo** -P2883 -p****** -c -A -Doceanbase
  - $obclient -h10.10.10.1 -**uobdemo:obmysql:username** -P2883 -p****** -c -A -Doceanbase

- Directly connect to OBServer
  - $obclient -h10.10.10.1 -uusername@obmysql -P2881 -p****** -c -A -Doceanbase

Connection Parameters

•-h: Provide the OceanBase database connection IP, usually the IP address of an OBServer node.
•-u: Provide the tenant's connection account, in the format of user_name@tenant_name. Using the MySQL client only supports connecting to MySQL tenants, and the default administrator username of the MySQL tenant is root.
•-P: Provide the OceanBase database connection port, which is 2881 by default and can be customized when deploying the OceanBase database.
•-p: Provide the account password. For security reasons, you can not provide it and enter it in the following prompt instead. The password text is not visible.
•-c: Indicates not to ignore comments in the MySQL operating environment. Hint is a special comment and is not affected by -c.
•-A: Indicates not to automatically obtain statistical information when MySQL connects to the database.
•-D: The name of the database to be accessed, which can be changed to the business database
•When a common tenant connects via direct connection, it is necessary to ensure that the tenant's resources are distributed on the OBServer node. If the tenant's resources are not distributed on the OBServer node, it is not possible to connect to the tenant via direct connection to the OBServer node.

Using Mysql Tenants for Common Database Development

Connection Method Introduction - Command Line Connection Example

```
$mysql -h10.10.10.1 -u******@obmysql#obdemo -P2883 -p****** -c -A oceanbase
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 5751
Server version: 5.6.25 OceanBase 4.0.0 (r10100032022041510-a09d3134c10665f03fd56d7f8bdd413b2b771977)
(Built Apr 15 2022 02:16:22)
<...>
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| oceanbase          |
| information_schema |
| mysql              |
| test               |
+--------------------+
4 rows in set

exit
Bye
```

Using Mysql Tenants for Common Database Development

Connection Method Introduction - Common Command Line Connection Errors

-- Business database connection error
$mysql -h10.10.10.1 -u******@obmysql#obdemo -P2883 -p****** -c -A -Doceanbaseerror
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1049 (42000): Unknown database 'oceanbaseerror'

-- Wrong port
$mysql -h10.10.10.1 -u******@obmysql#obdemo -P2889 -p****** -c -A -Doceanbase
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 2003 (HY000): Can't connect to MySQL server on '10.10.10.1' (111)

-- The cluster name is incorrect
$mysql -h10.10.10.1 -u******@obmysql#obdemoerror -P2883 -p****** -c -A -Doceanbase
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 4669 (HY000): cluster not exist

-- Wrong tenant
$mysql -h10.10.10.1 -u******@obmysqlerror#obdemo -P2883 -p****** -c -A -Doceanbase
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 4012 (HY000): Get Location Cache Fail

-- Wrong password
$mysql -h10.10.10.1 -u******@obmysql#obdemo -P2883 -p******error -c -A -Doceanbase
ERROR 1045 (42000): Access denied for user 'root'@'xxx.xxx.xxx.xxx' (using password: YES)

-- Directly connect to OBServer but fill in the cluster name
$mysql -h10.10.10.1 -u******@obmysql#obdemo -P2881 -p******error -c -A -Doceanbase
ERROR 1045 (42000): Access denied for user 'root'@'xxx.xxx.xxx.xxx' (using password: YES)

Using Mysql Tenants for Common Database Development

Connection Method Introduction - Graphical Tools

• ODC

- Introduction:

OceanBase Developer Center (ODC) is an enterprise-level database development platform tailored for the OceanBase database. ODC supports MySQL tenants and MySQL databases connected to the OceanBase database, and provides database developers with functions such as daily database development operations, WebSQL, SQL diagnosis, session management, and data import and export.

- Prerequisites:

Ensure that the OceanBase Developer Center (ODC) has been deployed. For detailed deployment operations, please refer to ODC deployment

- Connection Example:

For specific operation examples of using the OceanBase Developer Center to connect to OceanBase tenants, see Connecting to the OceanBase Database through ODC.

- Precautions:

The cluster name in the ODC connection data source page is optional. If you are directly connected to OBServer, you cannot fill in the cluster name, otherwise you will not be able to connect.

• DBeaver

- Introduction:

DBeaver is a general database client tool that can be used for OceanBasse database development and debugging, database management and maintenance, etc.

- Prerequisites:

1. Make sure you have obtained and installed DBeaver. You can visit the official download address of DBeaver to download the DBeaver corresponding to your system.
2. The default driver is mysql-connector-java-5.1.44. You can also visit the official MySQL JDBC driver download page to download other 5.x version driver files.
3. Make sure that the OBServer node IP to be connected is connected to the machine where DBeaver is located

- Connection Example:

For detailed operation examples of using DBeaver to connect to OceanBase tenants, see Connecting to the database through DBeaver

- Precautions:

The current OceanBase MySQL mode does not support event events. Some versions may prompt event-related errors when connecting to the database. You can ignore them directly.

• Navicat

- Introduction:

Navicat is a universal database client tool. Its principle is to use the JDBC driver provided by each database to connect to the database. It supports common relational databases, non-relational databases, distributed databases, etc. You can use the OceanBase driver or MySQL driver that comes with Navicat to connect to the MySQL tenant of the OceanBase database.

- Prerequisites:

1、 Please make sure you have obtained and installed Navicat. You can visit the official download address of Navicat Premium to download the corresponding Navicat for your system.
2、 Please ensure that the OBServer node IP to be connected is connected to the machine where Navicat is located

- Connection Example:

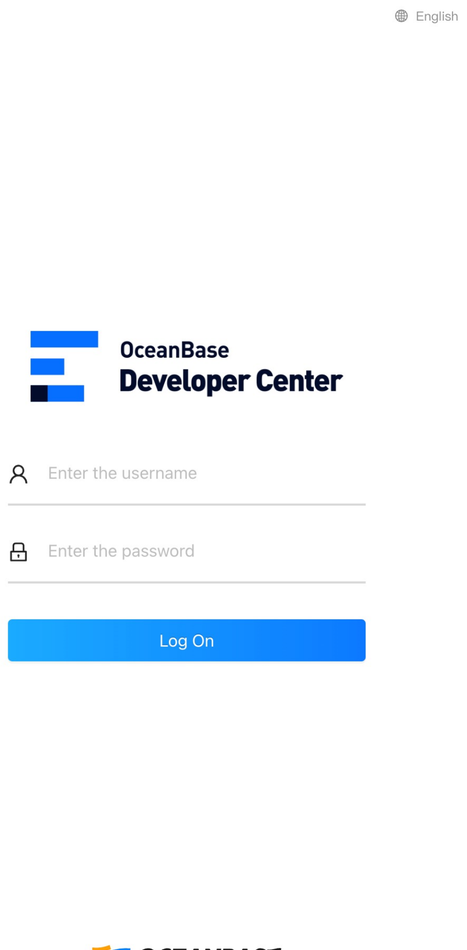
For detailed operation examples of using Navicat to connect to OceanBase tenants, see Connecting to the Database via Navicat

- Precautions:

The current OceanBase MySQL mode does not support event. Some versions may prompt event-related errors when connecting to the database. You can ignore them directly.

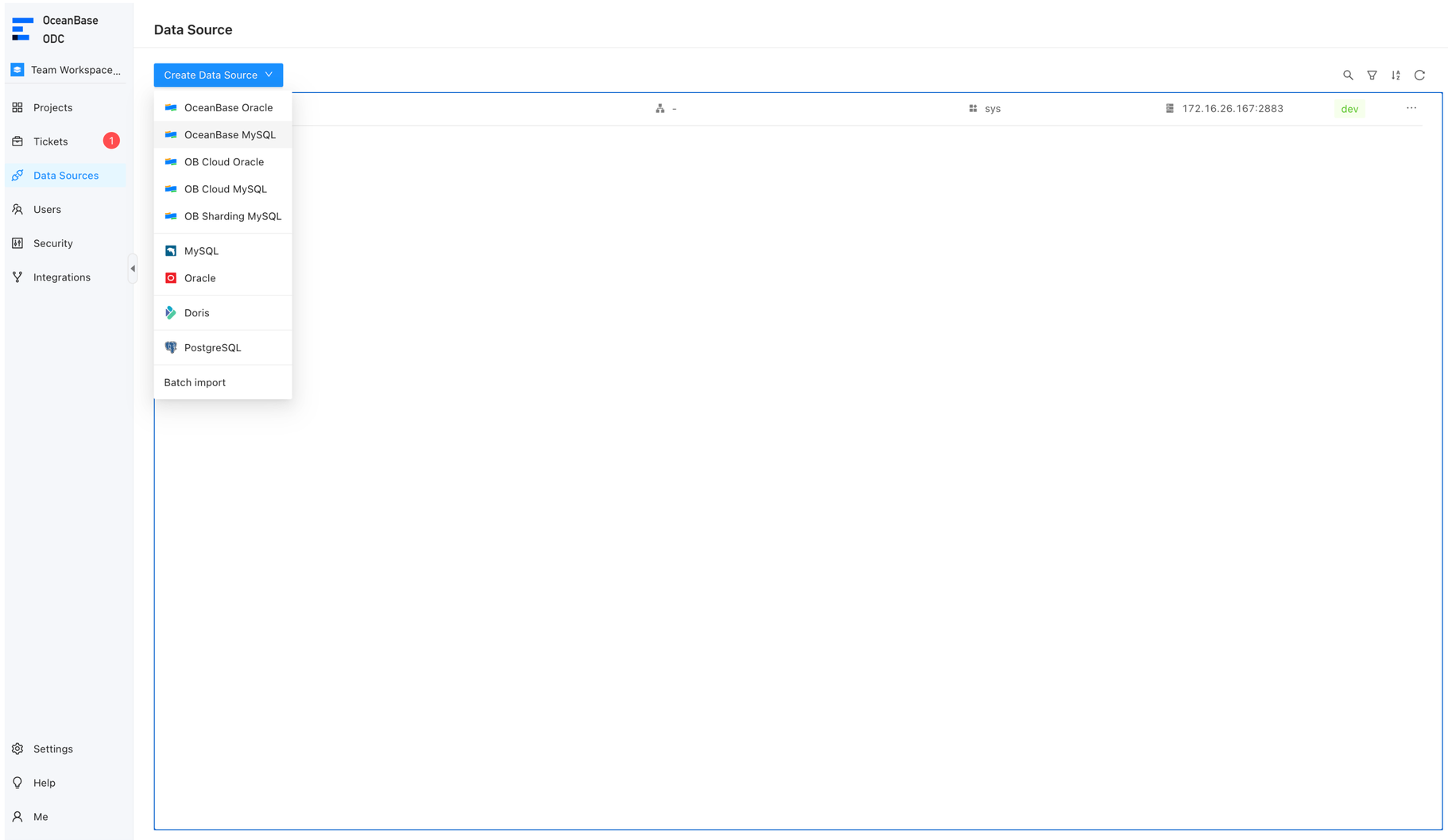
Using Mysql Tenants for Common Database Development

Connection Method Introduction - ODC Connection



Using Mysql Tenants for Common Database Development

Connection Method Introduction - ODC Connection



Using Mysql Tenants for Common Database Development

Connection Method Introduction - ODC Connection

Using Mysql Tenants for Common Database Development

Connection Method Introduction - Connection Pool - Data Source Configuration Recommendations

Parameter Meaning	illustrate	ZDAL Parameters	Druid Parameters	DBCP Parameters	C3P0 Parameters
Initialize the number of connections	The number of connections established when the connection pool is initialized	prefill=true initialize to minConn	initialSize(0)	initialSize(0)	initialPoolSize(3)
Minimum number of connections	The minimum number of available connections. This will allow the connection pool to retain this many connections on a daily basis	minConn(0)	minIdle(0)	minIdle(0)	minPoolSize(3)
Maximum number of connections	The maximum number of connections that can be used. If the number of connections exceeds this value, a connection pool full exception will be thrown	maxConn(10)	maxActive(8)	maxActive(8)	maxActive(8)
Link idle timeout	Set the idle time of the connection. When the connection is not used for a period of time, the connection pool will automatically disconnect the connection. MySQL disconnects the connection after 8 hours by default, and the connection becomes a dirty connection when the master-slave switch is made. If this mechanism is not available, the request may fail. The timeout of OceanBase SLB on the cloud is 15 minutes, and the timeout can be set to 12 minutes.	idleTimeoutMinutes(30min)	minEvictableIdleTimeMillis(30min)	minEvictableIdleTimeMillis(30min) needs to be set to timeBetweenEvictionRunsMillis(-1) > 0 to take effect. This parameter controls the asynchronous check cycle	maxIdleTime (0 means no timeout)
The timeout for the connection pool to obtain a link	If the value is set too high, the connection pool will be full and the application will respond slowly.	blockingTimeoutMillis(500ms)	maxWait (-1 means no timeout)	maxWaitMillis (-1 means no timeout)	checkoutTimeout (0 means no timeout)
Connection acquisition and non-release timeout	If the timeout period is exceeded, the connection will be destroyed directly if it is not returned to the connection pool. This can prevent connection leaks, but it will affect the transaction usage time limit.	N/A	removeAbandonedTimeoutMillis(300s)	removeAbandonedTimeout(300s)	N/A

Using Mysql Tenants for Common Database Development

Connection Method Introduction - JDBC Connection Pool Configuration Example

Connection Example

conn=jdbc:oceanbase://xxx.xxx.xxx.xxx:3306/test?rewriteBatchedStatements=TRUE&allowMultiQueries=TRUE&useLocalSessionState=TRUE&useUnicode=TRUE&characterEncoding=utf-8&socketTimeout=3000000&connectTimeout=60000

Parameter	Parameter Description
rewriteBatchedStatements	The recommended setting is TRUE. • By default, the JDBC driver of OceanBase will ignore the executeBatch() statement, split up a group of SQL statements to be executed in batches, and send them to the database one by one. In this case, batch inserts are actually single inserts, which directly results in lower performance. To perform batch inserts, you need to set this parameter to TRUE, and the driver will execute SQL in batches. That is, use the addBatch method to combine multiple insert statements on the same table into multiple values in one insert statement to improve the performance of batch inserts. • You must use prepareStatement to prepare each insert, and then addBatch, otherwise the execution cannot be merged
allowMultiQueries	It is recommended to set it to TRUE. The JDBC driver allows application code to concatenate multiple SQL statements with semicolons (;) and send them to the server as one SQL statement.
useLocalSessionState	It is recommended to set it to TRUE to avoid frequent transactions sending session variable query SQL to the OB database. The main session variables are autocommit, read_only, and transaction isolation
socketTimeout	When executing SQL, the time the socket waits for SQL to return, in milliseconds. When the value is 0, it means there is no timeout limit. You can also set the system variable max_statement_time to limit the query time. Default value: 0 (standard configuration) or 10000 ms
connectTimeout	The time to wait for a connection when establishing a connection, in milliseconds. A value of 0 means there is no timeout limit. Default value: 30000.
useCursorFetch	It is recommended to set it to TRUE. For large data query statements, the database server will create a Cursor and distribute data to the Client based on the size of FetchSize. If this property is set to TRUE, useServerPrepStms=TRUE will be automatically set.
useServerPrepStms	Controls whether to use PS protocol to send SQL to the database server. When set to TRUE, SQL will be executed in two steps in the database: 1. Send the SQL text containing ? to the database server for Prepare (SQL_audit: request_type=5); 2. Execute in the database with the real value (SQL_audit: request_type=6)
cachePrepStmts	Controls whether the JDBC driver enables PS cache to cache PreparedStatment to avoid repeated prepare operations (client and server). cachePrepStmts=TRUE is helpful for scenarios where useServerPrepStms=TRUE is used and batch execute is repeated for the same SQL. Each batch execute will contain prepare and executecachePrepStmts=TRUE to avoid repeated prepare operations
prepStmtCacheSQLLimit	The length limit of SQL that can be put into PS cache. SQL that is too long cannot be put into cache. prepStmtCacheSize: The number of SQL that can be stored in PS cache
maxBatchTotalParamsNum	For batch operations, a SQL statement can support a maximum of several parameters (i.e. the number of ? in the batch). If the number of parameters exceeds the limit, the batch SQL statement will be split

PS: For other connection pool configuration examples, see Database Connection Pool Configuration, Tomcat, C3P0, Proxool, HikariCP, DBCP, Commons Pool, Druid

Using Mysql Tenants for Common Database Development

Connection Method Introduction - Driver

When connecting to the MySQL tenant of the OceanBase database, the supported drivers are as follows:

- Java Driver（MySQL Connector/J）

MySQL Connector/J is the JDBC driver officially provided by MySQL.
For detailed operation examples of using Java applications to connect to the OceanBase database, see Creating a Java Sample Application.

- C Driver（OceanBase Connector/C）

OceanBase Connector/C is a C/C++ based OceanBase client development component that supports the C API Lib library.
OceanBase Connector/C allows C/C++ programs to access the OceanBase distributed database cluster in a low-level way to perform operations such as database connection, data access, error handling, and Prepared Statement processing.
OceanBase Connector/C is also called LibOBClient. It is used by applications as independent server processes to communicate with the database server OBServer node through a network connection. The client program will reference the C API header file when compiled and can connect to the C API library file. The so file generated by LibOBClient is libobclient.so (corresponding to libmysqlclient.so of MySQL). The command line tool after OceanBase database is installed is OBClient (corresponding to the command line tool of MySQL)
For a detailed example of how to use a C application to connect to the OceanBase database, see Creating a C Sample Application.

- Python Driver（PyMySQL）

PyMySQL is a library for connecting to MySQL servers in Python 3.x. It complies with the Python Database API v2.0 specification and includes a pure-Python MySQL client library. In the MySQL mode of the OceanBase database, users can use the PyMySQL driver to provide Python applications with connections to the OceanBase database.
For detailed operation examples of using Python applications to connect to the OceanBase database, see Creating Python Sample Applications.

- Go Driver（Go-SQL-Driver）

Go-SQL-Driver is a MySQL database driver for Go language. It implements Go's database/sql/driver interface. In the MySQL mode of OceanBase database, developers can operate OceanBase database through the API of database/sql package in Go application.
For a detailed example of how to use a Go application to connect to the OceanBase database, see Creating a Go Sample Application.

Using Mysql Tenants for Common Database Development

Connection Method Introduction - JAVA Sample Application

Coding:

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Test {
   public static void main(String[] args) {
       try {

           Class.forName("com.mysql.jdbc.Driver").newInstance();

           try{

               Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:2881/test?user=r***&password=");
               System.out.println(connection.getAutoCommit());
               Statement sm = connection.createStatement();
               //Create a new table t_meta_form
               sm.executeUpdate("CREATE TABLE t_meta_form (name varchar(36) , id int)");
               //Insert data
               sm.executeUpdate("insert into t_meta_form values ('an','1')");
               //Query data and output results
               ResultSet rs = sm.executeQuery("select * from t_meta_form");
               while (rs.next()) {
                   String name = rs.getString("name");
                   String id = rs.getString("id");
                   System.out.println(name + ','+ id);
               }
               //delete table
               sm.executeUpdate("drop table t_meta_form");

           }catch(SQLException se){
               System.out.println("error!");
               se.printStackTrace() ;
           }
           }catch (Exception ex) {
               ex.printStackTrace();
       }
   }
}
```

Code Compilation:

```
#Configure temporary environment configuration, fill in according to the actual installation path of mysql-connector-java-5.1.47.jar
export CLASSPATH=/usr/share/java/mysql-connector-java-5.1.47.jar:$CLASSPATH
#compile
javac Test.java
```

Code Run:

```
java test
#The following output shows that the database connection is successful and the sample statement is executed correctly
true
an,1
```

Using Mysql Tenants for Common Database Development

Introduction to Connection Methods - ORM Framework

ORM Framework

ORM Object Relational Mapping (ORM for short) is a programming technology that implements data conversion between different types of systems in object-oriented programming languages. It creates a "virtual object database" that can be used in programming languages.

When connecting to the MySQL tenant of the OceanBase database, the supported ORM frameworks are as follows:

For specific operation examples of SpringBoot connecting to OceanBase, see SpringBoot connecting to OceanBase.
For specific operation examples of SpringBatch connecting to OceanBase, see SpringBatch connecting to OceanBase.
For specific operation examples of SpringJDBC connecting to OceanBase, see SpringJDBC connecting to OceanBase.
For specific operation examples of SpringJPA connecting to OceanBase, see SpringJPA connecting to OceanBase.
For detailed operation examples of Hibernate connecting to OceanBase, see Hibernate connecting to OceanBase.
For specific operation examples of MyBatis connecting to OceanBase, see MyBatis connecting to OceanBase.

Using Mysql Tenants for Common Database Development

Database Operations - Object Management Overview

User (Permissions)

Database

Table

Index

Subroutines (functions, triggers, stored procedures, etc.)

Using Mysql Tenants for Common Database Development

Database Operations - User Management - Prerequisites

When the database is running, it is often necessary to create different users and grant them corresponding permissions. Generally, users who are granted the CREATE USER permission can create users. By default, the business tenant's root user is used to create other business users.

Because the CREATE USER permission is extensive, by default only cluster administrators and tenant administrators have this system permission. If other users need to create users, they need to be granted the CREATE USER permission. For authorization-related operations, see Directly Granting Permissions.

Using Mysql Tenants for Common Database Development

Database Operations - User Management - User Name Rules

When specifying a name for a user, the following restrictions should be noted:

- Uniqueness of usernames: Each username needs to be ensured to be unique within the tenant.
  The user name is unique within the tenant, and users under different tenants can have the same name.
  Therefore, by using the username @ tenant name format, a tenant user can be uniquely located globally in the system

Since system tenants and MySQL tenants belong to the same compatibility mode, it is recommended to use a specific prefix for user names under system tenants to distinguish them from users under regular tenants.

- The naming convention for usernames:
  - When creating users using OBClient, ODC, and other clients, it is required that the username length does not exceed 64 bytes
  - When creating a user using OCP, the user name must start with an English letter, contain uppercase letters, lowercase letters, numbers, and underscores, and be 2 to 64 characters long.

Using Mysql Tenants for Common Database Development

Database Operation-User Management-Create User

Currently, two methods are supported for creating users:

- create user
- "grant" statements automatically create users.

