

# OceanBase 2.2 MySQL 租户开发者指南

最后更新日期： 2020-04-03



# 目录

|   |          |
|---|----------|
| <b>目录</b>                               | <b>I</b> |
| <b>引言</b>                               | <b>1</b> |
| 受众                                      | 1        |
| 文档获取途径                                  | 1        |
| 文档反馈途径                                  | 1        |
| 相关文档                                    | 2        |
| 约定                                      | 2        |
| <b>OCEANBASE 开发者指南版本变化</b>              | <b>2</b> |
| OCEANBASE 2.2 的功能变化 (2.2.3.0)           | 2        |
| <b>第 1 章 前言</b>                         | <b>3</b> |
| 1.1 关于 OCEANBASE 数据库开发者                 | 3        |
| 1.2 关于这个文档                              | 4        |
| 1.3 关于 OCEANBASE 数据库                    | 4        |
| 1.3.1 OceanBase 集群简介                    | 4        |
| 1.3.2 OceanBase 的租户简介                   | 5        |
| 1.3.3 OceanBase 的 MySQL 租户的数据库对象        | 6        |
| 1.3.4 OceanBase 数据库访问方式                 | 6        |
| 1.4 关于示例数据库 TPCC                        | 8        |
| <b>第 2 章 连接 OCEANBASE 数据库，开启探索之旅</b>    | <b>9</b> |
| 2.1 通过 OCP 新建 OCEANBASE 租户(实例)          | 9        |
| 2.1.1 新建 OceanBase 的 MySQL 租户(实例)       | 9        |
| 2.2 通过 MySQL 客户端连接 OCEANBASE 的 MySQL 租户 | 11       |
| 2.3 通过 DBeaver 连接 OCEANBASE 数据库         | 12       |
| 2.3.1 DBeaver 注册 OceanBase 数据库驱动        | 12       |
| 2.3.2 DBeaver 添加 OceanBase 数据源          | 13       |
| 2.4 创建 OCEANBASE 示例数据库 TPCC             | 16       |
| 2.5 通过 OBCLIENT 探索 OCEANBASE MySQL 租户   | 18       |
| 2.5.1 通过 obclient 查看 MySQL 租户的数据库对象     | 18       |
| 2.5.2 通过 obclient 查看表属性和数据              | 19       |
| 2.6 查询表数据                               | 20       |
| 2.6.1 关于查询语句                            | 20       |
| 2.6.2 在 DBeaver 中运行查询                   | 21       |
| 2.6.3 查询表里符合特定搜索条件的数据                   | 22       |
| 2.6.4 对查询的结果进行排序                        | 22       |

|  |           |
|--|-----------|
| 2.6.5 从多个表里查询数据.....                     | 23        |
| 2.6.6 在查询中使用各种操作符、函数.....                | 25        |
| 2.6.7 查看查询执行计划.....                      | 33        |
| 2.6.8 在查询中使用 SQL Hint.....               | 34        |
| 2.6.9 关于查询超时设计.....                      | 37        |
| <b>第 3 章 关于 DML 语句和事务.....</b>           | <b>37</b> |
| 3.1 关于数据操作语言 (DML) 语句.....               | 38        |
| 3.1.1 关于 INSERT 语句.....                  | 38        |
| 3.1.2 关于 UPDATE 语句.....                  | 41        |
| 3.1.3 关于 DELETE 语句.....                  | 43        |
| 3.1.4 关于 REPLACE INTO 语句 (MySQL 租户)..... | 43        |
| 3.2 关于事务控制语句.....                        | 44        |
| 3.3 提交事务.....                            | 44        |
| 3.4 回滚事务.....                            | 45        |
| 3.5 事务保存点.....                           | 46        |
| 3.6 关于事务超时.....                          | 48        |
| 3.6.1 关于事务空闲超时.....                      | 49        |
| 3.6.2 关于事务未提交超时.....                     | 50        |
| <b>第 4 章 创建和管理数据库对象.....</b>             | <b>52</b> |
| 4.1 关于数据定义语言 (DDL) 语句.....               | 52        |
| 4.2 创建和管理表.....                          | 53        |
| 4.2.1 关于 SQL 数据类型.....                   | 53        |
| 4.2.2 创建表.....                           | 54        |
| 4.2.3 关于列的自增.....                        | 58        |
| 4.2.4 关于列的约束类型.....                      | 59        |
| 4.2.5 关于表的索引.....                        | 61        |
| 4.2.6 闪回被删除的表.....                       | 63        |
| 4.3 创建和管理分区表.....                        | 64        |
| 4.3.1 分区路由.....                          | 64        |
| 4.3.2 分区策略.....                          | 64        |
| 4.3.3 分区表的索引.....                        | 70        |
| 4.3.4 分区表使用建议.....                       | 70        |
| 4.4 创建和管理表分组.....                        | 71        |
| 4.4.1 关于表分组.....                         | 71        |
| 4.4.2 创建表时指定表分组.....                     | 72        |
| 4.4.3 查看表分组信息.....                       | 73        |
| 4.4.4 向表分组中增加表.....                      | 73        |
| 4.4.5 删除表分组.....                         | 74        |
| 4.5 创建和管理视图.....                         | 74        |
| 4.5.1 创建视图.....                          | 74        |
| 4.5.2 修改视图的查询语句.....                     | 75        |
| 4.5.3 删除视图.....                          | 75        |

|                               |           |
|-------------------------------|-----------|
| <b>第 5 章 向 OCEANBASE 迁移数据</b> | <b>77</b> |
| 5.1 关于数据迁移和同步                 | 77        |
| 5.2 通用数据同步框架 DATAX            | 77        |
| 5.2.1 DataX 简介                | 77        |
| 5.2.2 DataX 的使用示例             | 77        |
| 5.3 不同数据源的 DATAX 读写插件示例       | 79        |
| 5.3.1 CSV 文件的读写插件             | 79        |
| 5.3.2 MySQL 数据库的读写插件          | 80        |
| 5.3.3 ORACLE 数据库的读写插件         | 81        |
| 5.3.4 DB2 数据库的读写插件            | 82        |
| 5.3.5 OceanBase 数据库的读写插件      | 82        |
| 5.4 OCEANBASE 数据加载技术          | 86        |
| <b>附录</b>                     | <b>88</b> |
| JAVA 连接 OCEANBASE 示例          | 88        |
| OCEANBASE 常用参数变量              | 89        |
| OCEANBASE 常用 SQL HINTS        | 91        |

## 引语

### 受众

本文档是为任何想了解 OceanBase 数据库应用程序开发的人编写的，主要是向对 OceanBase 数据库新手的开发人员介绍应用程序开发。本文档假设您对关系数据库的概念有一个大致的了解，并了解您将使用 OceanBase 数据库开发应用程序的操作系统环境。