```
MySQL [oceanbase]> create user user01 identified by
'zf******MG';
Query OK, 0 rows affected (0.024 sec)

MySQL [oceanbase]> grant all privileges on test.* to user01 ;
Query OK, 0 rows affected (0.013 sec)

MySQL [oceanbase]> grant all privileges on test.* to user02
identified by 'dQ******M8';
Query OK, 0 rows affected (0.028 sec)
```

For more user management-related operations, please refer to Permission Management in MySQL Mode.

OceanBase MySQL tenants **do not support** updating the password field in user metadata.
You can view user permissions using the show grants statement.

```
MySQL [oceanbase]> show grants for user01;
+-----------------------------------------------+
| Grants for user01@%                           |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO 'user01'                |
| GRANT ALL PRIVILEGES ON `test`.* TO 'user01' |
+-----------------------------------------------+
2 rows in set (0.001 sec)

MySQL [oceanbase]> show grants for user02;
+-----------------------------------------------+
| Grants for user02@%                           |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO 'user02'                |
| GRANT ALL PRIVILEGES ON `test`.* TO 'user02' |
+-----------------------------------------------+
2 rows in set (0.001 sec)
```

Using Mysql Tenants for Common Database Development

Database Operations - Database Management - Prerequisites

In OceanBase, the database includes several tables, indexes, and metadata information of database objects. Try to avoid using the default database in a production environment, such as a test database. If it is not necessary, it is recommended to create your own database using SQL statements.

Before managing the database, you need to confirm the following:

- The OceanBase cluster has been deployed and a MySQL mode tenant has been created.
- The MySQL tenant has been connected to the OceanBase database.
- The ‘CREATE’, ‘ALTER’, and ‘DROP’ permissions have been obtained.

Using Mysql Tenants for Common Database Development

Database Operations - Database Management - Create Database Restrictions

When creating a database, please note the following restrictions:

- In OceanBase database, the name of each database must be globally unique.
- The database name is limited to 128 characters.
- It can only contain uppercase and lowercase letters, numbers, underscores, dollar signs, and Chinese characters.
- Avoid using reserved keywords as database names. For details on reserved keywords for the OceanBase database MySQL mode, see Reserved Keywords.

Using Mysql Tenants for Common Database Development

Database Operations - Database Management - Suggestions for Creating a Database

- It is recommended to give the database a meaningful name that reflects its purpose and content. For example, use application ID_sub-application name (optional)_db as the database name.
- It is recommended to use the root user to create the database and related users, and only grant necessary permissions to ensure the security and controllability of the database.
- When creating a database, make sure to set the appropriate default character and collation to ensure that data is stored and sorted correctly. To adapt to the long-term development of the business, it is recommended to use the utf8mb4 character set encoding when creating a database to ensure that most characters can be stored.
- You can create a database name with pure numbers by surrounding it with backticks (`), but this is not recommended because pure numeric names have no obvious meaning, and backticks (`) are required for query usage, which will lead to unnecessary complexity and confusion when using it.

Using Mysql Tenants for Common Database Development

Database Operations - Database Management - Creating a Database

Example 1: Create a database test_db and specify the character set as utf8mb4.

- obclient [(none)]> CREATE DATABASE test_db DEFAULT CHARACTER SET utf8mb4;

Example 2: Create a read-only database test_ro_db.

- obclient [(none)]> CREATE DATABASE test_ro_db READ ONLY;

Example 3: Create a database test_rw_db with read-write properties.

- obclient [(none)]> CREATE DATABASE test_rw_db READ WRITE;

For more detailed syntax related to database management, please refer to the “Creating and Managing Databases” section o n the OceanBase official website.

Using Mysql Tenants for Common Database Development

Database Operations - Table Management - Table Types

The OceanBase database supports primary key tables and tables without primary keys.

Primary key table:

A table with a primary key, i.e., a table with a primary key, needs to meet the following rules in the OceanBase database:

- Each data table has at most one primary key column set
- The number of primary key columns cannot exceed 64, and the total length of primary key data cannot exceed 16 KB

After creating a table with a primary key, a global unique index is automatically created for the primary key column, so that rows can be quickly located by the primary key.

As shown in the following example, a table 'emp_table' with 'emp_id' as the primary key is created. It belongs to the table with a primary key.

```
CREATE TABLE emp_table (
emp_id INT PRIMARY KEY,
emp_name VARCHAR(100),
emp_age INT NOT NULL
);
```

Table without primary key:

As shown in the following example, the data table `student_table` has no primary key specified, so it belongs to the table without primary key.

```
CREATE TABLE student_table (
student_id INT NOT NULL,
student_name VARCHAR(100),
student_age INT NOT NULL
);
```

The non-primary-key table in the OceanBase database uses a partition-level auto-increment column as a hidden primary key.

Using Mysql Tenants for Common Database Development

Database Operations - Table Management - Table Restrictions and Suggestions

Restrictions on creating tables:

- The name of a table cannot exceed 64 characters.
- The row length of a table cannot exceed 1.5M bytes.
- The number of columns in a single table cannot exceed 4096.
- The number of indexes in a single table cannot exceed 128.
- The total number of primary key columns in a single table cannot exceed 64, and the total length of primary key data cannot exceed 16 KB.
- The maximum number of partitions for a single table is controlled by the tenant-level configuration item max_partition_num (supported after V4.2.1_CE_BP3), which defaults to 8192 and cannot exceed 65536.

For more restrictions, please refer to the "Usage Restrictions" section on the OceanBase official website.

Suggestions for creating tables:

- It is recommended to consistently use either all uppercase or all lowercase English letters for table names, avoiding the mixing of upper and lowercase letters.
- It is recommended that the table name should be self–explanatory. Example: "TEST".
- System reserved words and keywords should not be used in the table name.
- The intermediate table is used to retain the intermediate result set. The recommended naming rule is: "tmp_table name (or abbreviation) column name (or abbreviation) creation time", example: "tmp_account_tbluser_20220224".
- The backup table is used to back up or capture the source table snapshot. The recommended naming rule is: "bak_table name (or abbreviation) column name (or abbreviation) creation time", example: "bak_ account_tbluser _20220224"

For more information about table naming standards, see Table Naming Standards.

Using Mysql Tenants for Common Database Development

Database Operation - Table Management - Table Creation Example

```
create table ware(w_id int
, w_ytd decimal(12,2)
, w_tax decimal(4,4)
, w_name varchar(10)
, w_street_1 varchar(20)
, w_street_2 varchar(20)
, w_city varchar(20)
, w_state char(2)
, w_zip char(9)
, unique(w_name, w_city)
, primary key(w_id)
);
```

Note:
OceanBase MySQL tenants support foreign keys. However, in a distributed database, if the read-and-write concurrency is high, it is not recommended to use foreign key constraints at the database level. Foreign keys will increase unnecessary blocking and deadlocks, which may hurt performance.

```
create table cust (c_w_id int NOT NULL
, c_d_id int NOT null
, c_id int NOT null
, c_discount decimal(4, 4)
, c_credit char(2)
, c_last varchar(16)
, c_first varchar(16)
, c_middle char(2)
, c_balance decimal(12, 2)
, c_ytd_payment decimal(12, 2)
, c_payment_cnt int
, c_credit_lim decimal(12, 2)
, c_street_1 varchar(20)
, c_street_2 varchar(20)
, c_city varchar(20)
, c_state char(2)
, c_zip char(9)
, c_phone char(16)
, c_since date
, c_delivery_cnt int
, c_data varchar(500)
, index icust(c_last, c_d_id, c_w_id, c_first, c_id)
, FOREIGN KEY (c_w_id) REFERENCES ware(w_id)
, primary key (c_w_id, c_d_id, c_id)
);
```

Using Mysql Tenants for Common Database Development

Database Operation - Table Management - Table Creation Example

Use **like** to copy the structure of a table. The structure of the table including the primary key, unique key, and index name will be copied. In MySQL syntax, the primary key name, unique constraint, and index name cannot be repeated in a table, but can be repeated between different tables.

create table t1 like ware;

```
MySQL [test]> show create table t1\G
*************************** 1. row ***************************
Table: t1
Create Table: CREATE TABLE `t1` (
`w_id` int(11) NOT NULL,
`w_ytd` decimal(12,2) DEFAULT NULL,
`w_tax` decimal(4,4) DEFAULT NULL,
`w_name` varchar(10) DEFAULT NULL,
`w_street_1` varchar(20) DEFAULT NULL,
`w_street_2` varchar(20) DEFAULT NULL,
`w_city` varchar(20) DEFAULT NULL,
`w_state` char(2) DEFAULT NULL,
`w_zip` char(9) DEFAULT NULL,
PRIMARY KEY (`w_id`),
UNIQUE KEY `w_name` (`w_name`, `w_city`) BLOCK_SIZE 16384 GLOBAL
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = COMPACT COMPRESSION = 'zstd_1.3.8'
REPLICA_NUM = 1 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE TABLET_SIZE =
134217728 PCTFREE = 0
1 row in set (0.003 sec)
```

To copy the structure and data of a table, use **create table ... as select**. However, please **Note** that this statement only copies the basic data types of the table, and does not copy the primary key, unique constraint, and index information.

create table t2 as select * from ware;

```
MySQL [test]> show create table t2\G
*************************** 1. row ***************************
Table: t2
Create Table: CREATE TABLE `t2` (
`w_id` int(11) NOT NULL,
`w_ytd` decimal(12,2) DEFAULT NULL,
`w_tax` decimal(4,4) DEFAULT NULL,
`w_name` varchar(10) DEFAULT NULL,
`w_street_1` varchar(20) DEFAULT NULL,
`w_street_2` varchar(20) DEFAULT NULL,
`w_city` varchar(20) DEFAULT NULL,
`w_state` char(2) DEFAULT NULL,
`w_zip` char(9) DEFAULT NULL
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = COMPACT COMPRESSION = 'zstd_1.3.8'
REPLICA_NUM = 1 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE TABLET_SIZE =
134217728 PCTFREE = 0
1 row in set (0.002 sec)
```

For more detailed syntax related to table management, see the "Creating and Managing Tables" section in the OceanBase official website.

Using Mysql Tenants for Common Database Development

Database Operations - Index Management - Prerequisites

An index is also called a secondary index, which is an optional table structure. The OceanBase database uses a clustered index table model. For the primary key specified by the user, the system automatically generates a primary key index, and for other indexes created by the user, they are secondary indexes. You can decide which fields to create indexes on based on your business needs to speed up queries on these fields.

For more information about OceanBase database indexes, see Index Introduction.

Before creating an index, you need to confirm the following:

- The OceanBase cluster has been deployed and the MySQL mode tenant and user have been created.
- The MySQL tenant has been connected to the OceanBase database.
- The database and table have been created.
- The INDEX privilege has been obtained

Using Mysql Tenants for Common Database Development

Database Operations - Index Management - Create Index Restrictions

Create Index Restrictions:

- In OceanBase, index names must be unique within the scope of the table.
- The length of the index name cannot exceed 64 bytes.
- Unique index usage restrictions:
  - You can create multiple unique indexes in a table, but the column values corresponding to each unique index must remain unique.
  - If you want the combination of other columns in addition to the primary key to meet the global uniqueness requirement, you need to use a global unique index to achieve it.
  - When using a local unique index, the index must contain all columns in the partition function of the table.
- When using a global index, the partitioning rules of the global index do not necessarily need to be exactly the same or consistent with the partitioning rules of the table.

Using Mysql Tenants for Common Database Development

Database Operations - Index Management - Create Index Suggestions

Create index suggestions:

- It is recommended to use a name that succinctly describes the columns and purpose of the index, for example, idx_customer_name
- If the partitioning rules of the global index are the same as those of the main table and the number of partitions is the same, it is recommended to create a local index.
- It is recommended that the number of SQL statements issued in parallel to create indexes should not exceed the upper limit of the number of cores in the tenant's Unit specification. For example, if the tenant's Unit specification is 4 cores (4C), it is recommended that no more than 4 indexes be created concurrently.
- Avoid using too many indexes for frequently updated tables, and create indexes for fields that are frequently used for queries.
- It is recommended not to use indexes for tables with small data volumes, because due to the small amount of data, the time spent querying all data may be shorter than the time spent traversing the index, and the index may not produce an optimization effect.
- When the modification performance is much greater than the retrieval performance, it is not recommended to create an index.
- Creating efficient indexes:
  - The index should contain all the columns required for the query. The more columns it contains, the better. This can minimize the number of rows returned to the table.
  - Equal value conditions are always placed first.
  - Filtering and sorting large amounts of data are placed first.

Using Mysql Tenants for Common Database Development

Database Operation - Index Management - Create Index Example

Example: Use the following SQL statements to create a table named `tbl2` and create an index based on the `col2` column for the table `tbl2`.

1. Create table `tbl2`
obclient [test]> CREATE TABLE tbl2(col1 INT, col2 INT, col3 VARCHAR(50), PRIMARY KEY (col1));

2. Create an index named `idx_tbl2_col2` on the `col2` column of the table `tbl2`.
obclient [test]> CREATE INDEX idx_tbl2_col2 ON tbl2(col2);

3. View table `tbl2` index information.
obclient [test]> SHOW INDEX FROM tbl2;

The returned results are as follows:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
tbl2	0	PRIMARY	1	col1	A	NULL	NULL	NULL		BTREE	available		YES	NULL
tbl2	1	idx_tbl2_col2	1	col2	A	NULL	NULL	NULL	YES	BTREE	available		YES	NULL

2 rows in set

For more detailed syntax related to index management, see the "Creating and Managing Indexes" section in the OceanBase official website.

Using Mysql Tenants for Common Database Development

Database Operations - Other Objects - Functions, Triggers, Stored Procedures

Subroutines:

A subroutine is a PL unit that contains several SQL and PL statements to solve a specific problem or perform a set of related tasks. Subprograms can contain parameters, the specific values of which are passed in by the caller. A subroutine can be a stored procedure or function. Typical usage is to use a stored procedure to operate and a function to calculate and return a value.

Stored procedures are subprograms stored inside the database that implement complex logical operations for many different database application subprograms. MySQL schema subroutines are only independent programs, that is, programs created within the Schema.

Note: MySQL mode subroutines follow the standards for stored procedures in the SQL standard, and are significantly different in syntax and functionality from Oracle mode subroutines (PL).

You can directly refer to the "PL - MySQL Mode" section on the official website to complete the self-study

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Inserting Data

Prerequisites:

- A MySQL tenant that is connected to the database
- Has INSERT privileges on the table to be operated on

Table Schema:

```
obclient [test]> CREATE TABLE t_insert(
id int NOT NULL PRIMARY KEY,
name varchar(10) NOT NULL,
value int,
gmt_create DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP
);
Query OK, 0 rows affected
```

The table's `id` and `name` columns cannot be empty, and the `id` column is the primary key column, which satisfies the unique constraint requirements and cannot have duplicate values; the `gmt_create` column specifies a default value.

Single row insert:

```
obclient [test]> INSERT INTO t_insert(id, name, value)
VALUES (1,'CN',10001);
Query OK, 2 rows affected
```

Multiple row insert:

When inserting data, if you want to insert multiple records, you can also use a single `INSERT` statement to include multiple `VALUES` to batch insert. A single multi-row insert statement is faster than multiple single-row insert statements.

```
obclient [test]> INSERT INTO t_insert(id, name, value)
VALUES (1,'CN',10001),(2,'US', 10002);
Query OK, 2 rows affected
```

In addition, when you need to back up table data or copy all records of a table to another table, you can use the query statement `INSERT INTO ... SELECT ... FROM` as the `values` clause of `INSERT` to perform batch insertion.

```
obclient [test]> INSERT INTO t_insert_bak SELECT * FROM t_insert;
Query OK, 2 rows affected
```

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Inserting Data

Table Schema:

```
obclient [test]> CREATE TABLE t_replace(
id int NOT NULL PRIMARY KEY,
name varchar(10) NOT NULL,
value int,
gmt_create timestamp NOT NULL DEFAULT current_timestamp
);
Query OK, 0 rows affected
```

Other ways to insert data:

In addition to the INSERT statement, when there are no data records in the table, or when there are data records in the table but there is no primary key or unique key conflict, you can also use the REPLACE INTO statement instead of the INSERT statement to insert data.

For detailed syntax and description of the REPLACE INTO statement, see REPLACE.

```
obclient [test]> SELECT * FROM t_replace;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 2001 | 2022-11-23 09:52:44 |
+----+------+-------+---------------------+
1 row in set
```

```
obclient [test]> REPLACE INTO t_replace values(2,'US',2002,
current_timestamp ());
Query OK, 1 row affected
```

```
obclient [test]> SELECT * FROM t_replace;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 2001 | 2022-11-23 09:52:44 |
| 2 | US | 2002 | 2022-11-23 09:53:05 |
+----+------+-------+---------------------+
2 rows in set
```

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Data Updating

The UPDATE statement is usually used to update table data.

Prerequisites:

- A MySQL tenant connected to the database
- Already has the UPDATE privilege for the table to be operated on

Table Schema:

```
obclient [test]> CREATE TABLE t_insert(
id int NOT NULL PRIMARY KEY,
name varchar(10) NOT NULL,
value int,
gmt_create DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP
);
Query OK, 0 rows affected
```

Grammar:

```
UPDATE table_name SET column_name = value [,
column_name = value]... [ WHERE condition ];
```

Example:

```
obclient [test]> UPDATE t_insert SET value = value+1;
Query OK, 4 rows affected
Rows matched: 4 Changed: 4 Warnings: 0

obclient [test]> SELECT * FROM t_insert;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 10002 | 1970-01-01 17:18:06 |
| 2 | US | 10003 | 1970-01-01 17:18:47 |
| 3 | EN | 10004 | 1970-01-01 17:18:47 |
| 4 | JP | 10005 | 1970-01-01 17:28:21 |
+----+------+-------+---------------------+
4 rows in set
```

Note:

When executing an UPDATE statement, be careful not to make the transaction too large. You can use the LIMIT keyword to control the number or the WHERE keyword to control the range. This is because when updating data without conditions, if the number of records reaches hundreds of thousands or millions, a large transaction will be generated, which may cause execution failure.

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Data Updating

Other statements update data:

In addition to explicit UPDATE statements, other statements can also update data. For example, when inserting data through the INSERT statement, due to constraint conflicts, you can use the ON DUPLICATE KEY UPDATE clause to convert the insert data into an update data statement to update the relevant fields.

Use the ON DUPLICATE KEY UPDATE clause to convert the insert data into an update data statement

Table Schema:

```
obclient [test]> SELECT * FROM t_insert;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 10001 | 2022-10-12 15:17:17 |
| 2 | US | 10002 | 2022-10-12 16:29:16 |
| 3 | EN | 10003 | 2022-10-12 16:29:26 |
| 4 | JP | 10004 | 2022-10-12 17:02:52 |
+----+------+-------+---------------------+
4 rows in set
```

Example:

```
obclient [test]> INSERT INTO t_insert(id, name, value) VALUES
(3,'UK', 10003),(5, 'CN', 10005) ON DUPLICATE KEY UPDATE name =
VALUES(name);
Query OK, 1 row affected

obclient [test]> SELECT * FROM t_insert;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 10001 | 2022-10-12 16:29:16 |
| 2 | US | 10002 | 2022-10-12 15:17:17 |
| 3 | UK | 10003 | 2022-10-12 16:29:26 |
| 4 | JP | 10004 | 2022-10-12 17:02:52 |
| 5 | CN | 10005 | 2022-10-12 17:27:46 |
+----+------+-------+---------------------+
5 rows in set
```

In the example, ON DUPLICATE KEY UPDATE name = VALUES(name) means that when the inserted data has a duplicate primary key value in the table, the value of the name column in the original data of the conflicting row in the table (3,'EN', 10003) is updated to the value of the name column to be inserted.

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Data Deletion

The DELETE statement is usually used to delete some or all data in a table.

Prerequisites:

- A MySQL tenant is connected to the database.
- Have the DELETE privilege for the table to be operated. If you use the TRUNCATE TABLE statement to clear the table data, you also need the CREATE privilege for the table.

Table Schema:

```
obclient [test]> CREATE TABLE t_insert(
id int NOT NULL PRIMARY KEY,
name varchar(10) NOT NULL,
value int,
gmt_create DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP
);
Query OK, 0 rows affected
```

Gammar:

```
DELETE FROM table_name [ WHERE condition ] ;
TRUNCATE [TABLE] table_name;
```

DELETE Exmple:

```
obclient [test]> DELETE FROM t_insert WHERE value < 100000;
```

Note:

When there are more than one million records in the table, deleting them all at once may cause performance problems. It is recommended to use the WHERE condition to delete them in batches according to the information in the table, or to directly use the TRUNCATE TABLE statement to clear the table data.

TRUNCATE Example:

```
obclient [test]> TRUNCATE TABLE t_insert;
```

The TRUNCATE TABLE statement is used to completely empty the specified table, but retain the table structure, including the partition information defined in the table. Logically speaking, this statement is equivalent to the DELETE FROM statement that deletes all rows.

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Data Replacement

The REPLACE INTO statement is usually used to replace one or more records in a table.

Prerequisites:

- A MySQL tenant that is connected to the database.
- Has INSERT, UPDATE, and DELETE permissions on the table to be operated on.

Grammar:

REPLACE INTO table_name VALUES(list_of_values);

The **REPLACE INTO** statement will judge the replacement data based on the primary key or unique key of the table:

- If there is no primary key or unique key conflict, insert the record.
- If there is a primary key or unique key conflict, delete the existing record first, then insert the new row record. It is recommended that the target table has a primary key or unique index, otherwise it is easy to insert duplicate records

Example 1:

No data was inserted into the **t_replace** table after it was created. After the **REPLACE INTO** statement was executed, a record was inserted into the table.