由于 OceanBase 可以兼容 MySQL 或 ORACLE，本文档主要是针对 MySQL 租户的开发指引。

### 文档获取途径

本文档以及后期更新可以从以下资源处获取：

- OceanBase 官网：<https://oceanbase.alipay.com/docs>
- OceanBase 官方公众号：OceanBase
- OceanBase 钉钉交流群：21949783



扫一扫群二维码，立刻加入该群。

- OceanBase 服务团队邮箱：[support.oceanbase@service.alipay.com](mailto:support.oceanbase@service.alipay.com)
- OceanBase 团队成员

### 文档反馈途径

如果您在使用文档过程中遇到问题，欢迎发邮件反馈给作者。  
邮箱地址：[qing.meiq@alibaba-inc.com](mailto:qing.meiq@alibaba-inc.com)。

## 相关文档

- 官网已有文档: <https://oceanbase.alipay.com/docs>
- [OceanBase 2.2 ORACLE 租户开发者指南](#)

## 约定

本文档使用的文本格式约定如下

| 约定        | 含义                                      |
|-----------|---|
| <b>粗体</b> | 粗体字表示与操作相关联的图形用户界面元素，或文本或术语表中定义的术语。     |
| <i>斜体</i> | 斜体类型表示您提供特定值的标题、强调或占位符变量。               |
| 等宽字体      | 等宽字体表示段落中的命令、URL、示例中的代码、屏幕上出现的文本或输入的文本。 |

## OceanBase 开发者指南版本变化

本节包含:

- [OceanBase 2.2 的功能变化 \(2.2.3.0\)](#)

### OceanBase 2.2 的功能变化 (2.2.3.0)

新增功能:

#### 1. 存储过程/客户端协议

- 兼容 Oracle 的 PL/SQL 第一版正式推出，提供基本的 PL/SQL 使用能力。后续版本继续不断增强对 PL/SQL 的支持。
- 部分 Oracle 兼容的 PL/SQL 系统包，方便 Oracle 用户使用习惯。
- 支持二进制 prepare statement 协议,提升应用调用数据库执行性能。

#### 2. SQL 功能增强

- 支持除 binary float/double 外的全部 Oracle 基础数据类型，以及 BLOB/CLOB（最大长度受限）。

- 支持 MINUS, ROLLUP, GROUPING 等 SQL 语法以及大量窗口函数支持，更好的支持用户复杂分析查询的需要。
- 支持 Oracle 兼容的 NLS 系列长度和时间日期类型设置，提供丰富的时间日期格式和转换支持。
- 支持 150+Oracle 兼容函数和表达式，30 张字典视图和 30 张性能视图。

### 3. SQL 执行计划管理

- 兼容 Oracle 管理接口的 SQL Plan Management 功能，支持执行计划的固化管理和自动演进，确保系统运行和升级后的稳定性，降低运维复杂度。

### 4. 事务能力增强

- 闪回查询增强(Flashback Query)，提供历史数据查询功能
- 支持串行化隔离级别（关系数据库的最强事务隔离级别），对有强事务隔离性需求的应用场景提供原生支持

#### 增强点：

#### 1. 稳定性提升

集群稳定性大幅提升，分布式事务运行稳定，最大单表行数达万亿级，单集群最大数据量超过 3 PB。

#### 2. 扩展性提升

Oracle 模式单表支持最大 65536 分区

#### 3. 兼容性增强

兼容 MySQL 5.6、Oracle 11.2（持续开发支持中）

#### 4. 性能增强

OLTP 性能相比 2.0 版本提升 50% 以上，部分复杂场景提升 100%；OLAP 场景查询优化和执行能力显著提升，TPC-H 全部 22 个查询，SF=1000(1TB)的数据量下，6 台 ECS(56 超线程) Server 总执行时间为 730s

## 第 1 章 前言

### 1.1 关于 OceanBase 数据库开发者

OceanBase 数据库开发人员负责创建或维护使用 OceanBase 相关产品的应用程序的数据库组件。OceanBase 数据库开发人员要么基于 OceanBase 开发新应用程序，要么将现有应用程序转换到 OceanBase 数据库环境中运行。



## 1.2 关于这个文档

此文档是为应用程序开发人员准备的 OceanBase 数据库文档的入门文档，它包含了：

- 解释了 OceanBase 数据库开发背后的基本概念。
- 展示并示例如何使用 OceanBase 的 MySQL 租户。

## 1.3 关于 OceanBase 数据库

### 1.3.1 OceanBase 集群简介

OceanBase 集群通常由运维人员管理，OceanBase 数据库集群将多个机器资源聚合成一个大的资源池之后再二次分配给不同租户(也称为实例)。OceanBase 不同租户之间彼此资源隔离，数据访问也是完全隔离的。

由于 OceanBase 有些特征会影响到租户的使用，所以这里需要对集群的一些概念功能特点进行一个简介。

- **OceanBase 集群的多副本架构**

OceanBase 集群通常是三副本架构，少数场景可能会使用五副本。在三副本架构下，OceanBase 集群的节点数通常是三的倍数，集群节点会分为三个区域 (Zone)。每个区域的节点数通常保持相等，可以有 1 或多个节点。每个租户的数据也会分布在这三个区域里，但不一定用尽每个 Zone 的所有节点，这取决于租户资源池属性的设置，由运维人员确定。

- **OceanBase 集群的高可用能力**

在三副本架构的 OceanBase 集群里，默认每份数据也有三份，分布在三个区域中。这个数据表示的最小粒度叫分区。分区是表的子集，有关表的详细介绍请参见后面章节“[创建表](#)”。OceanBase 集群的高可用的粒度就是分区。故障切换时的切换粒度也是分区。所以 OceanBase 节点故障时，应用可能是部分数据访问中断，并且会自动恢复访问。当故障节点内部分区非常多的时候，不同数据的恢复时间可能有细微的区别。

- **OceanBase 的参数设置**

OceanBase 支持通过参数 (parameters) 来影响集群和租户的功能、性能等。集群参数通常是在 sys 租户里设置，影响范围是整个集群，也包括集群里的租户。同时 OceanBase 针对租户也支持用变量 (variables) 来设置，影响范围是当前租户。

有关 OceanBase 常用的参数和变量请参考附录“[OceanBase 常用参数变量](#)”。

### ■ OceanBase 的写特点

当第一次向 OceanBase 的表中修改一笔数据时，OceanBase 会将该记录所在的块读入到内存中的一块只读区域中，然后在另外一块内存区域记录一笔修改记录。前称为基线数据( sstable )，后者这笔修改叫增量( memtable )，对应的内存称为增量内存。无论是插入、更新还是删除，OceanBase 都不会针对 sstable 进行修改，而是在原来的增量基础上追加( append )新的增量。这设计使得 OceanBase 的写产生的 IO 非常少，性能很好。增量会一直在内存中不落盘，直到增量内存使用率超过一定阈值后触发冻结事件( minor freeze )，此时会生成新的 memtable 供后续写入。老的增量 memtable 会直接转储到磁盘上，或者直接跟磁盘里基线数据( sstable )直接合并。

默认设置下，OceanBase 的增量和基线数据合并操作会在凌晨 2 点进行。由于 OceanBase 的内存写这个特点，开发和运维都需要关注自己的业务对数据库内存的消耗速度。如果增量内存写入速度远快于增量转储或合并速度，增量内存有可能会消耗完导致后续写入报错。此时需要在应用或数据库端做内存写入速度限流操作，或者对数据库实例内存进行扩容。

有关 OceanBase 集群的更多详细介绍请参考官网文档。

## 1.3.2 OceanBase 的租户简介

租户目前有两种兼容模式，兼容 ORACLE 或 MySQL(二选一)，不同兼容模式的实例再按照相应的方式把相关的业务数据组织在一起。在 ORACLE 兼容模式的租户下，这个组织方式叫模式( schema )；在 MySQL 兼容模式的租户下，这个组织方式叫数据库( database )。当您通过用户名、密码和 IP 连接 OceanBase 数据库时，您指定了相应的 OceanBase 集群名、租户名以及租户下的模式( schema )或者用户( user)，以表明您是这些数据的所有者。在 ORACLE 租户下，用户名和用户访问的模式名默认是相同的；在 MySQL 租户下，用户名和用户访问的数据库名默认可以不同。

OceanBase 集群默认会有一个租户(兼容 MySQL)，租户名是 **sys**，里面存储的是集群元数据信息。sys 租户主要是集群内部管理需要，运维人员经常访问使用。sys 租户下不建议建表存储数据。OceanBase 集群除去 sys 租户占用的资源外都是未分配资源，留给业务租户使用。业务使用 OceanBase 数据库需要先创建一个业务租户。

创建租户的工作通常是由数据库运维人员完成，通过产品 OCP 里“新建实例”完成。或者运维人员在 sys 租户下直接创建租户。

注意：租户跟实例是同义词。

从下面开始，本文档重点介绍 MySQL 租户的功能。有关 ORACLE 租户的功能，请参考文档《OceanBase 2.2 ORACLE 租户开发者指南》。

### 1.3.3 OceanBase 的 MySQL 租户的数据库对象

在 MySQL 租户下，每个数据库对象只属于一个数据库( database )，数据库名跟用户名没有直接关系，每个用户可以访问的数据库跟用户的权限有关系。用户访问数据库对象通常还需要切换到该数据库下；否则对象前面就要带上数据库对象所属的数据库名。

数据库下支持的数据库对象有：

- 表( Tables )  
表是 OceanBase 数据库最基本的存储单元，容纳用户的数据。每个表包含很多行( rows )，每行表示独立的数据记录( records )。每一行包含很多列( columns )，每列表示记录的一个字段( fields )。
- 索引( Indexes 或 Keys)
- 视图( Views )
- 存储的子程序  
存储的子程序是存储在数据库中的过程和函数。它们可以从访问数据库的客户端应用程序中调用。

**注意：** OceanBase 不建议客户过重依赖 MySQL 租户的存储过程和自定义函数，这一块也不是后期功能发展方向。

### 1.3.4 OceanBase 数据库访问方式

如果您要访问 OceanBase 的 MySQL 租户数据库，只能通过客户端程序，如 obclient 或 DBeaver。

客户端程序跟数据库的接口是结构化查询语言( SQL )。

#### 1.3.4.1 MySQL 客户端 ( mysql )

mysql 命令是 MySQL 数据库命令行下客户端，需要单独安装。OceanBase 的租户兼容类型支持 MySQL 和 OceanBase。当租户兼容 MySQL 时，可以通过 mysql 命令连接。

**注意：** OceanBase 当前版本支持的 MySQL 客户端版本只有：5.5、5.6、5.7 。

mysql 运行时需要指定 OceanBase MySQL 租户的连接信息，具体方法参考 [通过 MySQL 客户端 连接 OceanBase 数据库](#)。

连接上 OceanBase 数据库后，在 mysql 命令行环境里，可以运行一些 mysql 运维命令、SQL 语句来执行下面这些任务：

- 计算、存储和打印查询结果
- 创建数据库对象、检查和修改对象定义
- 执行数据库管理，修改参数等

### 1.3.4.2 关于 OceanBase 客户端 (obclient)

**obclient** 是一个交互式和批处理查询工具，需要单独安装。它是一个命令行用户界面，在连接到数据库时充当客户端。它支持 OceanBase 的 ORACLE 租户和 MySQL 租户。

obclient 运行时需要指定 OceanBase 租户的连接信息，具体方法参考 [通过 obclient 连接 OceanBase 数据库](#)。

连接上 OceanBase 数据库后，在 obclient 里，可以运行一些 obclient 命令(包含常用的 MySQL 命令)、SQL 语句、PL/SQL 语句来执行下面这些任务：

- 计算、存储和打印查询结果
- 创建数据库对象、检查和修改对象定义
- 开发和运行批处理脚本
- 执行数据库管理，修改参数等

### 1.3.4.3 关于数据库图形客户端 Dbeaver

DBeave 是一款开源产品，为开发人员、数据库管理员、分析人员和所有需要使用数据库的人员提供免费的多平台数据库工具。支持所有流行数据库：My SQL、PostgreSQL、SQLite、Oracle、DB2、SQLServer、Sybase、MSAccess、Teradata、Fire bird、Apache Hive、Phoenix、Presto 等。更多 Dbeaver 信息请查看 <https://dbeaver.io/>。

通过引用 OceanBase 的 Java 数据库连接驱动，Dbeaver 也可以支持 OceanBase 的 MySQL 和 ORACLE 租户的连接。

在 Dbeaver 客户端里，您可以方便的在 MySQL 租户下查看、定义和修改表、视图、索引等 OceanBase 常用数据库对象。

### 1.3.4.4 关于结构化查询语言 (SQL)

结构化查询语言(SQL)是一种基于集合和关系代数理论的高级计算机语言，所有程序和用户都可以使用 SQL 访问 OceanBase 数据库中的数据。

SQL 是一种声明性的或非程序性的语言，也就是说，它描述了该做什么，而不是如何做。您指定所需的结果集（例如，当前员工的名称），但不指定如何获取它。不过通过一些 SQL 写法或 SQL Hint 可以对 SQL 获取数据的执行计划施加一定的影响，从而提升 SQL 的执行性能。

### 1.3.4.5 Java 数据库连接驱动 (JDBC)

Java 数据库连接 (JDBC) 是一种 API，它使 Java 能够将 SQL 语句发送到对象关系数据库，如 ORACLE、MySQL 数据库。JDBC 支持为 Java 暴露 SQL 数据类型，并快速访问 SQL 数据。