```
obclient [test]> REPLACE INTO t_replace values(1,'CN',2001, current_timestamp ());
Query OK, 1 row affected

obclient [test]> SELECT * FROM t_replace;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 2001 | 2022-10-13 14:06:58 |
+----+------+-------+---------------------+
1 row in set
```

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Data Replacement

Example 2:

- Use the REPLACE INTO statement again to insert a row of data.
- There is already a record in the t_replace table. Since the (2,'US',2002,current_timestamp()) data does not violate the uniqueness constraint with the records in the table, the execution result is to insert a record in the t_replace table.

```
obclient [test]> REPLACE INTO t_replace(id, name, value, gmt_create) VALUES(2,'US',2002,current_timestamp ());
Query OK, 1 row affected

obclient [test]> SELECT * FROM t_replace;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 2001 | 2022-10-13 14:06:58 |
| 2 | US | 2002 | 2022-10-13 14:17:56 |
+----+------+-------+---------------------+
2 rows in set
```

Example 3:

Use a query statement as the `VALUES` clause of the `REPLACE INTO` statement to insert multiple records. Insert the data in the table `t_insert` into the table `t_replace`

```
obclient [test]> SELECT * FROM t_insert;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 7 | EN | 1007 | 2022-10-13 14:36:36 |
| 8 | JP | 1008 | 2022-10-13 14:36:36 |
+----+------+-------+---------------------+
2 rows in set

obclient [test]> REPLACE INTO t_replace
SELECT id,name,value,gmt_create FROM t_insert;
Query OK, 2 rows affected
Records: 2 Duplicates: 0 Warnings: 0

obclient [test]> SELECT * FROM t_replace;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 2001 | 2022-10-13 14:06:58 |
| 2 | US | 2002 | 2022-10-13 14:17:56 |
| 7 | EN | 1007 | 2022-10-13 14:36:36 |
| 8 | JP | 1008 | 2022-10-13 14:36:36 |
+----+------+-------+---------------------+
4 rows in set
```

Using Mysql Tenants for Common Database Development

Database Operation - Data Writing - Data Replacement

Example 4:
When there are data records in the table and the inserted data has a primary key or unique key conflict, the REPLACE INTO statement can be used to delete the conflicting data in the table and replace it with the new data.

```
obclient [test]> SELECT * FROM t_replace;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 2001 | 2022-10-13 14:06:58 |
| 2 | US | 2002 | 2022-10-13 14:17:56 |
| 7 | EN | 1007 | 2022-10-13 14:36:36 |
| 8 | JP | 1008 | 2022-10-13 14:36:36 |
+----+------+-------+---------------------+
4 rows in set
```

Use a query statement as the `VALUES` clause of the `REPLACE INTO` statement to insert multiple records. Insert the data in the table `t_insert` into the table `t_replace`

```
obclient [test]> REPLACE INTO t_replace(id, name, value, gmt_create) VALUES(2,'EN',2002,current_timestamp ());
Query OK, 2 rows affected

obclient [test]> SELECT * FROM t_replace;
+----+------+-------+---------------------+
| id | name | value | gmt_create |
+----+------+-------+---------------------+
| 1 | CN | 2001 | 2022-10-13 14:06:58 |
| 2 | EN | 2002 | 2022-10-13 14:44:33 |
| 7 | EN | 1007 | 2022-10-13 14:36:36 |
| 8 | JP | 1008 | 2022-10-13 14:36:36 |
+----+------+-------+---------------------+
4 rows in set
```

Using Mysql Tenants for Common Database Development

Database Operation - Data Reading - Single Table Query

Parameter	Description
select_list	A list of columns to retrieve, which can be column names, expressions, aggregate functions, etc. Multiple columns can be separated by commas
table_name	The name of the table to retrieve data from.
WHERE query_condition	Optional parameter. Used to specify the search conditions. Only rows that meet the conditions will be returned.
GROUP BY group_by_condition	Optional parameter. Used to group the results by the specified column. Usually used with aggregate functions.
HAVING group_condition	Optional parameter. Used to filter the result set after grouping and return only the groups that meet the conditions.
ORDER BY column_list	Optional parameter. Used to sort the result set. You can specify one or more columns to sort.
ASC	DESC
LIMIT limit_clause	Optional parameter. Used to limit the number of rows returned in the result set.
column_list	Parameters used to specify the columns to retrieve, can be a single column or multiple columns, separated by commas
column_name	The name of the column to retrieve.

Parameter Explanation:

When the keywords `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, and `LIMIT` are used together, the order of precedence is limited. The order of keyword execution is as follows:

1. Execute `FROM` to find the table.
2. Execute `WHERE` to specify constraints. Filter data before grouping
3. Execute `GROUP BY` to group (aggregate) each record taken out. If there is no `GROUP BY`, the whole is grouped as a group.
4. Execute `HAVING` to filter the grouped results and finally return the query results of the entire SQL
5. Execute `SELECT`.
6. Execute `DISTINCT` to remove duplicates.
7. Execute `ORDER BY` to sort the results in ascending or descending order according to the conditions.
8. Execute `LIMIT` to limit the number of results

Using Mysql Tenants for Common Database Development

Database Operation - Data Reading - Single Table Query

Query Examples:

```
SELECT * FROM student;
```

The returned results are as follows:

```
+----+-----------+--------+-----+-------+-----------------+-------+
| id | name      | gender | age | score | enrollment_date | notes |
+----+-----------+--------+-----+-------+-----------------+-------+
|  1 | Emma      |      0 |  20 |    85 | 2021-09-01      | NULL  |
|  2 | William   |      1 |  21 |  90.5 | 2021-09-02      | B     |
|  3 | Olivia    |      0 |  19 |  95.5 | 2021-09-03      | A     |
|  4 | James     |      1 |  20 |  87.5 | 2021-09-03      | NULL  |
|  5 | Sophia    |      0 |  20 |  91.5 | 2021-09-05      | B     |
|  6 | Benjamin  |      1 |  21 |  96.5 | 2021-09-01      | A     |
|  7 | Ava       |      0 |  22 |  89.5 | 2021-09-06      | NULL  |
|  8 | Michael   |      1 |  18 |  93.5 | 2021-09-08      | B     |
|  9 | Charlotte |      1 |  19 |    88 | 2021-09-06      | NULL  |
| 10 | Ethan     |      1 |  20 |    92 | 2021-09-01      | B     |
+----+-----------+--------+-----+-------+-----------------+-------+
10 rows in set
```

Data Filtering:

To query data that meets certain conditions, you can add a WHERE clause to the SELECT query statement to filter data. The WHERE clause can contain one or more conditions, which are used to filter data. Only data that meets the WHERE conditions will be returned. You can filter and retrieve target data by flexibly applying query conditions according to specific needs.

Query Condition Type	Predicate
Comparison Query	=, >, <, >=, <=, !=, <>
Logical Query (multiple conditions)	AND, OR, NOT
Fuzzy search (character matching)	LIKE, NOT LIKE
Interval query (determine range)	BETWEEN AND, NOT BETWEEN AND
Specifying a collection query	IN, NOT IN
NULL value query	IS NULL, IS NOT NULL

For more information about comparison operators, see Comparison Operators.

Using Mysql Tenants for Common Database Development

Database Operation - Data Reading - Single Table Query

Query Examples:

Display student information in ascending order of score in the student table

```
SELECT id, name, score
FROM student
ORDER BY score;
```

The returned results:

```
+----+-----------+-------+
| id | name      | score |
+----+-----------+-------+
|  1 | Emma      |    85 |
|  4 | James     |  87.5 |
|  9 | Charlotte |    88 |
|  7 | Ava       |  89.5 |
|  2 | William   |  90.5 |
|  5 | Sophia    |  91.5 |
| 10 | Ethan     |    92 |
|  8 | Michael   |  93.5 |
|  3 | Olivia    |  95.5 |
|  6 | Benjamin  |  96.5 |
+----+-----------+-------+
10 rows in set
```

Data Grouping:

You can use the GROUP BY clause to group query results in SQL queries. GROUP BY supports single-field grouping and multi-field grouping. Before grouping, you can use the WHERE clause to filter data, the HAVING clause to filter data after grouping, and the ORDER BY clause to sort data after grouping.

- When using the GROUP BY clause, the columns in the SELECT statement must be columns or aggregate functions in the GROUP BY clause.
- When using the HAVING clause, the HAVING condition filters the grouped results instead of the original data.

When the query contains HAVING, the SQL query result without the HAVING clause is obtained first, and then the HAVING condition is used to filter out the matching data based on this result, and finally the data is returned. Therefore, the aggregate function can be used after HAVING, and this aggregate function does not have to be the same as the aggregate function after SELECT

Using Mysql Tenants for Common Database Development

Database Operation - Data Reading - Single Table Query

The LIMIT clause restricts the result set:
In SQL queries, you can use the LIMIT clause to limit the number of rows returned in the result set.

LIMIT Grammar:
1、LIMIT [offset,] row_count LIMIT

2、LIMIT row_count OFFSET offset

- Offset: Indicates the offset, that is, the number of rows to skip. In format 1, the offset is optional and defaults to 0, indicating skipping 0 rows. The offset value range is [0,+∞).

- row_count: Indicates the number of rows to be returned. If offset is not specified in format 1, the default is to start from the first row. The value range of row_count is [0,+∞)

- The values of offset and row_count have the following restrictions:
  - Expressions cannot be used
  - It can only be a clear number, not a negative number

Example:
Query the three rows of data after the fifth row of columns id and name in the student table

```
SELECT id, name
FROM student
LIMIT 3 OFFSET 5;
```

The returned results

```
+----+----------+
| id | name     |
+----+----------+
|  6 | Benjamin |
|  7 | Ava      |
|  8 | Michael  |
+----+----------+
3 rows in set
```

Using Mysql Tenants for Common Database Development

Database Operation - Data Reading - Paging Query

LIMIT pagination query:
In SQL queries, you can use the LIMIT clause to limit the number of rows returned in the result set.

LIMIT paging query syntax:

LIMIT (page_no - 1) * page_size, page_size;

- page_no: indicates the page number, starting from 1 and ranging from [1,+∞).
- page_size: indicates how many records are displayed per page, the range is [1,+∞). For example: page_no = 5, page_size = 10, means to get 10 records on page 5.

Example:
In the student table, 2 data are displayed per page. Get the data of page 1 and page 2 in turn.

Page 1

```
SELECT id, name
FROM student
ORDER BY id
LIMIT 0,2;
```

The returned results:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | Emma    |
|  2 | William |
+----+---------+
2 rows in set
```

Page 2

```
SELECT id, name
FROM student
ORDER BY id
LIMIT 2,2;
```

The returned results:

```
+----+--------+
| id | name   |
+----+--------+
|  3 | Olivia |
|  4 | James  |
+----+--------+
2 rows in set
```

Using Mysql Tenants for Common Database Development

Database Operation - Data Reading - Subquery

Subqueries:

A subquery is a query that is nested within a parent query. The parent query is often referred to as the parent query or outer query. The result of the subquery is passed back as input to the "parent query" or "outer query". The parent query incorporates this value into its calculations to determine the final output.

SQL allows multiple nested queries, that is, a subquery can be nested in other subqueries. At the same time, subqueries can appear in various clauses in SQL statements, such as SELECT statements, FROM statements, WHERE statements, etc.

For detailed syntax related to subqueries, please refer to the "Subqueries" section on the OceanBase official website.

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Overview

A database transaction consists of a series of operations on the database. A transaction transforms the database from one consistent state to another consistent state.

Database transactions have two functions:

- Provides a method for database operation sequences to recover from failures to a normal state, and also provides a method for the database to maintain consistency even in abnormal states
- Provides isolation for multiple concurrent accesses to the database to prevent multiple concurrent operations from causing the database to enter an inconsistent state

In the OBClient command line environment, you can issue transaction control commands after the SQL prompt, or you can control whether the transaction is automatically committed by modifying the session-level autocommit variable.

- When setting variables through SET autocommit, the current session takes effect immediately. The variables set after disconnecting the connection will become invalid.
- When setting tenant-level variables through SET GLOBAL autocommit, you need to disconnect the connection to take effect.
- If the autocommit value of the current Session is 0 and there is no explicit transaction commit, when the program is abnormally interrupted, the OceanBase database will automatically roll back the last uncommitted transaction

Basic transaction control statements:

- BEGIN: explicitly start a transaction. This statement is optional during use:
  - When the value of the tenant session system variable autocommit is 0, it means that the transaction autocommit function is turned off, and there is no need to explicitly issue the BEGIN command to identify multiple SQLs as a transaction.
  - When the value of the tenant session system variable autocommit is 1, it means that the transaction autocommit function is turned on. In this mode, each SQL is an independent transaction. If you want multiple SQLs to form a transaction, you can explicitly start a transaction through the BEGIN command, and the transaction autocommit function will be disabled. It will not be restored to the autocommit mode until the COMMIT or ROLLBACK statement is executed.
- SAVEPOINT: Marks a "save point" during a transaction, and the transaction can be rolled back to this point later. Save points are optional, and there can be multiple save points during a transaction.
- COMMIT: commits and ends the current transaction, making all changes in the transaction persistent and effective, clearing all save points, and releasing the locks held by the transaction
- ROLLBACK: Rolls back the changes made by the entire transaction or only rolls back the changes made by the transaction after a certain savepoint, clears all savepoints included in the rollback part, and releases the locks held by the transaction

Using Mysql Tenants for Common Database Development

Database Operation - Transaction Management - Start Transaction

Open transaction example:

Execute BEGIN Command

```
obclient [test]> BEGIN;        // Open transaction
obclient [test]> INSERT INTO table1 VALUES(1,1);
obclient [test]> COMMIT;
```

Execute START TRANSACTION Command

```
obclient [test]> START TRANSACTION; // Open transaction
obclient [test]> INSERT INTO table1 VALUES(1,1);
obclient [test]> COMMIT;
```

After configuring autocommit=0 (turning off autocommit), the system will open a new transaction by default when executing INSERT, UPDATE, DELETE, or SELECT FOR UPDATE statements.

```
obclient [test]> SET AUTOCOMMIT=0;
obclient [test]> INSERT INTO table1 VALUES(1,1); // Open transaction
obclient [test]> COMMIT;

obclient [test]> SET AUTOCOMMIT=0;
obclient [test]> UPDATE table1 SET id = 2 WHERE id = 1;  // Open transaction
obclient [test]> COMMIT;

obclient [test]> SET AUTOCOMMIT=0;
obclient [test]> DELETE FROM table1 WHERE id = 2;  // Open transaction
obclient [test]> COMMIT;

obclient [test]> SET AUTOCOMMIT=0;
obclient [test]> SELECT id FROM table1 WHERE id = 1 FOR UPDATE; // Open transaction
obclient [test]> COMMIT;
```

Using Mysql Tenants for Common Database Development

Database Operation - Transaction Management - Start Transaction

Open transaction example 2:

When a transaction is started, the OceanBase database assigns a transaction ID to the transaction, which is used to uniquely identify the transaction.

In actual use, when operating on the same data table under multiple concurrent connections, two transactions may operate on the same row of data. For query read operations, you can use the SELECT FOR UPDATE statement to lock the query results to prevent other DML statements from modifying the record at the same time.

```
obclient [test]> SELECT * FROM oceanbase.V$OB_TRANSACTION_PARTICIPANTS;
*************************** 1. row ***************************
       TENANT_ID: 1002
          SVR_IP: xx.xx.xx.223
        SVR_PORT: 2882
      SESSION_ID: 3221487660
  SCHEDULER_ADDR: "xx.xx.xx.223:2882"
         TX_TYPE: UNDECIDED
           TX_ID: 110352
           LS_ID: 1001
    PARTICIPANTS: NULL
 CTX_CREATE_TIME: 2022-10-19 14:55:23.763474
 TX_EXPIRED_TIME: 2022-10-19 14:55:23.763474
           STATE: ACTIVE
          ACTION: START
PENDING_LOG_SIZE: 116
FLUSHED_LOG_SIZE: 0
            ROLE: LEADER
1 row in set
```

The following example turns off the autocommit function by using SET autocommit=0, and then starts a transaction by using the UPDATE statement.

```
obclient [test]> CREATE TABLE ordr(
id INT NOT NULL PRIMARY KEY,
name VARCHAR(10) NOT NULL,
value INT,
gmt_create DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP );
Query OK, 0 rows affected

obclient [test]>  INSERT INTO ordr(id, name, value)
VALUES (1,'CN',10001),(2,'US', 10002),(3,'EN', 10003);
Query OK, 3 rows affected
 Records: 3  Duplicates: 0  Warnings: 0

obclient [test]> SELECT * FROM ordr;
+----+-------+-------+---------------------+
| id | name | value | gmt_create          |
+----+-------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   | 10003 | 2022-10-19 14:51:12 |
+----+-------+-------+---------------------+
2 rows in set

obclient [test]> SET autocommit=0;
Query OK, 0 rows affected

obclient [test]> UPDATE ordr SET id=4 WHERE name='US';
Query OK, 1 row affected
Rows matched: 1  Changed: 1  Warnings: 0
```

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management – Savepoints (I)

Savepoint is an execution marker within a transaction that can be defined by the user and provided by the OceanBase database. Users can define several markers within a transaction and restore the transaction to the state at the specified marker when needed.

When the user performs some incorrect operations after defining a Savepoint during the execution process, the user does not need to roll back the entire transaction and re-execute it, but can roll back the changes after the Savepoint by executing the ROLLBACK TO command.

As shown in the following table, users can create Savepoint sp1 to roll back the data inserted later.

Command	Description
BEGIN;	Open transaction
INSERT INTO a VALUE(1);	Insert row 1
SAVEPOINT sp1;	Create a Savepoint named sp1
INSERT INTO a VALUE(2);	Insert row 2
SAVEPOINT sp2;	Create a Savepoint named sp2
ROLLBACK TO sp1;	Roll back the changes to sp1
INSERT INTO a VALUE(3);	Insert row 3
COMMIT;	Commit transaction

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Savepoints (II)

In the implementation of OceanBase database, each modification during the transaction execution has a corresponding "sql sequence", which is incremented during the transaction execution (without considering the parallel execution scenario). The operation of creating a Savepoint maps the Savepoint name created by the user to the current "sql sequence" of the transaction execution. When the ROLLBACK TO command is executed, the following operations are performed inside the OceanBase database:

1. Roll back all modifications in the transaction that are greater than the “sql sequence” corresponding to the Savepoint, and release the corresponding row locks, such as row 2 in the last example.
2. Delete all Savepoints created after this Savepoint, such as sp2 in the last example.
   After the ROLLBACK TO command is executed successfully, the transaction can continue to operate.

For detailed syntax related to savepoints, see the "Savepoints" section on the OceanBase official website.

Using Mysql Tenants for Common Database Development

Database Operation - Transaction Management - Commit Transaction

Committing a transaction:

The database can commit transactions explicitly or implicitly. To commit a transaction explicitly, you need to use the COMMIT statement or the Commit button (in the graphical client tool) to end the transaction. Implicit commit does not require active commit. When the variable autommit is set to 1, after each statement is executed, the OceanBase database will automatically commit the transaction in which the statement is located. One statement is a transaction.