OceanBase 的 MySQL 租户兼容 MySQL 的连接协议，使用标准的 JDBC 可以连接 OceanBase 的 MySQL 租户。但 JDBC 默认不支持 ORACLE 租户的连接协议。

### 1.3.4.6 OceanBase Java 数据库连接驱动

OceanBase 实现了自己的 JDBC 驱动，使 Java 能够将 SQL 语句发送到 OceanBase 的 MySQL 租户和 ORACLE 租户。OceanBase JDBC 支持为 Java 暴露 SQL 数据类型、PL/SQL 对象，并快速访问 SQL 数据。

OceanBase JDBC 驱动文件名为: oceanbase-client-[版本号].jar，可以从官网下载。

OceanBase 数据库驱动文件 1.0 相关版本的类名为:  
`com.alipay.oceanbase.obproxy.mysql.jdbc.Driver`。

OceanBase 数据库驱动文件从 1.1.0 后类名更改为:  
`com.alipay.oceanbase.jdbc.Driver`，原类名会保留，但是不推荐使用。

## 1.4 关于示例数据库 TPCC

示例数据库安装脚本可以安装在 OceanBase 的 ORACLE 租户或者 MySQL 租户上。如果是 ORACLE 租户，需要先创建一个用户和模式 (schema)，然后登录这个用户执行脚本；如果是 MySQL 租户，需要先创建一个数据库 (database)，然后进入这个数据库下执行脚本。

脚本分为两部分。一是数据库建表语句，文件名 “create\_tables\_mysql.sql”，创建业务表和视图。二是数据初始化语句，文件名 “init\_data.sql”，初始化相关表数据。注意，这里只是部分测试数据。

示例数据库的业务是由国际事务性能委员会 (TPC) 制定的 TPC-C 标准，定义了商品销售的订单创建和订单支付等的基准测试标准，是数据库联机交易处理系统 (OLTP) 的权威基准测试标准。

示例数据库可以从 <https://github.com/obpilot/ob-samples> 免费下载，具体使用方法请查看 README.md。

## 第 2 章 连接 OceanBase 数据库，开启探索之旅

您可以使用客户端工具连接 OceanBase 数据库也就是租户，如 obclient 或 DBeaver。当连接到数据库后，您可以查看或修改数据库对象、属性参数以及表的数据等，您可以使用 SQL 查询多个表或视图的数据，以及修改这些数据。

当您通过客户端程序连接到数据库后，您可以在客户端程序里输入和运行很多命令。具体细节信息，请查看客户端程序文档。

### 2.1 通过 OCP 新建 OceanBase 租户(实例)

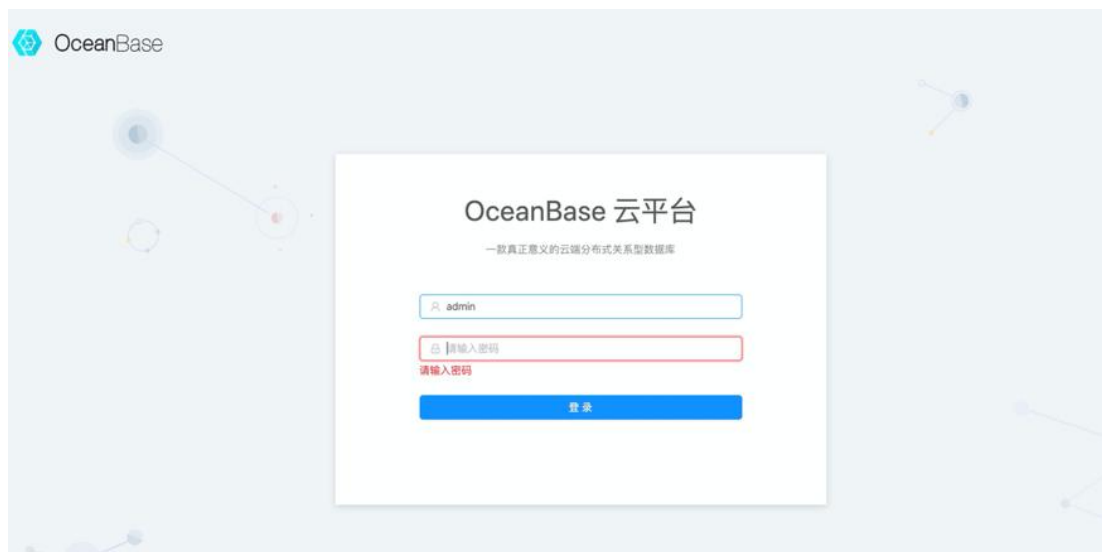
业务使用 OceanBase 数据库之前，需要先从 OceanBase 集群里创建一个租户(即实例)。有关租户介绍请参考“[OceanBase 的租户](#)”。

创建租户通常由数据库运维人员完成，可以在 obclient 下创建，也可以通过 OceanBase 运维平台(OCP)创建。创建 OceanBase 租户时需要选择兼容的类型，在 MySQL 和 ORACLE 之间二选一。

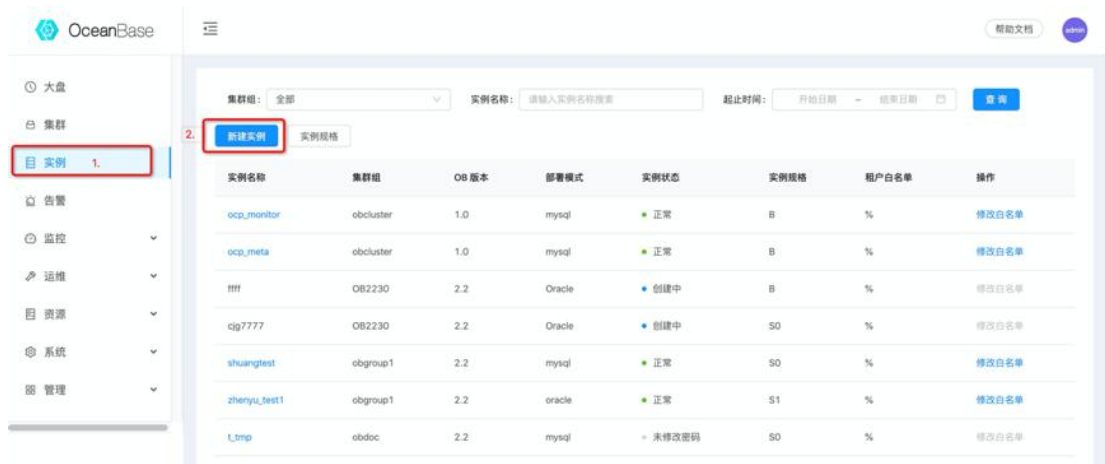
#### 2.1.1 新建 OceanBase 的 MySQL 租户(实例)