The OceanBase database will implicitly initiate a COMMIT statement before and after the DDL statement, and will also commit the transaction.

Implicit commit is the process in which the OceanBase database automatically executes COMMIT on the currently active transaction without the user issuing a COMMIT/ROLLBACK statement to end the transaction.

Implicit commit occurs when:

- Execute a statement that opens a transaction
- Execute DDL

Using Mysql Tenants for Common Database Development

Database Operation - Transaction Management - Commit Transaction

Commit Transactions Explicitly :

Start a transaction with BEGIN, then use the INSERT statement to insert data into the ordr table, and finally commit the transaction explicitly with the COMMIT statement.

```
obclient [test]> SELECT * FROM ordr;
+----+------+-------+---------------------+
| id | name | value | gmt_create          |
+----+------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   | 10003 | 2022-10-19 14:51:12 |
+----+------+-------+---------------------+
3 rows in set

obclient [test]> BEGIN;
Query OK, 0 rows affected

obclient [test]> INSERT INTO ordr(id,name) VALUES(4,'JP');
Query OK, 1 row affected

obclient [test]> COMMIT;
Query OK, 0 rows affected
```

After successful execution, exit the session and reconnect, and you will find that the table data has been correctly inserted and saved successfully.

```
obclient [test]> exit;

$obclient -h192.168.0.0 -ut***@obbmsql#obdemo -P2883 -p******  test
Welcome to the OceanBase.  Commands end with ; or \g.
Your OceanBase connection id is 3221487662
Server version: OceanBase 4.0.0.0 (r100000252022102910-df01cef074936b9c9f177697500fad1dc304056f)

Copyright (c) 2000, 2018, OceanBase and/or its affiliates. All rights reserved.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

obclient [test]> SELECT * FROM ordr;
+----+------+-------+---------------------+
| id | name | value | gmt_create          |
+----+------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   | 10003 | 2022-10-19 14:51:12 |
|  4 | JP   |  NULL | 2022-10-19 14:51:44 |
+----+------+-------+---------------------+
4 rows in set
```

Using Mysql Tenants for Common Database Development

Database Operation - Transaction Management - Commit Transaction

Commit Transactions Implicitly:

Enable automatic commit by setting the configuration variable autocommit=1.

```
obclient [test]> SELECT * FROM ordr;
+----+------+-------+---------------------+
| id | name | value | gmt_create          |
+----+------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   | 10003 | 2022-10-19 14:51:12 |
|  4 | JP   |  NULL | 2022-10-19 14:51:44 |
+----+------+-------+---------------------+
4 rows in set

obclient [test]> SET autocommit=1;

obclient [test]>  INSERT INTO ordr(id,name) VALUES(5,'CN');
Query OK, 1 row affected
```

After successful execution, exit the session and reconnect, and you will find that the table data has been correctly inserted and saved successfully.

```
obclient [test]> exit;

$obclient -h192.168.0.0 -utpcc@obbmsql#obdemo -P2883 -p******  test
Welcome to the OceanBase.  Commands end with ; or \g.
Your OceanBase connection id is 3221487662
Server version: OceanBase 4.0.0.0 (r100000252022102910-df01cef074936b9c9f177697500fad1dc304056f)

Copyright (c) 2000, 2018, OceanBase Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

obclient [test]> SELECT * FROM ordr;
+----+------+-------+---------------------+
| id | name | value | gmt_create          |
+----+------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   | 10003 | 2022-10-19 14:51:12 |
|  4 | JP   |  NULL | 2022-10-19 14:51:44 |
|  5 | CN   |  NULL | 2022-10-19 14:53:56 |
+----+------+-------+---------------------+
5 rows in set
```

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Rollback Transaction (I)

Rollback Transaction:

Rolling back a transaction means undoing all changes made to the transaction. You can roll back the entire uncommitted transaction or roll back to any save point in the transaction.

After rolling back the entire transaction:

- All modifications are discarded.
- All savepoints are cleared.
- All locks held by the transaction are released.

Grammar:

ROLLBACK;

The following example rolls back all changes of the current transaction using ROLLBACK.

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Rollback Transaction (II)

```
obclient [test]> SELECT * FROM ordr;
+----+------+-------+---------------------+
| id | name | value | gmt_create          |
+----+------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   |  1003 | 2022-10-19 14:51:12 |
|  4 | JP   |  NULL | 2022-10-19 14:51:44 |
|  5 | CN   |  NULL | 2022-10-19 14:53:56 |
|  6 | JP   | 10007 | 2022-10-19 14:58:24 |
|  8 | FR   | 10008 | 2022-10-19 14:58:35 |
|  9 | RU   | 10009 | 2022-10-19 14:58:35 |
+----+------+-------+---------------------+
8 rows in set

obclient [test]> ROLLBACK;
Query OK, 0 rows affected

obclient [test]> SELECT * FROM ordr;
+----+------+-------+---------------------+
| id | name | value | gmt_create          |
+----+------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   |  1003 | 2022-10-19 14:51:12 |
|  4 | JP   |  NULL | 2022-10-19 14:51:44 |
|  5 | CN   |  NULL | 2022-10-19 14:53:56 |
+----+------+-------+---------------------+
5 rows in set
```

Figure 1

In this example, a transaction is explicitly started using BEGIN. Before the transaction is explicitly committed using COMMIT, all changes are visible only to the current session and are not persisted. You can use the ROLLBACK statement to roll back the changes.

```
obclient [test]> SELECT * FROM ordr;
+----+------+-------+---------------------+
| id | name | value | gmt_create          |
+----+------+-------+---------------------+
|  1 | CN   | 10001 | 2022-10-19 14:51:12 |
|  2 | US   | 10002 | 2022-10-19 14:51:12 |
|  3 | EN   | 10003 | 2022-10-19 14:51:12 |
|  4 | JP   |  NULL | 2022-10-19 14:51:44 |
|  5 | CN   |  NULL | 2022-10-19 14:53:56 |
+----+------+-------+---------------------+
5 rows in set

obclient [test]> BEGIN;
Query OK, 0 rows affected

obclient [test]> INSERT INTO ordr(id, name, value) VALUES(6,'JP',10007);
Query OK, 1 row affected

obclient [test]> INSERT INTO ordr(id, name, value) VALUES(8,'FR',10008),(9,'RU',10009);
Query OK, 2 rows affected
Records: 2  Duplicates: 0  Warnings: 0
```

Figure 2

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Rollback Transaction

Automatic Rollback

Automatic rollback is a rollback initiated by the OceanBase database without the user issuing a ROLLBACK command. It usually occurs in the following situations:

- Session disconnected
- Transaction execution timeout (ob_trx_timeout, transaction timeout, in microseconds)
- The session of the active transaction has no statement executed for a certain time (ob_trx_idle_timeout, transaction idle timeout, that is, the execution interval between two statements in the transaction exceeds this value, in microseconds.)

Transaction Interrupted

When an internal error occurs during the execution of a transaction, such as a participant node crash or other reasons that cause the transaction to be unable to continue, the current transaction cannot continue to successfully execute statements and can only be rolled back.

When this happens, the user will receive a "transaction need rollback" error when executing SQL statements, and the user needs to execute ROLLBACK to end the current transaction.

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Transaction Isolation Level

Isolation levels are used to describe the degree to which transactions interfere with each other when they are executed concurrently. The ANSI/ISO SQL standard (SQL 92) defines four isolation levels based on the abnormal situations that must be avoided during transaction execution. The higher the isolation level, the less mutual influence between transactions, and the fewer abnormal situations are allowed. In the highest isolation level, Serializable, no abnormal situations are allowed.

These abnormal situations that need to be avoided include:

- Dirty Read: A transaction reads data that has not been committed by other transactions.

- Non Repeatable Read: A row of data that has been read before is found to have been modified or deleted when it is queried again. For example: select c2 from test where c1=1; The result of the first query of c2 is 1. The result of the second query is 2 because other transactions have modified the value of c2.

- Phantom Read: During an execution request, when the same search condition is executed again, the result set reads a row that meets the condition and is newly inserted by another committed transaction.

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Transaction Isolation Level

Isolation Level	Dirty Read	Non-repeatable read	Phantom Read
Read Uncommitted	Possible	Possible	Possible
Read Committed	Not possible	Possible	Possible
Repeatable Read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

OceanBase (MySQL mode) Supported Isolation Levels :

- Read Committed: A query executed by a transaction can only see data committed before the query starts. Read Committed cannot prevent two abnormal situations - non-repeatable reads and phantom reads.
- Repeatable Read: The same batch of data read at different times within a transaction is consistent.
- Serializable: this isolation level is similar to the Serializable of the Oracle, but it is not strictly Serializable.

The default isolation level of OceanBase is **Read Committed**.

Only two isolation levels are implemented in the OceanBase, namely read committed and serializable. When the user specifies the isolation level of repeatable read, serializable is used. In other words, the isolation level of repeatable reads in the OceanBase is more stringent and will not cause phantom read anomalies. The read committed in the OceanBase will not cause dirty read anomalies, but may cause non-repeatable read and phantom read anomalies; while serializable will not cause dirty read, non-repeatable read, and phantom read anomalies.

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Isolation Level Comparison

Database	Read Uncommitted	Read Committed	Repeatable Read	Serializable
OceanBase	Unsupported syntax	Supported, consistent with the SQL standard	Supported, and no phantom reading	Supported, but not guaranteed to be strictly serializable
MySQL	Supported, dirty data may be read	Supported, consistent with the SQL standard	Supported, and no phantom reading	Support, strict serializability can be guaranteed
Oracle	Unsupported syntax	Supported, consistent with the SQL standard	Unsupported syntax	Supported, but not guaranteed to be strictly serializable
PostgreSQL versions prior to 9.1	The syntax is supported, but it is actually Read Committed	Supported, consistent with the SQL standard	Supported, and no phantom reading	Supported, but not guaranteed to be strictly serializable
PostgreSQL 9.1 and later	The syntax is supported, but it is actually Read Committed	Supported, consistent with the SQL standard	Supported, and no phantom reading	Support, strict serializability can be guaranteed

Using Mysql Tenants for Common Database Development

Database Operations - Transaction Management - Isolation Level Comparison

The specific differences between OceanBase MySQL mode and MySQL database transaction isolation level are as follows:

- Read Uncommitted: The MySQL database supports Read Uncommitted, but the OceanBase does not support Read Uncommitted.

- Read Committed: The MySQL database uses semi-consistent reads when judging whether a row meets the update condition under the Read Committed isolation level. If a concurrent transaction has updated a row, the MySQL database will wait for the end of the concurrent transaction and then judge whether an update is required based on the latest version. In OceanBase, regardless of whether a concurrent transaction is updated, it always judges whether a row meets the update condition based on the version in the statement snapshot.

- Repeatable read: When a write conflict occurs under the repeatable read isolation level, the later transaction in the MySQL database will wait for the earlier transaction to end. If the earlier transaction is rolled back, the later transaction will be directly updated on the original version. The later transaction will be updated on the newly committed version if the earlier transaction is committed. In OceanBase, the later transaction will wait for the earlier transaction to end. If the earlier transaction is rolled back, the later transaction will be directly updated on the original version. If the earlier transaction is committed, the later transaction will be rolled back and an error will be returned.

- Serializability: The serializability isolation level of the MySQL database uses two-phase locking (2PL), which can ensure strict serializability; the serializability isolation level of OceanBase uses snapshot isolation (Snapshot Isolation), which cannot guarantee strict serializability.

Thank You!

OceanBase Official website:
https://oceanbase.github.io/

GitHub Discussions:
https://github.com/oceanbase/oceanbase/discussions

FROM INTRODUCTION TO PRACTICE

Lesson 6: Using OceanBase Community Edition for Business Development

Peng Wang
OceanBase Global Technical Evangelist

Agenda

- Lesson 6.1
  - Using MySQL tenants for common database development
- Lesson 6.2
  - Developed through ODC graphical development tools
  - Horizontal Split Using OceanBase Partition Table

Developed through ODC Graphical Development Tools

ODC Introduction

OceanBase Developer Center (ODC) is a database graphical development tool and a collaborative platform for data R&D and production change management.

ODC has two product forms	desktop version
	web version
The web version of ODC supports two deployment forms	single node
	high availability
ODC Web version provides two working modes	personal space
	team space

Personal space is suitable for individual developers, providing you with a desktop experience on the browser side. You can freely access, create new data sources, and use various windows and tools provided by the platform to develop the database. Team space is suitable for developers and DBAs to work together. It is both a development tool and a management and control collaboration platform. ODC provides a one-stop service for database development and control through a series of functions such as project collaboration, stable changes, data security, and separation of hot and cold data.

The ODC desktop version focuses on database development tool capabilities, supports Windows, Mac, and Linux operating systems, and is lightweight and easy to deploy. The Web version provides both tool capabilities and management and collaboration capabilities, focusing on database changes' security, compliance, and efficiency.

It is recommended that users use the Web version of ODC.

User-oriented

DBA | Application Development | Data Analysis | Business Operations | ...

SQL Development

SQL Execution
- SQL I/F
- CLI
- Script MGT
- Execution CFG

PL Lifecycle
- ANON Block I/F
- PL Run
- PL Compile
- PL Debug

Object MGT
- Tables, views
- PL objects
- Sequences, synonyms
- EVENT/JOB...

Perform Analysis
- Execution plan
- Execution analysis
- Full-link diagnosis

Data Impt. & Expt.
- Data import
- Data export
- Result set export

Data Generation
- Test data generation
- Scenario-based data generation

Database Operation and Maintenance
- Session MGT
- Recycle Bin MGT
- Global Variable MGT

Management and Control Coordination

Access Rights
- User MGT
- Data source MGT
- RBAC

Change Control
- Project Collab
- Work order APPV
- SQL DEV SPEC

Stable Changes
- Lock-free SC
- Lock-free data changes
- Structure SYN
- Structure comparison

Data Lifecycle MGT
- Data archiving
- Data cleaning
- Partition planning

Data Security CPL
- Data masking
- Operation audit
- SQL audit

Collab Efficiency
- SQL plan
- Auto Authorization
- Notification center

System Integration
- Account INTEG
- Approval INTEG
- SQL Auditing INTEG
- Bastion Host INTEG

Data Source

OceanBase EE | OceanBase CE | ODP | MySQL | ...

Product Form

Desktop Version

win | mac | linux

Web Version

x86 | arm

Cloud Services

Alibaba Cloud Public Cloud | Alibaba Cloud Private Cloud | OB Cloud

❖ *Abbreviation*

SYN: synchronization, CLI: command line interface, MGT: management, CFG: configuration, Impt.: import, Expt.: export, INTEG: Integration, DEV: development, SPEC: specification, Collab: Collaborative, I/F: Interface, SC: structural changes, CPL: Compliance

Developed through ODC Graphical Development Tools

ODC Introduction - Usage Restrictions

Data Source Types Supported by ODC	Supported Versions
OceanBase MySQL	OceanBase Community Edition
OB Sharding MySQL	ODP V3.2.8 and above
MySQL	MySQL 5.7
Oracle	Oracle 11g
Doris	Doris 2.0.0 and above

For more information about ODC function restrictions, see the Usage Restrictions section on the OceanBase official website.

Developed through ODC Graphical Development Tools

ODC Introduction-Deployment Requirements

Environmental Requirements:

1、Meta-database

- Used to store user data generated by new connections and scripts when using ODC. Before installing the Web version of ODC, you need to prepare a metadata database (OceanBase MySQL mode). The metadata database of the desktop version of ODC is local and will be automatically created when used for the first time, so there is no need to manually prepare the metadata database.
- The character set of the metadata database must be utf8mb4, otherwise, the input content will be restricted when filling in the field, such as not being able to use Emoji

2、 Install Docker, version 20.10 or above is recommended.

Web-version ODC deployment environment configuration, based on the empirical value of the number of ODC users:

Number of ODC Users	Server Type	Number of Servers	Minimum Functionality Configuration	Minimum Performance Configuration
20	ODC Docker Deployment Server	1, reusable OCP control server	2Core，4GB Mem	4Core，16GB Mem
100	ODC Docker Deployment Server	3, reusable OCP control servers	4Core，16GB Mem	8Core，32GB Mem
500	ODC Docker Deployment Server	3, reusable OCP control servers	8Core，32GB Mem	16Core，64GB Mem

Developed through ODC Graphical Development Tools

ODC Introduction-Deployment Requirements

Web version of ODC:

Environment	Require
System	•CentOS 7.2 and later. •AliOS 7.2 and later. •Huawei EulerOS 2.0 SP8.
Docker	It is recommended to use version 20.10 and later
Browser	•Chrome 76 and later. •Firefox 60 and later. •Edge 79 and later.
CPU	X86 or ARM architecture, 64-bit processor with 2 cores or above.
Memory	> 4GB

Desktop ODC:

Environment	Require
System	•Windows：Win 7/Win 10/Win Server 2016 /Win Server 2019。 •Mac：10.13.6（17G65）。 •Linux：Ubuntu 18.x/Ubuntu 20.x/UOS 1060 arm-desktop。
Java	JDK >= 1.8.0_200 and JDK < 9 Description: ODC V3.2.0 and later versions have a JRE installation package (including Mac and Win 64-bit platforms) attached to the desktop version installation package, which does not depend on the JRE of the deployment environment.
CPU	X86 64-bit processor with 2 cores or above.
Memory	> 4GB

Developed through ODC Graphical Development Tools

ODC Introduction - Web Version Deployment Demonstration

1. **Install the OceanBase metadata repository for ODC**

2. **Get the ODC image**

   Download ODC image: https://en.oceanbase.com/softwarecenter

3. **Loading the image**

   gunzip -c obodc-{$version}.tar.gz | docker load

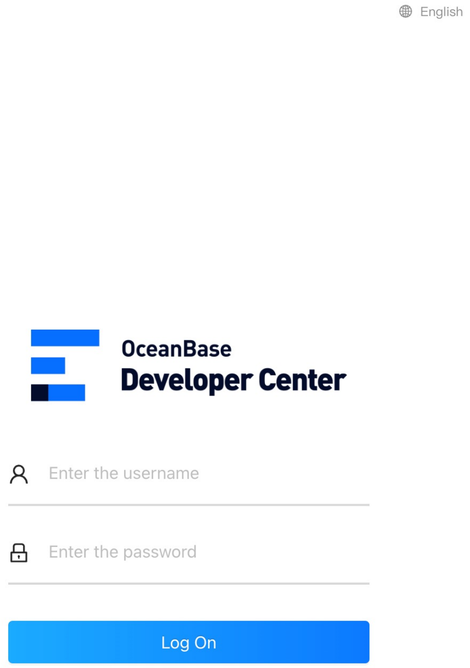
4. **Run the image**

```
#!/usr/bin/env bash
docker run -v /var/log/odc:/opt/odc/log -v /var/data/odc:/opt/odc/data \
-d -i --net host --cpu-period 100000 --cpu-quota 400000 --memory 8G --name "obodc" \
-e "DATABASE_HOST=xxx.xx.xx.xx" \
-e "DATABASE_PORT=60805" \
-e "DATABASE_USERNAME=[username]@[Tenant Name]#[Cluster Name]" \
-e "DATABASE_PASSWORD=******" \
-e "DATABASE_NAME=odc_metadb" \
-e "ODC_PROFILE_MODE=alipay" \
-e "ODC_ADMIN_INITIAL_PASSWORD=******" \
oceanbase/odc:4.2.2
```

Deploy ODC Official Docs: https://en.oceanbase.com/docs/common-odc-10000000001297401

Developed through ODC Graphical Development Tools

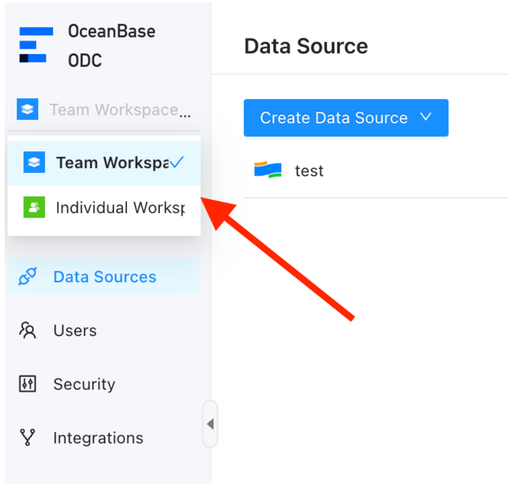
ODC Introduction - Web Version Deployment Demonstration

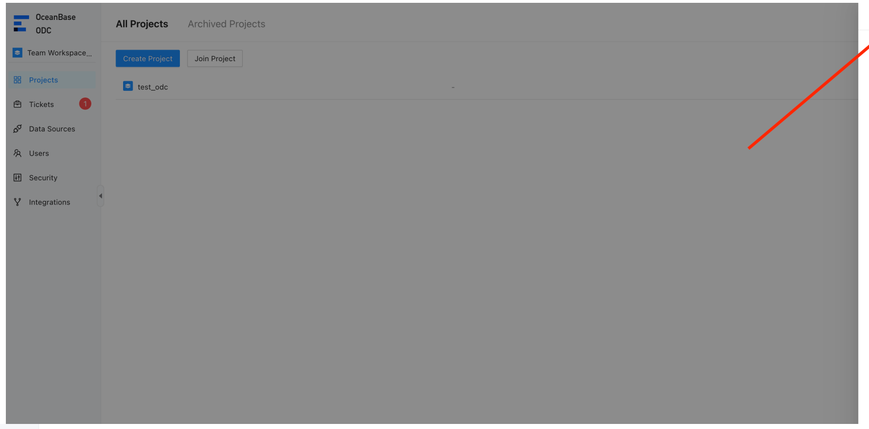


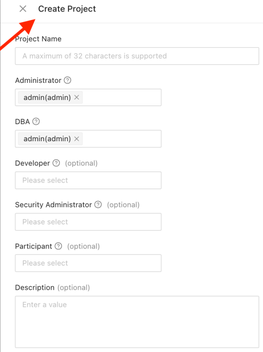
FROM INTRODUCTION TO PRACTICE

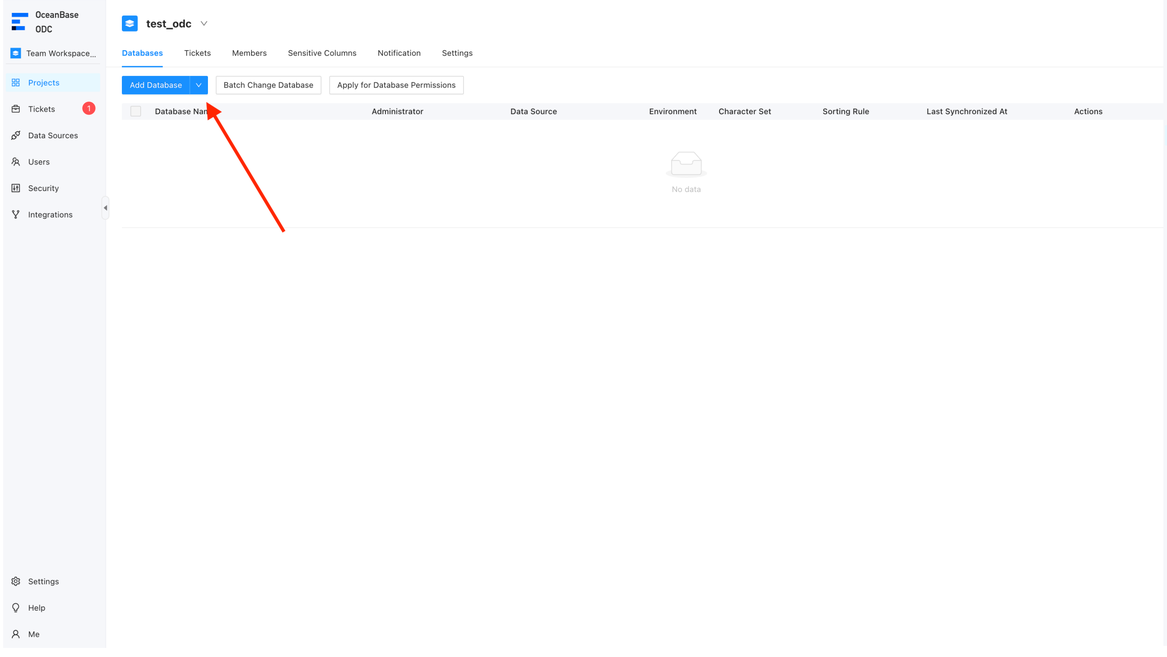
Developed through ODC Graphical Development Tools

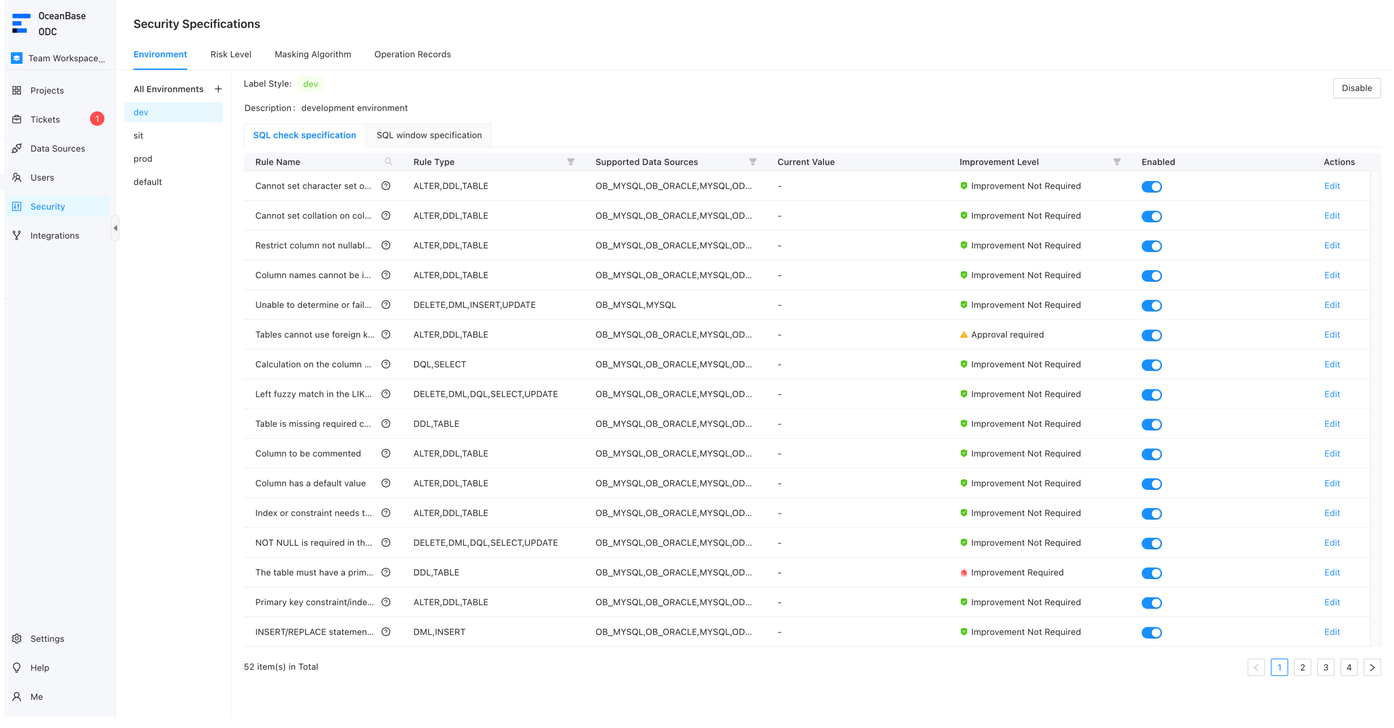
ODC Tools - Management and Control Collaboration





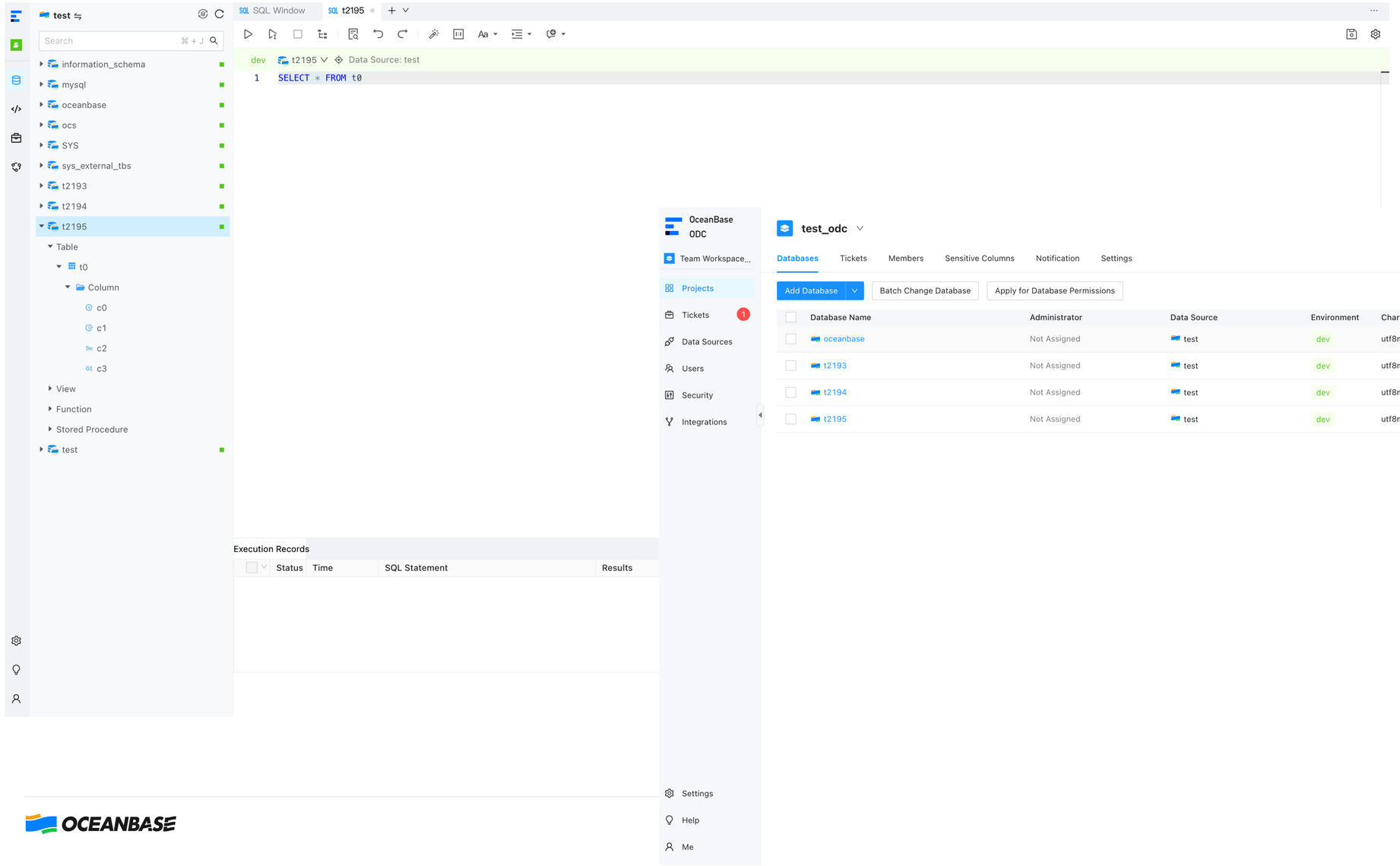






Developed through ODC Graphical Development Tools

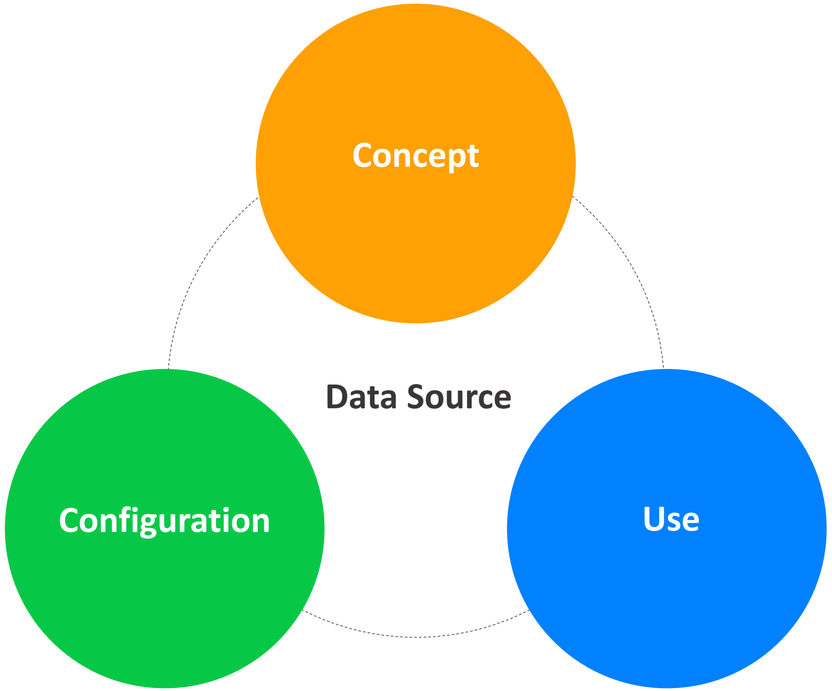
ODC Tools-Basic Tool Capabilities

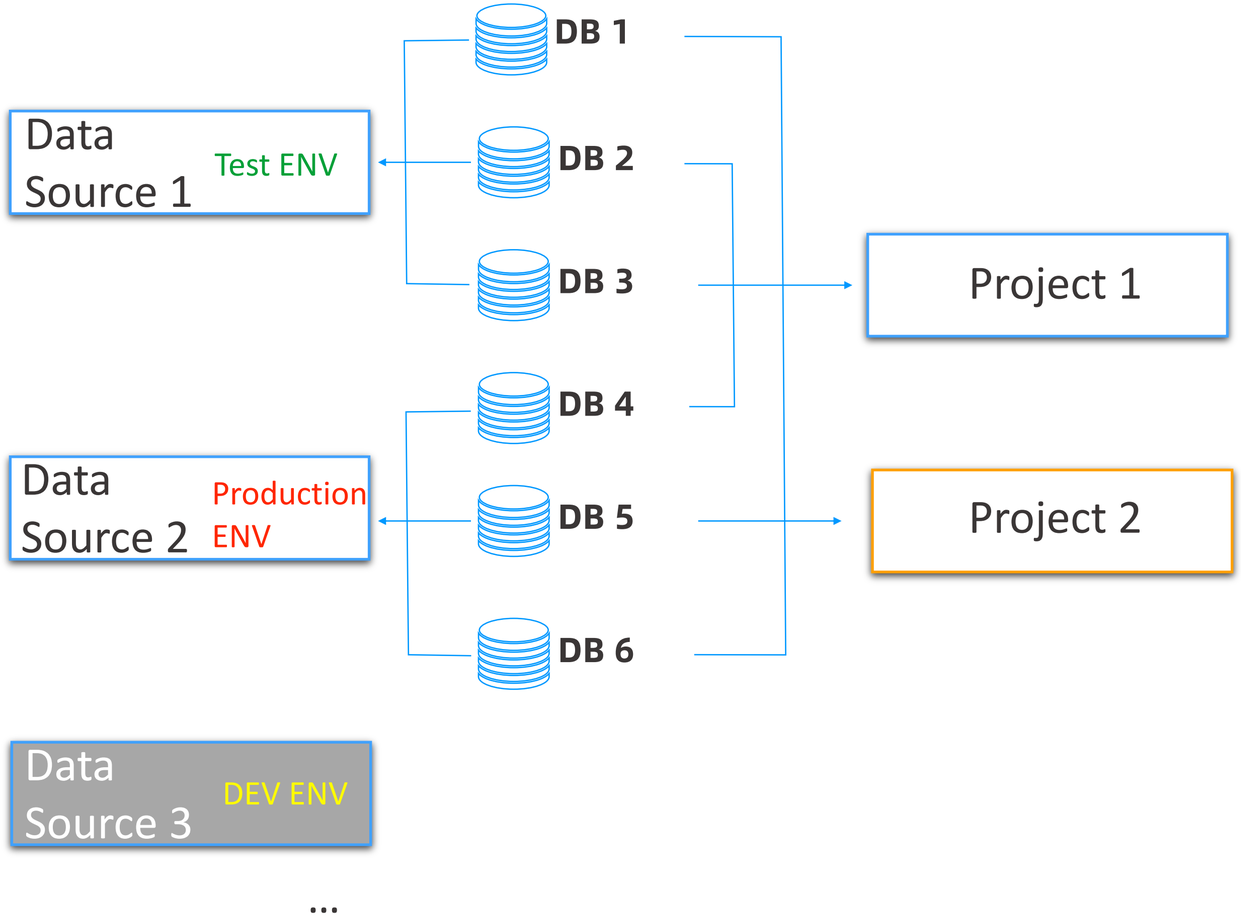


Developed through ODC Graphical Development Tools

ODC Tools - Data Sources - Concepts

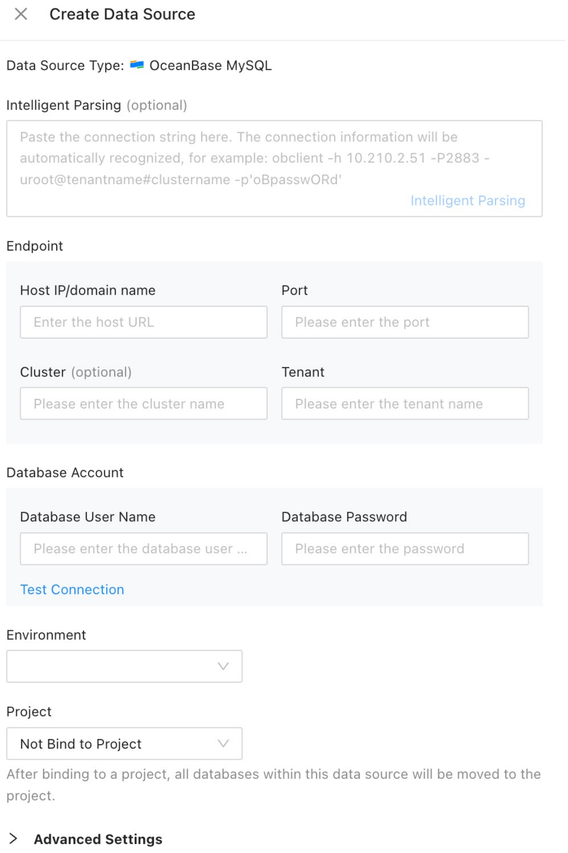
A data source refers to a remote database environment, such as an independently deployed OB cluster or other database system instance.