示例：通过 OCP 新建 MySQL 租户(实例)

##### 1. 登录 OCP 平台



##### 2. 点击左侧边栏“实例”，然后点击右边“新建实例”。



### 3. 输入实例相关必填信息



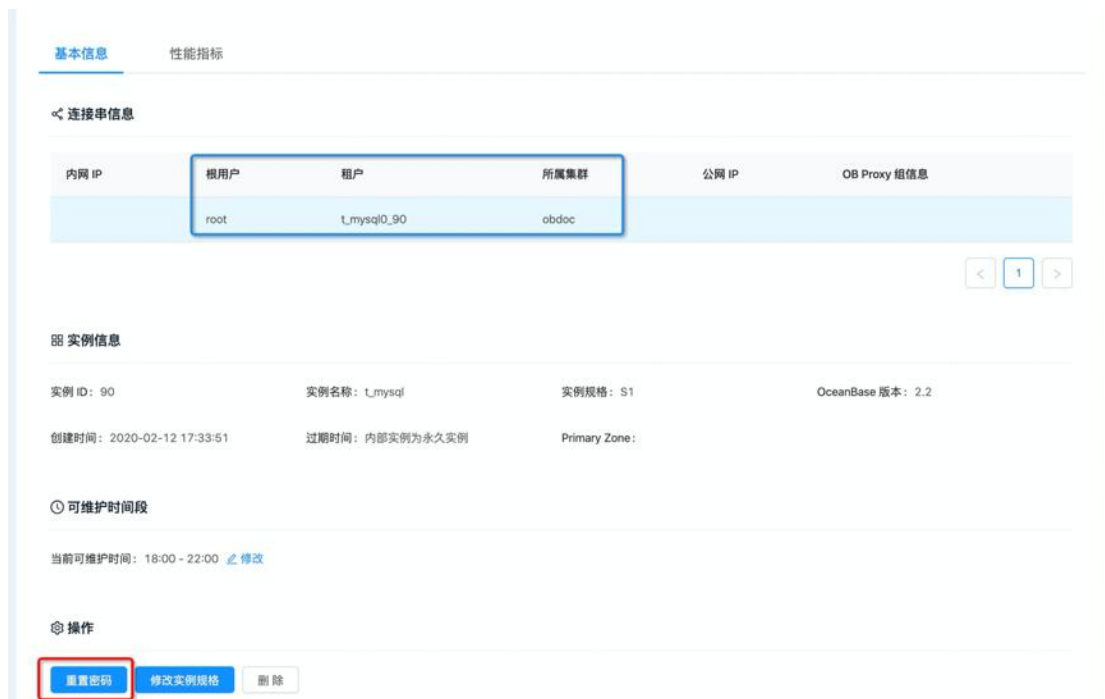
4. 等待 5 分钟。OCP 后台任务在创建实例。

5. 查找刚新建的实例。



6. 点击实例名称，进入实例详情，重置密码。





从这个页面里可以得到 MySQL 租户的管理员帐号信息是：

root@obmysql#obdemo 或者 obdoc:t\_mysql0\_90:root。实例新建好后，必须重置帐号的密码后才可以使用。

## 2.2 通过 MySQL 客户端连接 OceanBase 的 MySQL 租户

OceanBase 租户兼容 MySQL 的时候，您可以使用 MySQL 客户端连接该租户。MySQL 客户端版本目前仅支持 5.5、5.6、5.7。

使用 MySQL 客户端连接 OceanBase MySQL 租户步骤：

1. 打开一个命令行终端，确保环境变量 **PATH** 包含了 **MySQL** 客户端命令所在目录。
2. 参照下面格式提供 **mysql** 的运行参数

```
$mysql -h192.168.1.101 -uroot@obmysql#obdemo -P2883 -pabcABC123 -c -A oceanbase
```

说明：

- -h 后提供 OceanBase 数据库连接 IP，通常是一个 OBProxy 地址。
- -u 后提供租户的连接账户，格式有两种：用户名@租户名#集群名 或者 集群名:租户名:用户名。MySQL 租户的管理员用户名默认是 **root**。
- -P 后提供 OceanBase 数据库连接端口，也是 OBProxy 的监听端口，默认是 **2883**，可以自定义。
- -p 后提供账户密码，为了安全可以不提供，改为在后面提示符下输入，密码文本不可见。



- -c 表示在 `mysql` 运行环境中不要忽略注释。
- -A 表示在 `mysql` 连接数据库时不自动获取统计信息。
- `oceanbase` 是访问的数据库名，可以改为业务数据库

连接成功后，默认会有命令行提示符

```
MySQL [oceanbase]>
```

然后就可以发一些 MySQL 运维命令或者 SQL 语句等。

要退出 OceanBase 命令行，输入 `exit` 然后 回车；或者按快捷键 `ctrl+d`。

### 示例：通过 MySQL 客户端连接 OceanBase 的 MySQL 租户

```
$mysql -h192.168.1.101 -uroot@obmysql#obdemo -P2883 -pabcABC123 -c -A oceanbase
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 62488
Server version: 5.6.25 OceanBase 2.2.20 (...) (Built Aug 10 2019 15:27:33)

<...省略...>

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

MySQL [oceanbase]> show databases;
+-----+
| Database |
+-----+
| oceanbase |
| information_schema |
| mysql |
| test |
+-----+
4 rows in set (0.00 sec)

MySQL [oceanbase]> exit
Bye
```

## 2.3 通过 DBeaver 连接 OceanBase 数据库

DBeaver 是一款开源的通用的数据库图形客户端工具。您可以从它的官网获取：

<https://dbeaver.io/>

本节假设您已经安装好 DBeaver，然后演示如何使用 DBeaver 注册 OceanBase 数据库类型和连接到 OceanBase 数据库。

### 2.3.1 DBeaver 注册 OceanBase 数据库驱动

注册 OceanBase 数据库驱动需要有 OceanBase Java 驱动文件，可以从 OceanBase 官网下载文件里获取。

1. 驱动名称，自定义，便于识别。

2. 输入该驱动的类型名:

`com.alipay.oceanbase.obproxy.mysql.jdbc.Driver`

3. 输入 URL 模板: `jdbc:oceanbase://<server>:<port>/<database>`。

4. 点击“添加文件”以添加驱动对应的 jar 文件。

5. 点击“找到类”，选择对应的类。

6. 点击“确认”。



**注意:**

OceanBase 数据库驱动文件从 1.1.0 后类名更改为:

`com.alipay.oceanbase.jdbc.Driver`，原类名会保留，但是不推荐使用。

## 2.3.2 DBeaver 添加 OceanBase 数据源

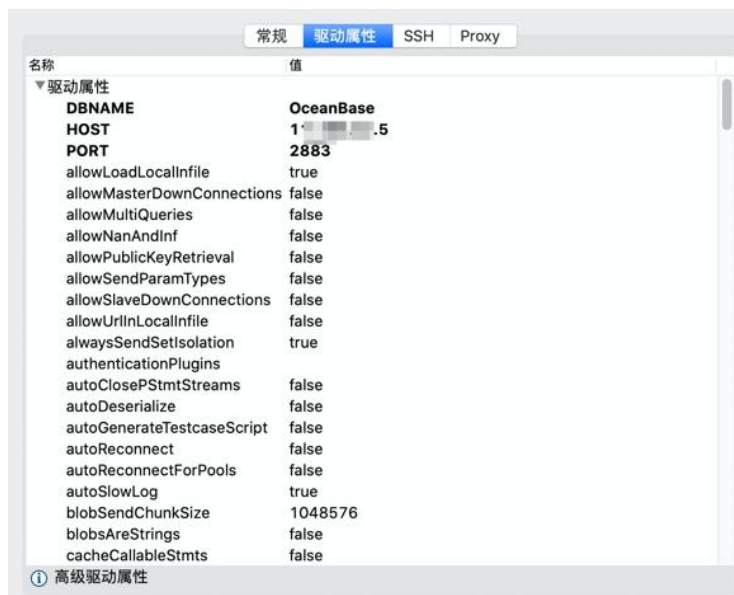
在 DBeaver 里新增一个 OceanBase 数据源的步骤如下:

1. 点击“新建连接”，选择类型“OceanBase”。

2. 输入用户名和密码。这个用户名对应 OceanBase 租户的账户名，格式是：[用户名]@[租户名]#[集群名]。



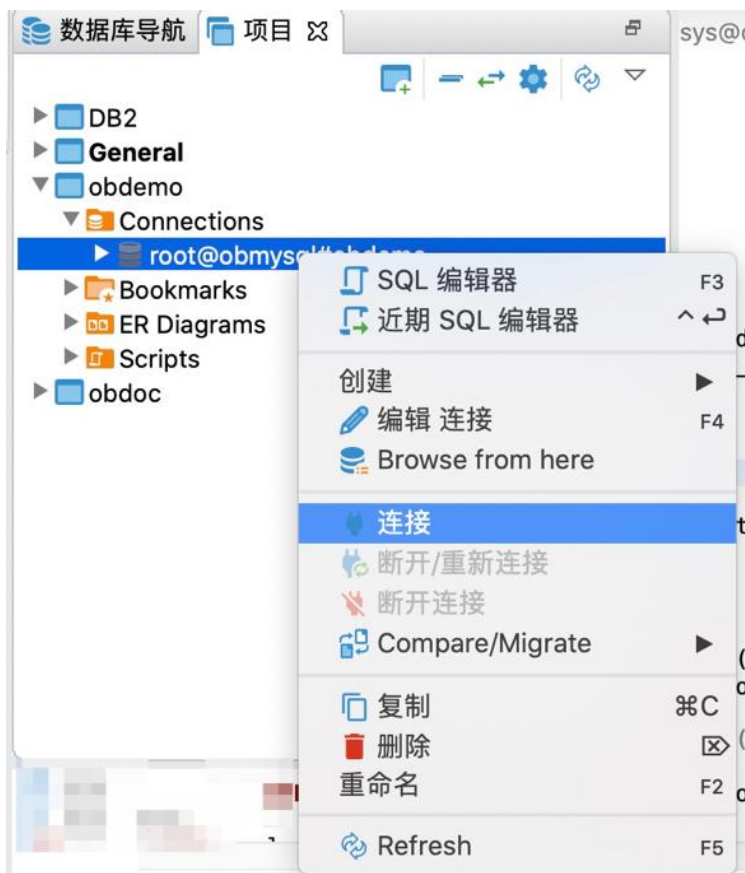
3. 输入连接 IP 和端口，以及默认模式(或数据库)



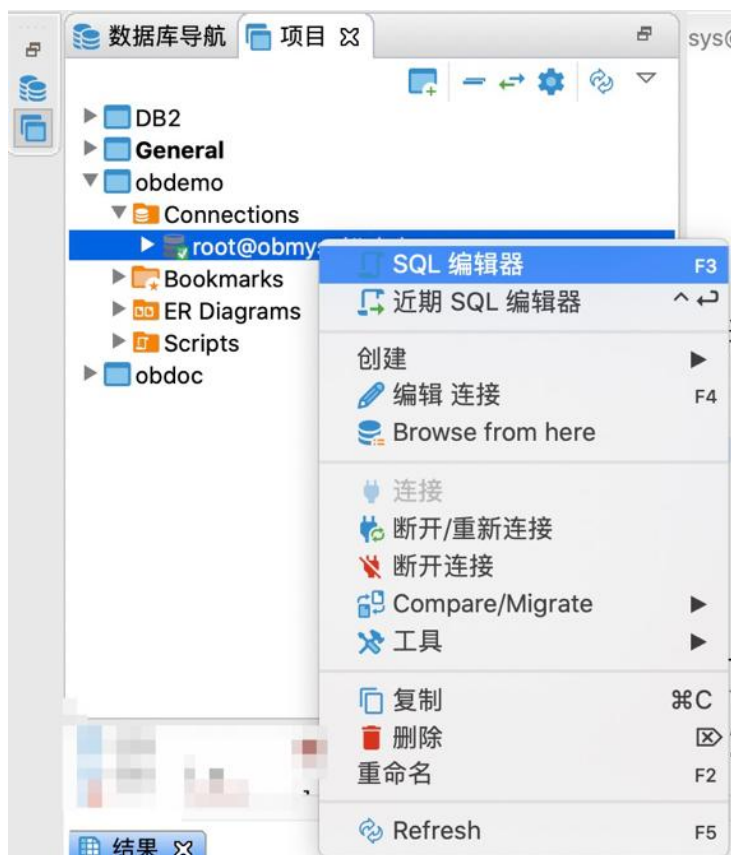
其中 DBNAME 对应的是 MySQL 租户的数据库( database )。

4. 点击“测试连接”，提示“已连接”。

5. 在 DBeaver 的左侧边栏“数据库”或“项目”里找到新建的连接，重命名后，右键点击，选择“连接”



6. 再在侧边栏的数据源上点击右键，选择“SQL 编辑器”，开启一个 SQL 查询窗口。



## 2.4 创建 OceanBase 示例数据库 TPCC

默认情况 OceanBase 没有创建示例数据库 TPCC，需要手动创建。示例数据库必须在业务租户(实例)下创建。有关 OceanBase 实例创建方法请参考“[通过 OCP 新建 OceanBase 租户\(实例\)](#)”。有关示例数据库介绍请参考“[关于示例数据库 TPCC](#)”。