Developed through ODC Graphical Development Tools

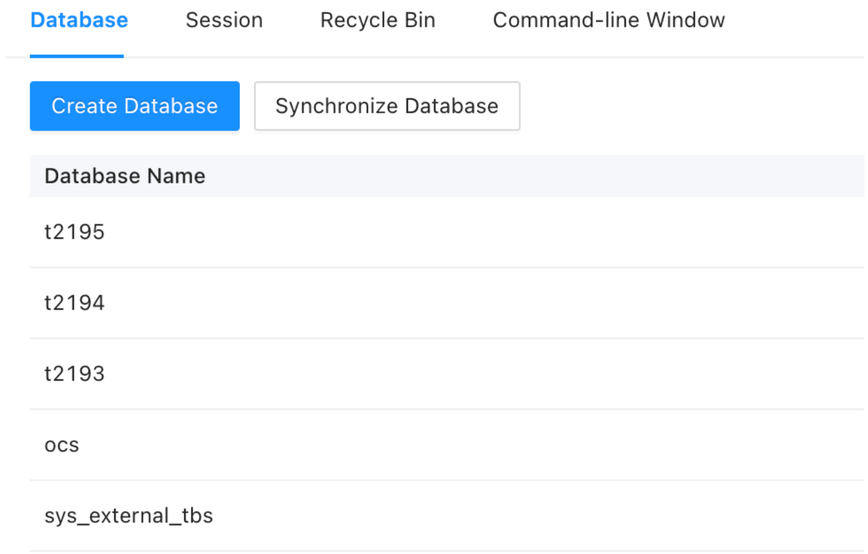
ODC Tools - Data Source - Configuration



- Supported data source types: OB-MySQL, OB-Oracle, OB-MySQL-Cloud, OB-Oracle-Cloud, ODP-Sharding-MySQL, MySQL
- Main configurable information
  - Host IP: IP address or domain name of the data source
  - Port: port number of the data source
  - Database username: database username
  - Database password: database user password
- OceanBase unique options: cluster name, tenant name
- OceanBase-Cloud special features: does not contain cluster name, tenant name, host IP, must fill in the domain name where the data source is located
- Environment: used to configure the operating environment to which the data source belongs, such as development environment, production environment, etc., used to configure different management and control strategies

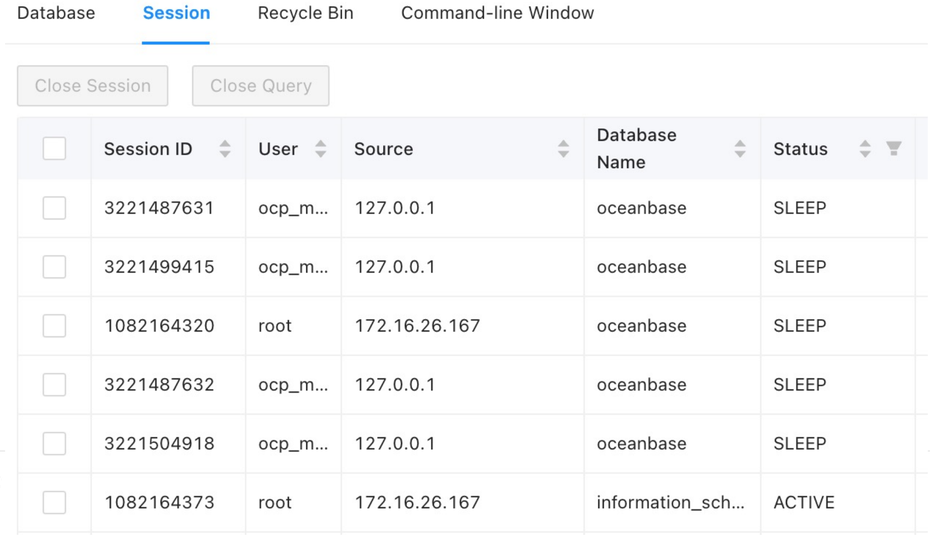
Developed through ODC Graphical Development Tools

ODC Tools - Data Sources - Usage



Managing Databases

- Entry: Left navigation bar -> Data source -> Data source xxx -> Database
- Function
  - Create a new database
  - Assign the database to the project

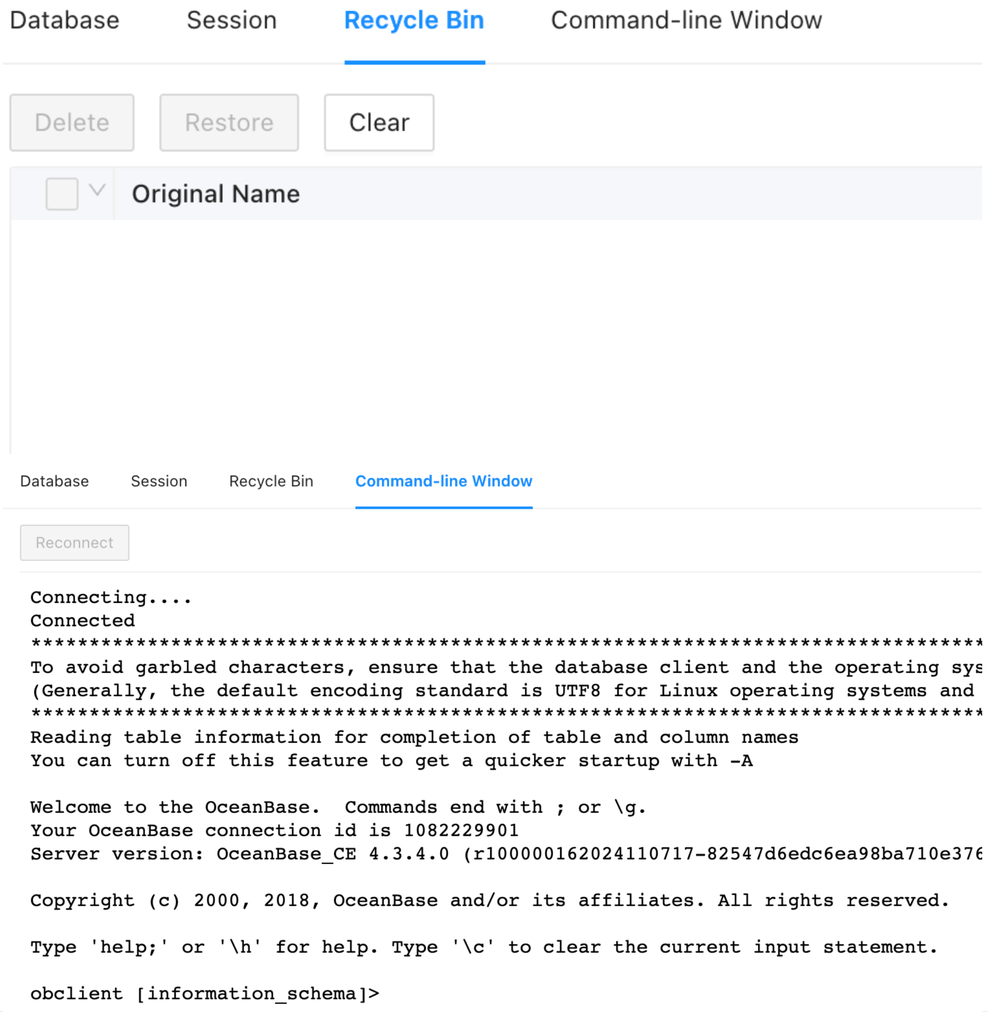


Managing Data Source Sessions

- Entry: Left navigation bar -> Data source -> Data source xxx -> Session
- Function
  - View all sessions under this data source
  - Close a session
  - Close the query running on a session

Developed through ODC Graphical Development Tools

ODC Tools - Data Sources - Usage



Recycle Bin

- Entry: Left navigation bar -> Data source -> Data source xxx -> Recycle Bin
- Function
  - Restore deleted tables, views, and other database objects
  - Empty the Recycle Bin and delete a specific object in the Recycle Bin
- **Note**: This feature is tenant-level, please enable it with caution

Command Line Interface

- Entry: Left navigation bar -> Data source -> Data source xxx -> Command line window
- Function
  - Connect to the target data source via OBClient


Database DEV
• traceId
• Execution Record
• Execution Time
• Execution Details
• Execution Plan
Execution Details and Diagnostics
Database Object Management
SQL DEV
Result Set Viewing and Editing
SQL Editing and Execution
• Data Editing
• Data View
• Result Set Export
• Data Replication
• Large Field View
Database MGT
• Table Object
• View Object
• PL Object
• Synonyms
• Syntax Highlighting
• Smart Tips
• SQL Script
• SQL Examine
• Code Beautification

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - SQL Editing and Execution

Syntax Highlighting/Code Hinting SQL Checking

Provide professional code highlighting and prompting services, SQL beautification makes the structure clear at a glance.
Dozens of rules form a reliable expert system to help users write good SQL and avoid risks

SQL Script

Provides custom SQL script storage and code snippet functions to efficiently manage commonly used SQL.

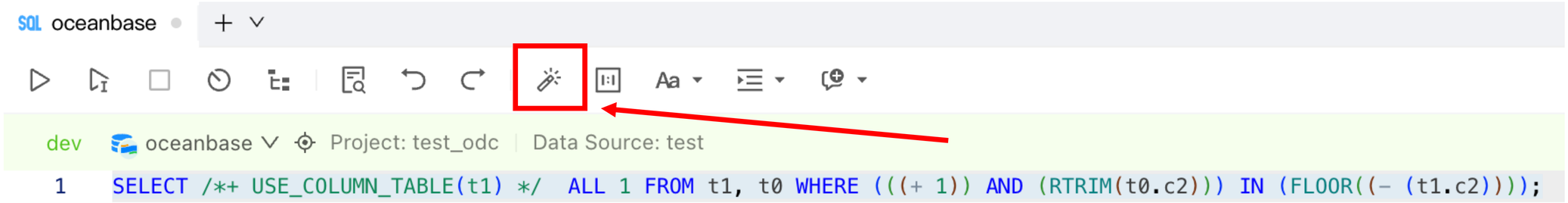
Result Set Display

Clear interface, rich display methods, excellent performance when there is a lot of data, smooth experience.

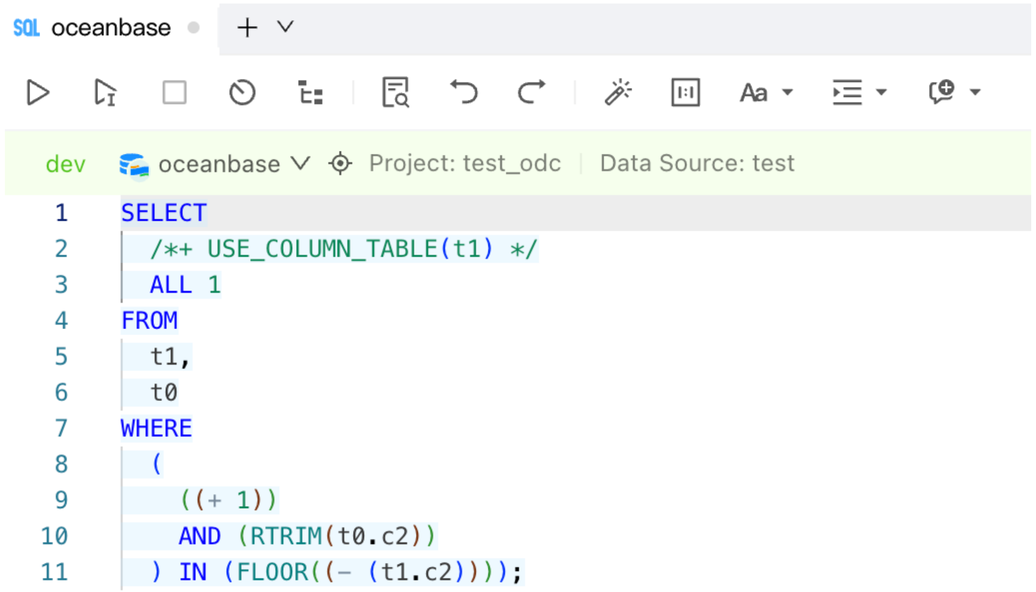
Provide a full range of services for your SQL editing and execution

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - Syntax Highlighting and Formatting



One Click Format



Syntax Highlighting

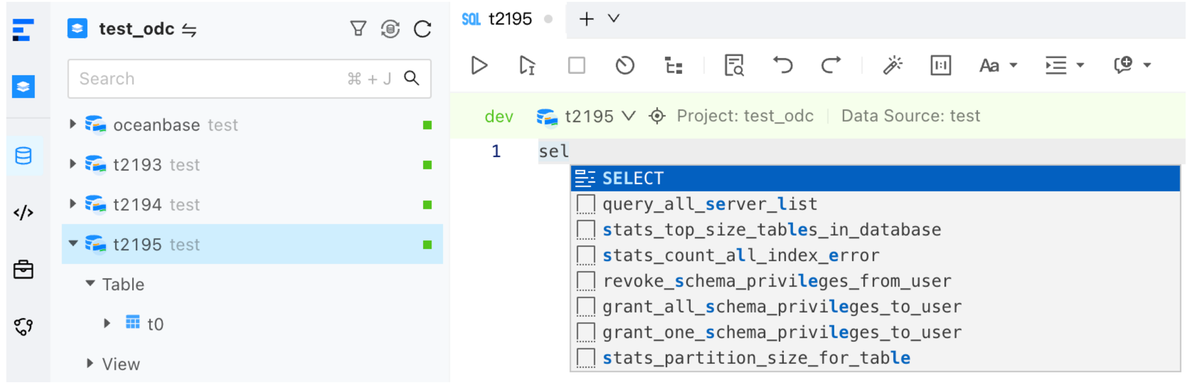
- Keyword highlighting allows you to grasp key information at a glance

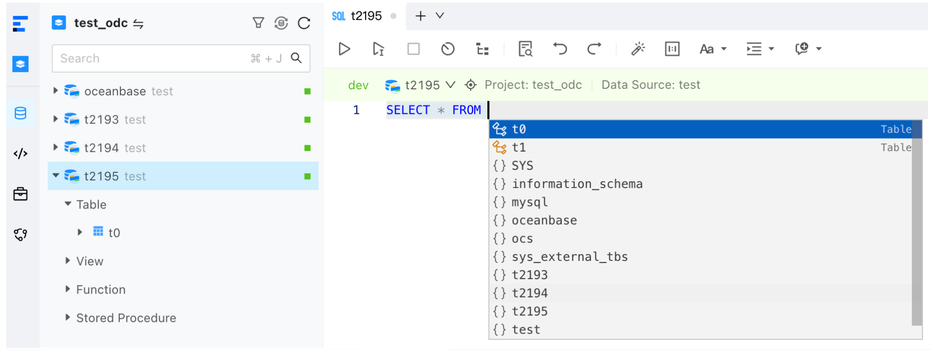
SQL Format

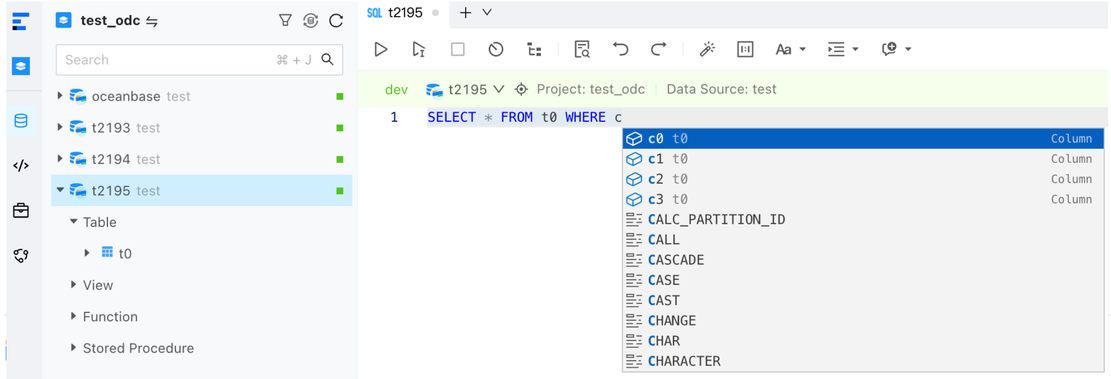
- Format the code according to the grammatical structure, making the user's SQL pleasing to the eye
- Expand according to the grammatical structure, making the user's SQL structure clear at a glance and easier to understand

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - Syntax Tips







Keyword Tips

- Keyword tips, so that users no longer have to worry about misspelling keywords

Database object name tips

- Database object name prompts allow users to match the target in the vast database object list quickly

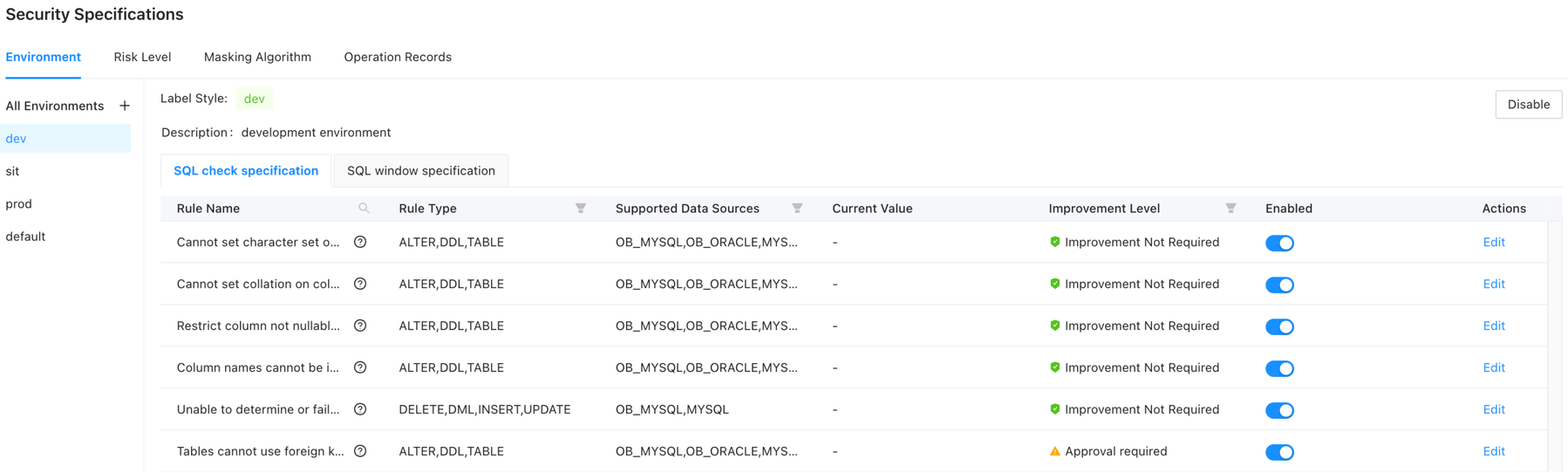
Column Name Hint

- Intelligent column name prompts, which can intelligently give column suggestions based on the table the user is currently operating

FROM INTRODUCTION TO PRACTICE

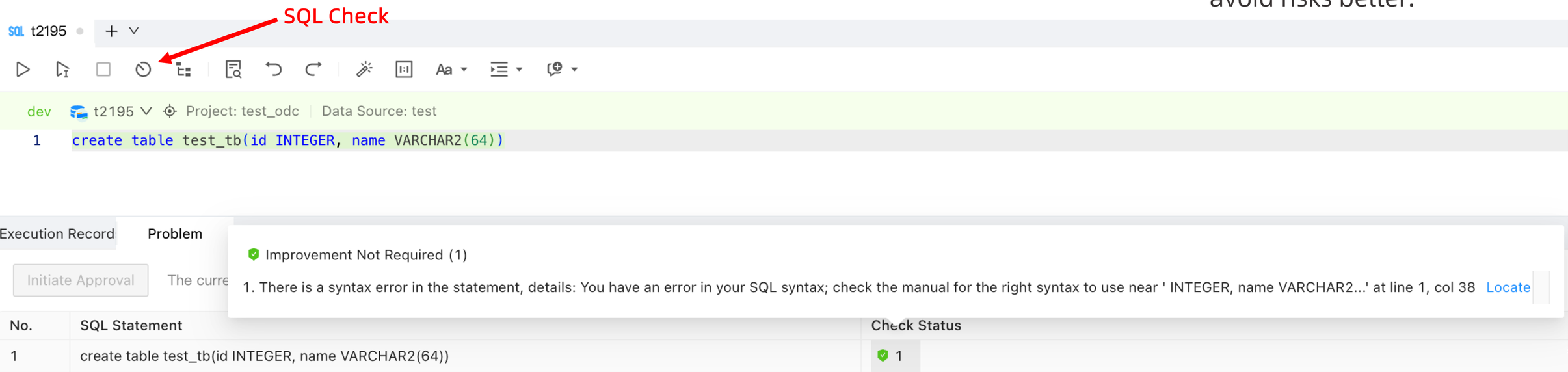
Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - SQL Check



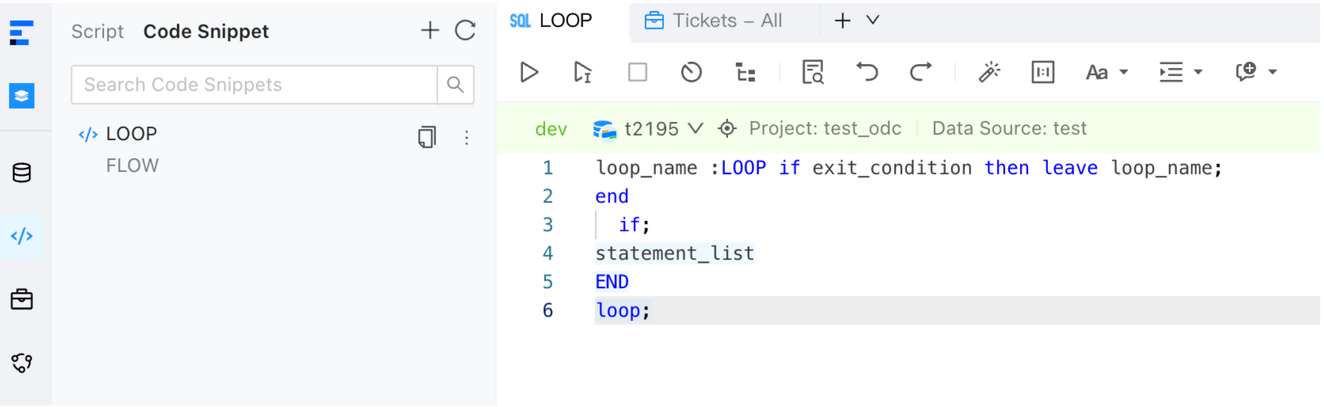
SQL Check

- There are many rules and they cover a wide range of areas.
- The rules are based on OceanBase's best practices and have high reference value.
- Detection is convenient and can be done with one click.
- Intelligent risk level prompts help you avoid risks better.



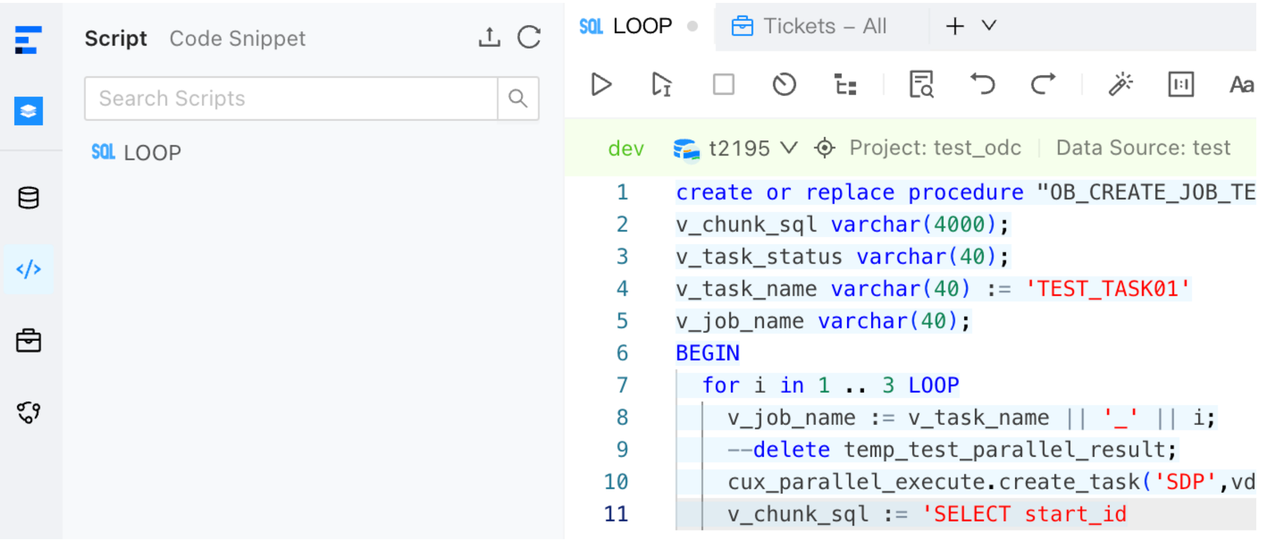
Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - SQL Scripts



Code Snippet

- Save commonly used syntax blocks, such as loop control, branch control
- Drag to recall, convenient and fast operation
- Save the trouble of remembering many SQL/PL syntax structures

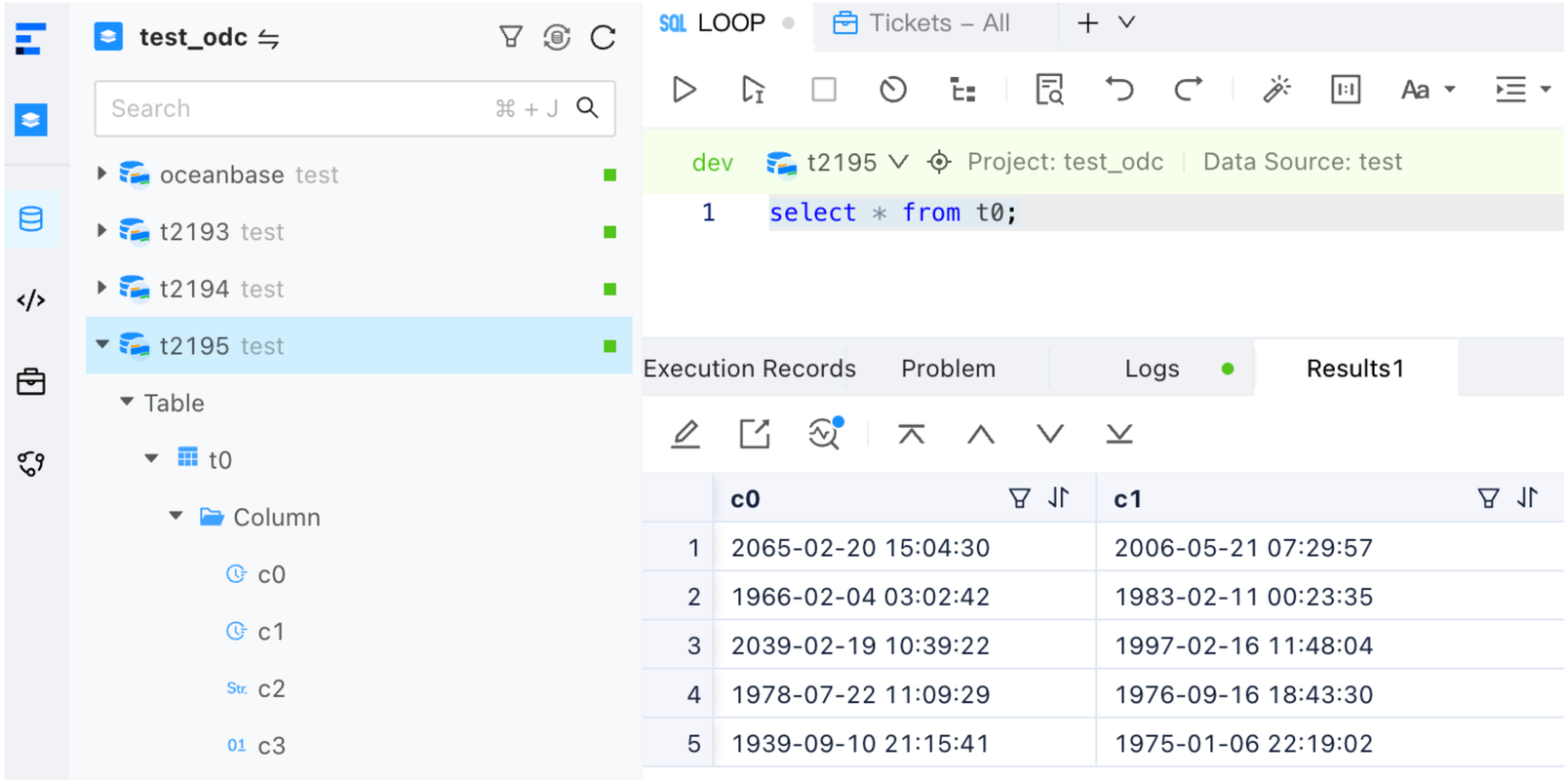


SQL Script

- Save your frequently used SQL scripts, such as frequently used anonymous blocks and frequently used query logic.
- Efficient retrieval helps users manage many frequently used SQL scripts.

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - Result Set Display

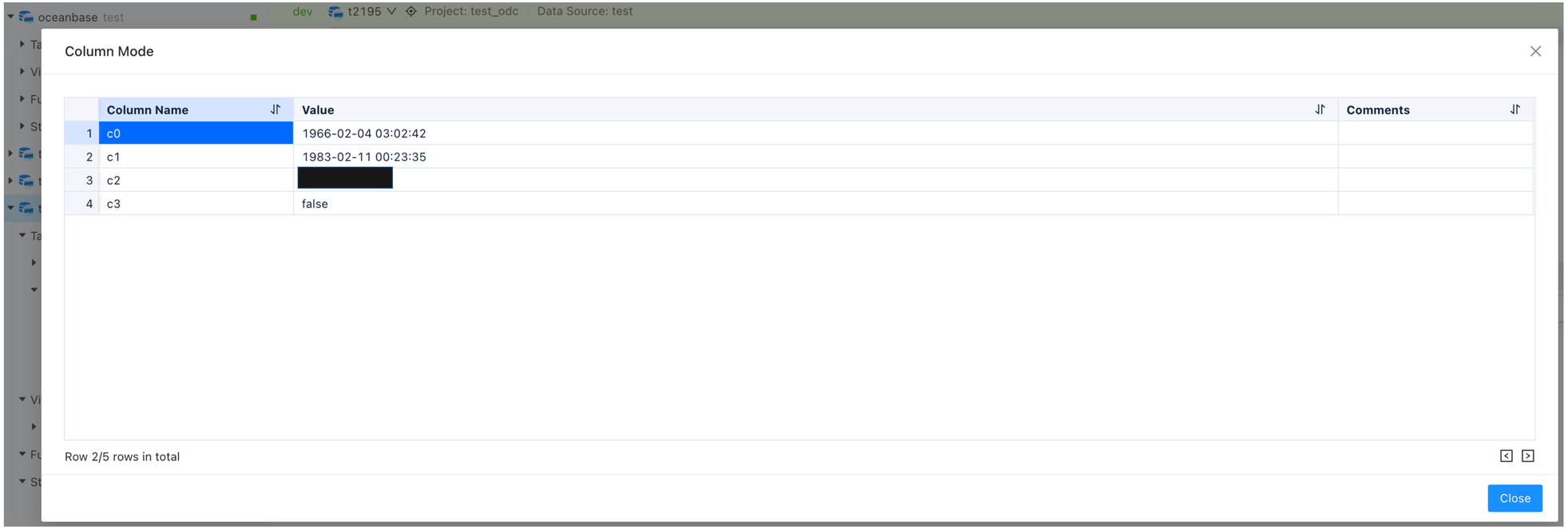


Data display is efficient and professional

- Table display, clear and beautiful, data at a glance
- Ten thousand data are displayed smoothly, without lag or frame drop
- Data of different data types are displayed in a more professional way that is more in line with the habits of database developers
- Freeze rows, multiple rows, multiple columns selection
- Conversion of result sets to SQL, CSV

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - Result Set Display

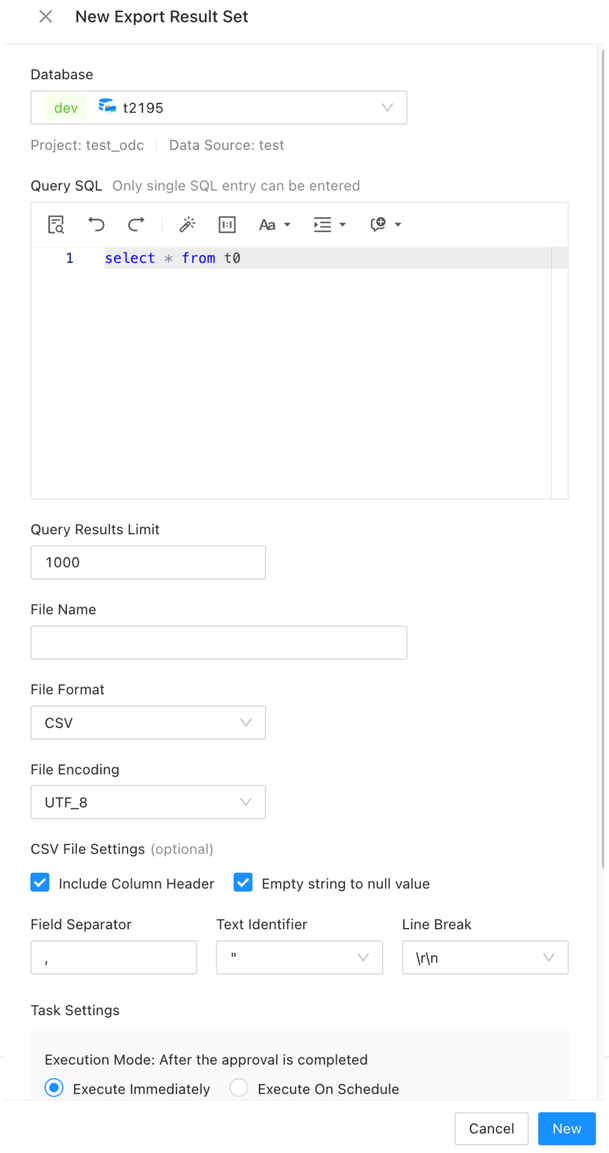


Personalization

- Display data in row mode, allowing users to get an overview of the overall situation.
- Display data in column mode, allowing users to focus on details.
- Large field data can be displayed in text, hexadecimal, or image mode, allowing users to restore the true appearance of the data.

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - Result Set Export and Editing



Result Set Editing

- Edit the selected row and modify it accurately
- Customize different modification methods for different data types to modify data more efficiently
- Support multi-row modification, and one-click submission, what you see is what you get

Result Set Export

- Export SQL execution results
- Support multiple formats: CSV, EXCEL, SQL
- Support multiple encoding formats

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - SQL Execution Plan and Details

Execution time statistics

- Time statistics for each executed SQL statement
- Segmented statistics to let users know the stages of SQL execution by ODC
- The kernel execution time is also included so that users can know the specific time without checking SQL-Audit
- Record the trace-id of each SQL statement to make troubleshooting easier

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - SQL Execution Plan and Details

Execution Profile for Trace ID "YB427F000001-000628CD3C88183E-0-0"

SQL: select * from t0

Execution Details | Execution Plan | End-to-end Trace Diagnostics | Search Keywords | Export JSON

Span ID	Span	Node	Time Started	Duration
000628d1-6ae4-d423-6988-ca231129dced	com_query_process	OBServer	2024-12-09 15:36:06.921251	1.51 ms
000628d1-6ae4-d427-5f0c-cc46a2e95264	mpquery_single_stmt	OBServer	2024-12-09 15:36:06.921255	1.49 ms
000628d1-6ae4-d42b-ca14-51d3e6309973	sql_compile	OBServer	2024-12-09 15:36:06.921259	1.12 ms
000628d1-6ae4-d42f-7d52-38bee13ba808	pc_get_plan	OBServer	2024-12-09 15:36:06.921263	3 us
000628d1-6ae4-d45d-3488-4a8c4e312652	hard_parse	OBServer	2024-12-09 15:36:06.921309	1.06 ms
000628d1-6ae4-d45e-f6fa-3b3c5f6325ca	parse	OBServer	2024-12-09 15:36:06.921310	28 us
000628d1-6ae4-d4a7-76a3-8748849f26cf	resolve	OBServer	2024-12-09 15:36:06.921383	121 us
000628d1-6ae4-d549-ecdd-d84f44cb0b00	rewrite	OBServer	2024-12-09 15:36:06.921545	218 us
000628d1-6ae4-d638-434e-76e767669ac4	optimize	OBServer	2024-12-09 15:36:06.921784	313 us
000628d1-6ae4-d78e-a48d-ca3966e55462	code_generate	OBServer	2024-12-09 15:36:06.922126	79 us
000628d1-6ae4-d847-2f35-55c0c31b6fcd	pc_add_plan	OBServer	2024-12-09 15:36:06.922311	38 us
000628d1-6ae4-d89e-af60-47525da73ab6	sql_execute	OBServer	2024-12-09 15:36:06.922398	301 us
000628d1-6ae4-d89f-7e7f-07fdb9c7126d	open	OBServer	2024-12-09 15:36:06.922399	17 us
000628d1-6ae4-d8bd-df78-1b5459263743	response_result	OBServer	2024-12-09 15:36:06.922429	128 us
000628d1-6ae4-d8c2-4e86-4509267e0e01	do_local_das_task	OBServer	2024-12-09 15:36:06.922434	46 us
000628d1-6ae4-d94c-a5b0-697ed3af5da8	close	OBServer	2024-12-09 15:36:06.922572	74 us
000628d1-6ae4-d94d-3999-52da3f75781c	close_das_task	OBServer	2024-12-09 15:36:06.922573	11 us
000628d1-6ae4-d97a-c2bb-13d23d669d69	end_transaction	OBServer	2024-12-09 15:36:06.922618	2 us

Implementation Details

- Display SQL trace-id, time consumption, execution plan and other information

Full-link Diagnosis

- OB 4.2 introduced, ODC quickly compatible
- Fully display the entire process of SQL execution on the entire link of driver, ob-proxy, and observer to help users better troubleshoot problems

Developed through ODC Graphical Development Tools

ODC Tools - SQL Development - Object Management

Multiple Database Objects

ODC can manage various types of database objects, including tables, views, functions, stored procedures, packages, triggers, types, sequences, and synonyms.

Database Object MGT

ODC provides various database object management operations, including basic operations such as adding, deleting, modifying, querying, and some special operations.

Create Database Objects on GUI

The GUI capability is provided to help users create database objects at low cost, reducing the difficulty of using the database.

For detailed management functions, please refer to the “Database Objects” section:
https://en.oceanbase.com/docs/common-odc-10000000001297510

Developed through ODC Graphical Development Tools

ODC Tools - Benefits Summary

Product Form

Free and open

Combines development and management capabilities

- It is the only free, open-source development platform on the market that integrates database development and database management and control collaborative capabilities.
- Compared with other management and control collaboration platforms, ODC not only provides management and control capabilities but also provides rich development capabilities.
- Compared with other development tools, ODC not only provides tool capabilities but also provides safe and compliant management and control capabilities.
- At the same time, it supports the free switching between personal development and team collaboration, which is flexible and efficient.

Tool Orientation

Stable and reliable

Sound and easy to use

- ODC tool interaction focuses on ease of use, and its ease of use has been recognized by a large number of users.
- ODC tool capabilities currently cover 90% of common customer scenarios.

MGT and Control Coordination

R&D and operation integration work concept
Provide comprehensive security and compliance solutions

- ODC combines the work concept of integrated R&D and operation and takes projects as the core for management and control collaboration. While ensuring management and control compliance, it simplifies the management and control collaboration process and improves work efficiency.
- In addition to change control, ODC also provides compliance solutions such as R&D specifications, data desensitization, lock-free changes, and data archiving.

Agenda

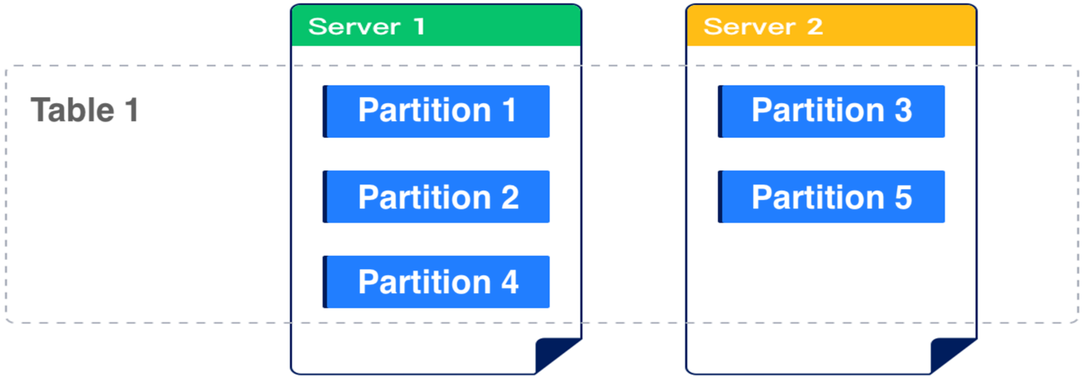
- Lesson 6.1
  - Using MySQL tenants for common database development
- Lesson 6.2
  - Developed through ODC graphical development tools
  - Horizontal Split Using OceanBase Partition Table

Using Oceanbase Partition Table for Horizontal Splitting

Partition Table - Function Definition

Partition Table:

In OceanBase database, partitioning means breaking a table into multiple smaller and more manageable parts according to certain rules. Each partition is an independent object with its own name and optional storage characteristics. 。



For applications accessing the database, there is only one table or one index in the logical sense, but in reality, the table may consist of dozens of physical partition objects, each of which is an independent object that can be accessed independently or as part of a table. Partitions are completely transparent to the application and do not affect the business logic of the application.

From the application's perspective, only one Schema object exists. Accessing partitioned tables does not require modifying SQL statements. Partitioning is very useful for many different types of database applications, especially those that manage large amounts of data.

Using Oceanbase Partition Table for Horizontal Splitting

Partition Table – Introduction to Business Splitting

Benefits of using partition tables:

- Improved usability

A partition being unavailable does not mean that the object is unavailable. The query optimizer automatically removes unreferenced partitions from the query plan. Therefore, queries are not affected when a partition is unavailable.

- Easier object management

Partition objects have fragments that can be managed collectively or individually. DDL statements can operate on partitions instead of entire tables or indexes. Therefore, resource-intensive tasks such as rebuilding indexes or tables can be broken down. For example, you can move only one partition at a time. If a problem occurs, only the partition move needs to be redone, not the table move. In addition, TRUNCATE operations on partitions can avoid large amounts of data being deleted.

- Reduce contention for shared resources in OLTP systems

In TP scenarios, partitioning can reduce contention for shared resources. For example, DML is distributed across many partitions instead of one table.

- Improving query performance in data warehouses

In the AP scenario, partitioning can speed up the processing of instant queries. Partition keys have a natural filtering function. For example, to query sales data for a quarter, when the sales data is partitioned by sales time, you only need to query one or several partitions instead of the entire table.

- Provide better load-balancing effect

The storage unit and load balancing unit of the OceanBase database are partitions. Different partitions can be stored on different nodes. Therefore, a partition table can distribute different partitions on different nodes, so that the data of a table can be distributed more evenly across the entire cluster.

Using Oceanbase Partition Table for Horizontal Splitting

Partition Table – Notes

Notes on creating partition tables

- If the amount of data is large and the access is concentrated, you can use a partition table when creating the table. Generally, partitioning can be considered for data volumes above 10 million, and the upper limit of the number of partitions is 8192.
- Partition table constraint considerations.
  - When creating a partitioned table, at least one field corresponding to each primary key and unique key on the table must be included in the partition key field of the table. If the table does not have a primary key or unique key, the partition key can be any field.
  - It is recommended that global uniqueness in a partitioned table be achieved through the primary key if it can be achieved through the primary key.
  - The unique index of a partitioned table must include the split key of the table partition

- Regarding the partitioning strategy, it is recommended to design it based on the actual purpose and application scenarios of the table.
  - Practical use: historical table, flow table.
  - Application scenario: tables with obvious access hotspots.

- Regarding the selection of partition keys, when using partition tables, you should choose appropriate split keys and split strategies.
  - HASH partitioning: Select a field with high discrimination and the highest frequency of occurrence in the query conditions as the partition key for HASH partitioning.
  - RANGE and LIST partitioning: Select appropriate fields as partition keys according to business rules, but the number of partitions should not be too small. Example: If it is a large log-type table, use RANGE partitioning based on the time type column.
  - RANGE partitioning: It is not recommended to set the last column to MAXVALUE.
  - Under HASH partitioning, it is not suitable to perform range queries based on partition fields

Using Oceanbase Partition Table for Horizontal Splitting

Partition Table - Create A Partition Table

The partition table of the OceanBase database is a built-in function. You only need to specify the partition strategy and number of partitions when creating the table. The query SQL of the partition table is the same as that of the ordinary table. OceanBase's OBProxy or OBServer will automatically route the user SQL to the corresponding node. Therefore, the partition details of the partition table are transparent to the business.