租户(实例)创建好后，如果是 ORACLE 租户，需要创建相应的模式( schema )来存放示例数据库的对象。如果是 MySQL 租户，需要创建相应的数据库( database )来存放实例数据库对象。然后还要分配相应的用户和访问权限。

### 示例：在 MySQL 租户下创建示例数据库 TPCC

#### 1. 通过 obclient 连接 MySQL 租户。

具体方法请参考“[通过 obclient 连接 OceanBase 的租户](#)”。

```
#obclient -h192.168.1.101 -uroot@obmysql#obdemo -P2883 -pabcABC123 -A oceanbase
obclient: [Warning] Using a password on the command line interface can be insecure.
Welcome to the OceanBase monitor. Commands end with ; or \g.
Your OceanBase connection id is 57981
Server version: 5.6.25 OceanBase 2.2.20 (...) (Built Aug 10 2019 15:27:33)

<...省略...>

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

## 2. 创建一个数据库

```
obclient> create database tpccdb;
Query OK, 1 row affected (0.02 sec)
obclient> show databases;
+-----+
| Database |
+-----+
| oceanbase |
| information_schema |
| mysql |
| test |
| tpccdb |
+-----+
5 rows in set (0.01 sec)
```

## 3. 创建一个用户并赋权

```
obclient> grant all privileges on tpccdb.* to tpcc identified by '123456';
Query OK, 0 rows affected (0.02 sec)

obclient> show grants for tpcc;
+-----+
| Grants for tpcc@% |
+-----+
| GRANT USAGE ON *.* TO 'tpcc' |
| GRANT SELECT ON `oceanbase`.* TO 'tpcc' |
| GRANT ALL PRIVILEGES ON `tpccdb`.* TO 'tpcc' |
+-----+
3 rows in set (0.01 sec)

obclient>
```

## 4. 创建数据库对象

建表脚本 `create_tables_mysql.sql` 中有建表语句。使用之前请查看相关的说明文件( `readme` )。

```
obclient> use tpccdb;
Database changed
obclient> source create_mysql_tables.sql
Query OK, 0 rows affected (0.01 sec)
<...省略...>
Query OK, 0 rows affected (0.01 sec)
+-----+
| Tables_in_tpccdb |
+-----+
| cust |
| dist |
| hist |
| item |
| load_hist |
| load_proc |
| nord |
| ordl |
| ordr |
| stock_item |
| stok |
| ware |
+-----+
12 rows in set (0.00 sec)
```

## 5. 初始化表数据

```
obclient> source init_data.sql
Query OK, 0 rows affected (0.01 sec)
<...省略...>
Query OK, 0 rows affected (0.01 sec)
+-----+-----+
| table_name | rows_cnt |
+-----+-----+
| WARE       |        2 |
| DIST       |       20 |
| NORD       |       40 |
| ORDR       |       60 |
| HIST       |      240 |
| ITEM       |     622 |
| ORDL       |     626 |
| CUST       |    1040 |
| STOK       |    1244 |
+-----+-----+
9 rows in set (0.01 sec)
```

## 2.5 通过 obclient 探索 OceanBase MySQL 租户

MySQL 租户连接方法请参考“[通过 obclient 连接 OceanBase 的租户](#)”。

### 2.5.1 通过 obclient 查看 MySQL 租户的数据库对象

**示例：**通过 obclient 查看 MySQL 租户的数据库对象

```
#obclient -h192.168.1.101 -utpcc@obmysql#obdemo -P2883 -p123456 -A tpccdb

obclient> show tables;
+-----+
| Tables_in_tpccdb |
+-----+
| cust              |
| dist              |
| hist              |
| item              |
| load_hist         |
| load_proc         |
| nord              |
| ordl              |
| ord               |
| stock_item        |
| stok              |
| ware              |
+-----+
12 rows in set (0.01 sec)
```

## 2.5.2 通过 obclient 查看表属性和数据

### 示例：通过 obclient 查看 MySQL 租户的表的属性

```
obclient> desc ordl;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ol_w_id        | int(11)       | NO   | PRI | NULL    |      |
| ol_d_id        | int(11)       | NO   | PRI | NULL    |      |
| ol_o_id        | int(11)       | NO   | PRI | NULL    |      |
| ol_number      | int(11)       | NO   | PRI | NULL    |      |
| ol_delivery_d  | date          | YES  |     | NULL    |      |
| ol_amount      | decimal(6,2)  | YES  |     | NULL    |      |
| ol_i_id        | int(11)       | YES  |     | NULL    |      |
| ol_supply_w_id | int(11)       | YES  |     | NULL    |      |
| ol_quantity    | int(11)       | YES  |     | NULL    |      |
| ol_dist_info   | char(24)      | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.01 sec)

obclient> show create table ordl\G
***** 1. row *****
      Table: `ordl`
Create Table: CREATE TABLE `ordl` (
  `ol_w_id` int(11) NOT NULL,
  `ol_d_id` int(11) NOT NULL,
  `ol_o_id` int(11) NOT NULL,
  `ol_number` int(11) NOT NULL,
  `ol_delivery_d` date DEFAULT NULL,
  `ol_amount` decimal(6,2) DEFAULT NULL,
  `ol_i_id` int(11) DEFAULT NULL,
  `ol_supply_w_id` int(11) DEFAULT NULL,
  `ol_quantity` int(11) DEFAULT NULL,
  `ol_dist_info` char(24) DEFAULT NULL,
  PRIMARY KEY (`ol_w_id`, `ol_d_id`, `ol_o_id`, `ol_number`)
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = DYNAMIC COMPRESSION = 'zstd_1.0' REPLICA_NUM = 1 BLOCK_SIZE = 16384
USE_BLOOM_FILTER = FALSE TABLET_SIZE = 134217728 PCTFREE = 10 TABLEGROUP = 'tpcc_group'
  partition by hash(ol_w_id) partitions 6

1 row in set (0.02 sec)
```

### 示例：通过 obclient 查看 MySQL 租户的表的数据

```
obclient> select * from ware;
+-----+-----+-----+-----+-----+-----+-----+-----+
| w_id | w_ytd | w_tax | w_name      | w_street_1 | w_street_2 | w_city | w_state |
| w_zip |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 1200.00 | 0.1868 | n1P4zYo8OH | jTNkXKWXOdh | lf9QXTXXGoF04IZBkCP7 | srRq15uvxe5 | GQ |
506811111 |
| 2 | 1200.00 | 0.0862 | L6xwRsbDk | xEdT1jkENtLwo11Zb0 | NT0j4RCQ40qrS | vlwzndw2FPrO | XR |
063311111 |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