If you know the partition number of the data you want to read, you can directly access a partition of the partition table through SQL. The simple syntax format is as follows:
part_table partition (p[0,1,...][sp[0,1,...]])

Except for the special case where the table defines partition names, by default, partition names are numbered according to certain rules, such as:
First-level partition names: p0, p1, p2, ...
Second-level partition names: p0sp0, p0sp1, p0sp2, ... ; p1sp0, p1sp1, p1sp2, ...

Example: Accessing a specific partition of a partitioned table

```
select * from t1 partition (p0) ;

select * from t1 partition (p5sp0) ;
```

Partitioning Strategy:

- Range partitioning
- RANGE COLUMNS partitioning
- List partitioning
- LIST COLUMNS partitioning
- HASH
- Key partitioning
- Combined partitioning

Using Oceanbase Partition Table for Horizontal Splitting

Partitioned Tables – RANGE Partitions (I)

RANGE partitioning maps data to the corresponding partition according to the partition key value range established for each partition when the partition table is defined. It is a common partitioning type and is often used with the date type. For example, a business log table can be partitioned by day/week/month.

The syntax of RANGE partitioning

```
CREATE TABLE table_name (
    column_name1        column_type
    [, column_nameN     column_type]
) PARTITION BY RANGE ( expr(column_name1) )
(
    PARTITION   p0      VALUES LESS THAN ( expr )
    [, PARTITION pN     VALUES LESS THAN (expr ) ]
 [, PARTITION pX    VALUES LESS THAN (maxvalue) ]
);
```

The example of RANGE partitioning

```
CREATE TABLE test_range(id INT, gmt_create TIMESTAMP, info VARCHAR(20), PRIMARY KEY (gmt_create))
PARTITION BY RANGE(UNIX_TIMESTAMP(gmt_create))
(PARTITION p0 VALUES LESS THAN (UNIX_TIMESTAMP('2015-01-01 00:00:00')),
PARTITION p1 VALUES LESS THAN (UNIX_TIMESTAMP('2016-01-01 00:00:00')),
PARTITION p2 VALUES LESS THAN (UNIX_TIMESTAMP('2017-01-01 00:00:00')));
```

Using Oceanbase Partition Table for Horizontal Splitting

Partitioned Tables - RANGE Partitions (II)

RANGE partitioning rules:

- The result of the expr expression in PARTITION BY RANGE (expr) must be an integer.
- Each partition has a VALUES LESS THAN clause that specifies a non-inclusive upper bound for the partition. Partition key values equal to or greater than this upper bound are mapped to the next partition.
- All partitions except the first have an implicit lower bound, which is the upper bound of the previous partition.
- The last partition upper bound is allowed to be defined as MAXVALUE, which has no specific value and is greater than the upper bounds of all other partitions, including null values.

Note:

RANGE partitions can add and delete partitions. If the last RANGE partition specifies MAXVALUE, no new partitions can be added. Therefore, it is recommended not to use MAXVALUE to define the last partition.

RANGE partitioning requires that the result of the table split key expression must be an integer. If you want to perform RANGE partitioning by time type column, you must use the timestamp type and use the UNIX_TIMESTAMP function to convert the time type to a numeric value. This requirement can also be achieved using RANGE COLUMNS partitioning, which has no integer restriction.

Using Oceanbase Partition Table for Horizontal Splitting

Partitioned Tables – RANGE COLUMNS Partitions

RANGE COLUMNS partitioning is similar to RANGE partitioning, but the difference is that RANGE COLUMNS partitioning can be partitioned by one or more partition key vectors, and the type of each partition key can support other types besides INT, such as VARCHAR, DATETIME, etc.

The difference between RANGE COLUMNS and RANGE:

- RANGE COLUMNS partitions do not require INT type, but can be of any type
- RANGE COLUMNS partitions cannot contain expressions
- RANGE COLUMNS partitions support vectors

The syntax of RANGE COLUMNS partitioning is as follows:

```
CREATE TABLE table_name (column_name column_type[, column_name column_type])
  PARTITION BY { RANGE COLUMNS(column_name [,column_name])
                }
    (
     PARTITION partition_name VALUES LESS THAN(expr)
     [, PARTITION partition_name VALUES LESS THAN (expr )...]
     [, PARTITION partition_name VALUES LESS THAN (MAXVALUE)]
     );
```

Example:

```
CREATE TABLE tbl1_log_rc (log_id BIGINT NOT NULL,log_value VARCHAR(50),log_date DATE NOT NULL)
       PARTITION BY RANGE COLUMNS(log_date)
        (PARTITION M202001 VALUES LESS THAN('2020/02/01')
       , PARTITION M202002 VALUES LESS THAN('2020/03/01')
       , PARTITION M202003 VALUES LESS THAN('2020/04/01')
       , PARTITION M202004 VALUES LESS THAN('2020/05/01')
       , PARTITION M202005 VALUES LESS THAN('2020/06/01')
       , PARTITION M202006 VALUES LESS THAN('2020/07/01')
       , PARTITION M202007 VALUES LESS THAN('2020/08/01')
       , PARTITION M202008 VALUES LESS THAN('2020/09/01')
       , PARTITION M202009 VALUES LESS THAN('2020/10/01')
       , PARTITION M202010 VALUES LESS THAN('2020/11/01')
       , PARTITION M202011 VALUES LESS THAN('2020/12/01')
       , PARTITION M202012 VALUES LESS THAN('2021/01/01')
       , PARTITION MMAX VALUES LESS THAN MAXVALUE
        );
Query OK, 0 rows affected
```

Using Oceanbase Partition Table for Horizontal Splitting

Partition Table – LIST Partition

LIST partitions are divided according to the value of the enumeration type and are mainly used for enumeration types.

LIST partitioning can explicitly control how rows are mapped to partitions by specifying a list of discrete values for the partition key of each partition, which is different from RANGE partitioning and HASH partitioning. The advantage of LIST partitioning is that it can easily partition unordered or unrelated data sets.

LIST partition rules:

- The partition key can be a column name or an expression. The partition key must be of INT type or an expression that returns INT type.
- A partition expression can only reference one column and cannot have multiple columns (i.e., column vectors). For example, partition by list (c1, c2) is not allowed.

The syntax format of LIST partition is as follows:

```
CREATE TABLE table_name (column_name column_type[,column_name column_type])
  PARTITION BY { LIST ( expr(column_name) | column_name )
                 }
    (PARTITION partition_name VALUES IN ( v01 [, v0N])
     [,PARTITION partition_name VALUES IN ( vN1 [, vNN])]
     [,PARTITION partition_name VALUES IN (DEFAULT)]
    );
```

Example:

```
CREATE TABLE tbl1_l (col1 BIGINT PRIMARY KEY,col2 VARCHAR(50))
       PARTITION BY LIST(col1)
        (PARTITION p0 VALUES IN (1, 2, 3),
         PARTITION p1 VALUES IN (5, 6),
         PARTITION p2 VALUES IN (DEFAULT)
        );
Query OK, 0 rows affected
```

Using Oceanbase Partition Table for Horizontal Splitting

Partitioned Tables – LIST COLUMNS Partitions

LIST COLUMNS partitioning is an extension of LIST partitioning. It supports multiple partitioning keys and supports INT data, DATE type, and DATETIME type.
If you want to use LIST partitioning with multiple columns or LIST partitioning with other data types, you can use LIST COLUMNS partitioning.

The difference between LIST COLUMNS and LIST:

- LIST COLUMNS partitions do not require INT type, but can be of any type
- Expressions cannot be written in LIST COLUMNS partitions
- LIST COLUMNS partitions support vectors

The syntax format of the LIST COLUMNS partition is as follows:

```
CREATE TABLE table_name (column_name column_type[,column_name column_type])
  PARTITION BY { LIST COLUMNS ( column_name [,column_name])
                 }
    (PARTITION partition_name VALUES IN ( v01 [, v0N])
     [,PARTITION partition_name VALUES IN ( vN1 [, vNN])]
     [,PARTITION partition_name VALUES IN (DEFAULT)]
    );
```

Example:

```
CREATE TABLE tbl1_lc (id INT,partition_id VARCHAR(2))
       PARTITION BY LIST COLUMNS(partition_id)
        (PARTITION p0 VALUES IN ('00','01'),
         PARTITION p1 VALUES IN ('02','03'),
         PARTITION p2 VALUES IN (DEFAULT)
        );
Query OK, 0 rows affected
```

Using Oceanbase Partition Table for Horizontal Splitting

Partitioned Table – HASH Partition

HASH partitioning is suitable for scenarios where the RANGE partitioning and LIST partitioning methods cannot be used. The implementation method of HASH partitioning is simple, by hashing the value of the HASH function on the partition key and recording it in different partitions. Usually used for point queries with a given partition key, such as partitioning by user ID. HASH partitioning can usually eliminate hotspot queries.

If the data in the table meets the following characteristics, using HASH partitioning is a good choice:

- You cannot specify the list characteristics of the partition key of the data.
- The data size in different ranges varies greatly and is difficult to manually adjust the balance.
- The data is severely clustered after using RANGE partitioning.
- The performance of parallel DML, partition pruning, and partition joins is very important.

HASH partitioning rules:

- The partition key is of INT type, or an expression that returns INT type.
- Vectors such as partition by hash(c1, c2) cannot be written.
- If the partition name is not specified when creating Hash partitions, the partition is named by the system according to the naming rules. Each partition is named p0, p1, ..., pn for a first-level partition table.
- HASH partitions do not support adding or deleting partitions.

The syntax format of HASH partition is as follows:

```
CREATE TABLE table_name (column_name column_type[,column_name column_type])
  PARTITION BY { HASH(expr) }
    PARTITIONS partition_count;
```

Example:

```
create table t1 (
    c1 int,
    c2 int
) partition by hash(c1 + 1) partitions 5;
```

Using Oceanbase Partition Table for Horizontal Splitting

Partitioned Table – KEY Partition

KEY partitioning is similar to HASH partitioning. It also determines which partition the data belongs to by taking the modulus of the number of partitions.

KEY partitioning rules:

- The system will perform an internal default HASH function on the KEY partition key and then take the modulus.
- Users usually have no way to know which partition a row belongs to through simple calculations.
- KEY partitions do not require INT type, they can be any type.
- KEY partitions cannot be written as expressions, only columns.
- KEY partitions support vectors.
- When the KEY partition key does not write any column, it means that the column of the KEY partition is the primary key

The syntax format of KEY partition is as follows:

```
CREATE TABLE table_name (column_name column_type[,column_name column_type])
  PARTITION BY { KEY([column_name_list]) }
    PARTITIONS partition_count;
```

Example:

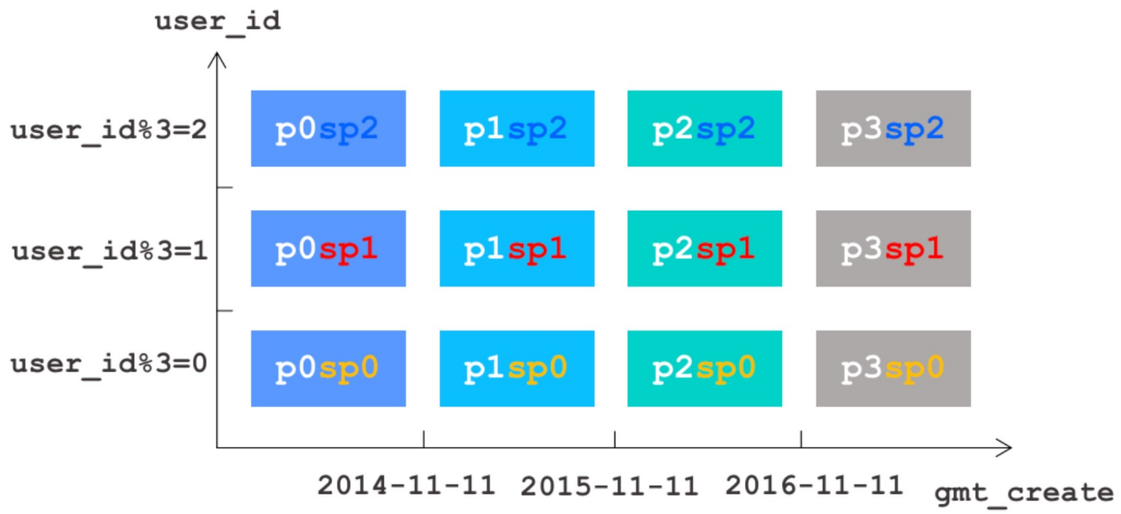
```
CREATE TABLE tbl1_k(id INT,gmt_create DATETIME,info VARCHAR(20))
       PARTITION BY KEY(id,gmt_create) PARTITIONS 10;
Query OK, 0 rows affected
```

Using Oceanbase Partition Table for Horizontal Splitting

Partition Tables - Composite Partitions (I)

Secondary partitioning means splitting data into partitions according to two dimensions. The most common place is in the field of user bills, where HASH partitioning is done according to user_id and RANGE partitioning is done according to bill creation time.

OceanBase MySQL mode currently supports six partition types: HASH, RANGE, LIST, KEY, RANGE COLUMNS, and LIST COLUMNS. Secondary partitioning is a combination of any two partition types.



Using Oceanbase Partition Table for Horizontal Splitting

Partition Tables – Composite Partitions (II)

Advantages of composite partitioning:

- Depending on the SQL statement, partition pruning on one or two dimensions may improve performance.
- Queries may use full partitions or partial partition joins on either dimension.
- You can perform parallel backups and restores on a single table.
- The number of partitions is greater than a single level of partitioning, which may benefit from parallel execution.
- You can implement a rolling window to support historical data, and you can still partition on another dimension if many statements can benefit from partition pruning or partition joins.
- You can store data differently based on the identity of the partition key. For example, you may decide to store data for a specific product type in a read-only compressed format and keep data for other product types uncompressed.

Although OceanBase supports RANGE + HASH and HASH + RANGE combinations in composite partitioning, the add/drop operation of RANGE partitioning must use RANGE partitioning as the first-level partitioning method. Therefore, for example, for flow tables with large amounts of data, it is recommended to use the RANGE + HASH combination for easy maintenance (adding and deleting partitions).

```
CREATE TABLE t_log_part_by_range_hash (
    log_id      int NOT NULL
    , log_value varchar(50)
    , log_date  TIMESTAMP NOT NULL
    , PRIMARY key(log_id, log_date)
) PARTITION BY RANGE(UNIX_TIMESTAMP(log_date))
SUBPARTITION BY HASH(log_id) SUBPARTITIONS 16
(
    PARTITION M202001 VALUES LESS THAN(UNIX_TIMESTAMP('2020/02/01'))
    , PARTITION M202002 VALUES LESS THAN(UNIX_TIMESTAMP('2020/03/01'))
    , PARTITION M202003 VALUES LESS THAN(UNIX_TIMESTAMP('2020/04/01'))
    , PARTITION M202004 VALUES LESS THAN(UNIX_TIMESTAMP('2020/05/01'))
    , PARTITION M202005 VALUES LESS THAN(UNIX_TIMESTAMP('2020/06/01'))
    , PARTITION M202006 VALUES LESS THAN(UNIX_TIMESTAMP('2020/07/01'))
    , PARTITION M202007 VALUES LESS THAN(UNIX_TIMESTAMP('2020/08/01'))
    , PARTITION M202008 VALUES LESS THAN(UNIX_TIMESTAMP('2020/09/01'))
    , PARTITION M202009 VALUES LESS THAN(UNIX_TIMESTAMP('2020/10/01'))
    , PARTITION M202010 VALUES LESS THAN(UNIX_TIMESTAMP('2020/11/01'))
    , PARTITION M202011 VALUES LESS THAN(UNIX_TIMESTAMP('2020/12/01'))
    , PARTITION M202012 VALUES LESS THAN(UNIX_TIMESTAMP('2021/01/01'))
);
```

Using Oceanbase Partition Table for Horizontal Splitting

Partition Table – Manage partition tables

After the partition table is created successfully, you can modify the partition rules, add partitions, delete partitions, and truncate partitions according to business needs. For specific syntax, refer to the Creating and Managing Partitions section.

Using Oceanbase Partition Table for Horizontal Splitting

Partitioned Tables – Indexes

The query performance of partitioned tables is related to the SQL conditions. When the SQL contains a split key, the OceanBase database will perform partition retrieval based on the conditions and only search for specific partitions; if there is no split key, all partitions must be scanned.

Partitioned tables can also be indexed to improve performance. Like partitioned tables, indexes on partitioned tables can be partitioned or non-partitioned.

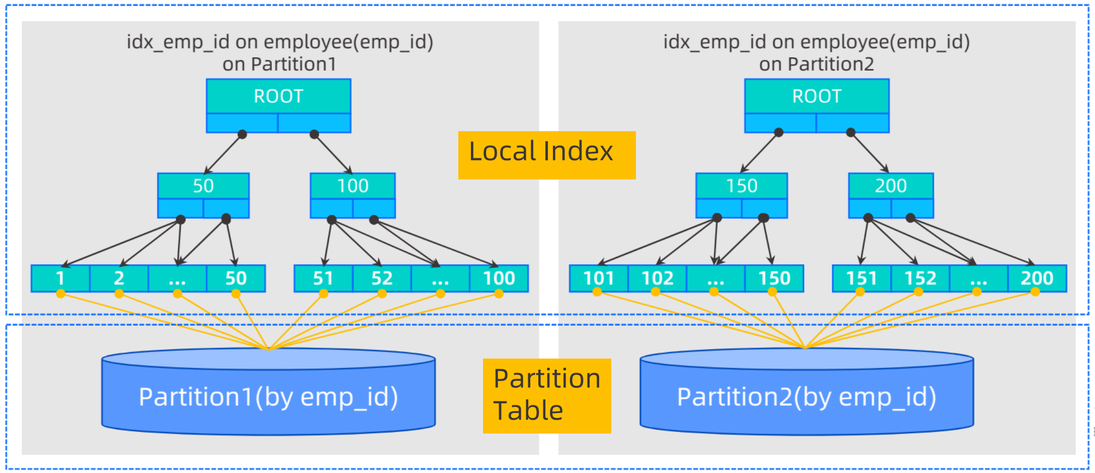
- If the index of a partitioned table is not partitioned, it is a global index (GLOBAL), which is an independent partition, and the index data covers the entire partition table.
- If the index partition of the partition table can be divided into two categories according to the partition strategy
  - One type is a partition strategy that is consistent with the partition table. The index data of each index partition covers the partition of the corresponding partition table. This index is also called a local index (LOCAL).
  - The other type is a global index (GLOBAL) whose partition strategy is inconsistent with the partition table. It is called a global partition index

Precautions:

- When creating an index for a partitioned table, a local index is created by default.
- It is recommended to use local indexes as much as possible and use global indexes only when necessary. Because global indexes will reduce the performance of DML, DML may generate distributed transactions.

Example:

```
CREATE INDEX idx_log_date ON
t_log_part_by_range_hash(log_date) LOCAL;
```



Using Oceanbase Partition Table for Horizontal Splitting

Partition Tables – Usage Recommendations

- Partitioning is done based on business forms (scattering of hot data, convenience of historical data maintenance, conditional forms of business SQL).
- Regarding the selection of partition keys in multi-dimensional business query scenarios, if both account numbers and card numbers exist, the selection of partition keys should be considered based on business usage frequency and business importance.
- Partition planning is done based on business scenarios when business query conditions are clear. The purpose of partitioning is to use the ability of partition pruning to improve query efficiency. It is forbidden to arbitrarily plan partition rules when the scenario is unclear. If only the first-level partition can be covered in some scenarios of query conditions, it is recommended to plan according to the first-level partition, and there is no need to forcibly plan for the second-level partition.
- For index creation of partitioned tables, select in the order of local index -> global partition index -> global index. Global index is used only when necessary, because global index will reduce the performance of DML and may generate distributed transactions.
- Try to bring partition keys when querying or modifying partitioned tables.
- It is not recommended to specify MAXVALUE for RANGE partitions, otherwise new partitions cannot be added later. For RANGE partitions, it is recommended that the business manage partitions by itself and add partitions regularly to avoid partition out-of-bounds problems. Consider using the ODC partition management function.
- To avoid write amplification problems, when selecting a custom primary key for a table, do not use randomly generated values. Try to keep them in order, such as increasing in time.
- For tables with historical data cleanup, partition table design needs to be based on business usage scenarios and cleanup cycles. For example, for transaction flow tables, partitions can be created by day and old partitions can be deleted by day.

Thank You!

OceanBase Official website:
https://oceanbase.github.io/

GitHub Discussions:
https://github.com/oceanbase/oceanbase/discussions