也可以使用参数 ' \G ' 结尾按列展示每行数据

```
obclient> select * from ware\G
```



```

***** 1. row *****
  w_id: 1
  w_ytd: 1200.00
  w_tax: 0.1868
  w_name: n1P4zYo80H
w_street_1: jTNkXKWxOdh
w_street_2: lf9QXTXXGoF04IZBkCP7
  w_city: srRq15uvxe5
  w_state: GQ
  w_zip: 506811111
***** 2. row *****
  w_id: 2
  w_ytd: 1200.00
  w_tax: 0.0862
  w_name: L6xwRsbdK
w_street_1: xEdTljKENTbLwo1IZb0
w_street_2: NT0j4RCQ40qrS
  w_city: v1wzndw2FPrO
  w_state: XR
  w_zip: 063311111
2 rows in set (0.01 sec)
obclient>

```

## 2.6 查询表数据

### 2.6.1 关于查询语句

一个查询，指一个 SELECT SQL 语句，从一个或多个表或者视图里查询数据。

最简单的 SQL 语句格式是：

```
SELECT select_list FROM table_list
```

- *select\_list* 指定的可以是后面 *table\_list* 里的列，也可以是函数值、字符常量、计算变量等。
- *table\_list* 指定的是包含所查数据的表或者视图。

上面是简单的查询语句格式，实际 *table\_list* 还可以是一个子查询语句。同时还可以带上 *where* 条件以限定返回的结果要符合某种条件。

需要注意的是，在 MySQL 租户里，*FROM table\_list* 可以没有，这样 *select\_list* 也不是具体的列，而是常量或者变量值。如下面这个 SQL：

```

$obclient -h192.168.1.101 -utpcc@obmysql#obdemo -P2883 -p123456 -A tpccdb
obclient> select '中', 6*6 ;
+-----+-----+
| 中 | 6*6 |
+-----+-----+
| 中 | 36 |
+-----+-----+
1 row in set (0.00 sec)

```

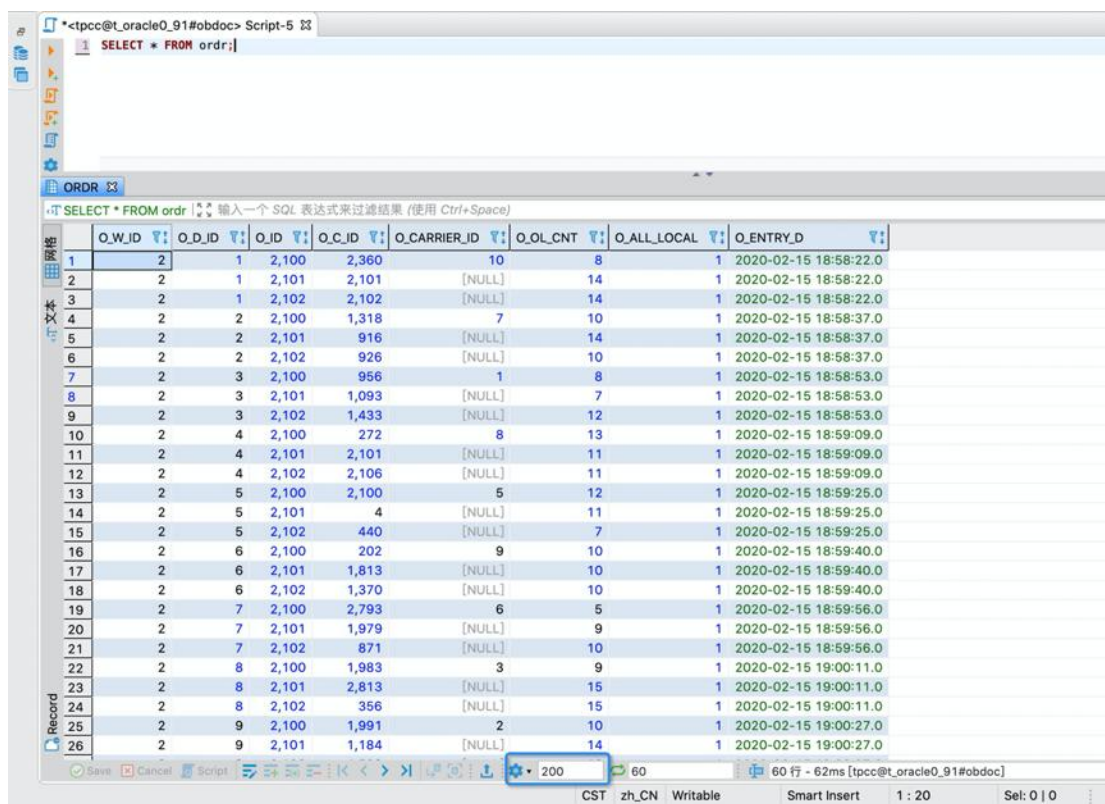
查询语句可以很复杂，后面介绍的查询用法都是很简单的用法。详细的用法请参考[《OceanBase SQL 参考（MySQL 租户）》](#)。

## 2.6.2 在 DBeaver 中运行查询

本节展示如何在 DBeaver 里通过 SQL 编辑器执行查询语句。

在 DBeaver 里运行查询步骤：

1. 在左侧边栏的“数据库”或者“项目”文件夹中，找到此前新建的数据库连接(参考[通过 DBeaver 连接 OceanBase 数据库](#))，点击鼠标右键，选择“SQL 编辑器”。
2. 在右边编辑器里输入查询语句，以分号结尾。然后点击编辑器侧边工具栏中“运行图标”或者快捷键“CTRL+回车”。
3. 下面“结果”栏目里显示了查询结果。
4. 如果有多条查询语句，每个语句都以“;”结尾。同时选中多条查询语句，然后执行快捷键“ALT+X”。
5. 下面“结果”栏目会顺序显示多个查询结果。
6. 默认结果集最多会展示 200 行记录，如果实际记录超过 200，需要滚动结果集到最后一行时 DBeaver 会自动获取下一个 200 行记录。



## 2.6.3 查询表里符合特定搜索条件的数据

当要查询满足特定搜索条件的数据时，给 SELECT 查询语句增加一个 WHERE 子句即可。

SQL 语句格式如下：

```
SELECT select_list FROM table_list
WHERE query_condition
```

接上图，查询 2 号仓库第 5 区的订单，SQL 语句如下：

```
SELECT * FROM ordr WHERE o_w_id=2 and o_d_id=5 ;
```

在 DBeaver 中查询如下：



The screenshot shows the DBeaver interface with the SQL query executed. The results are displayed in a table with the following columns: O\_W\_ID, O\_D\_ID, O\_ID, O\_C\_ID, O\_CARRIER\_ID, O\_OL\_CNT, O\_ALL\_LOCAL, and O\_ENTRY\_D.

|   | O_W_ID | O_D_ID | O_ID  | O_C_ID | O_CARRIER_ID | O_OL_CNT | O_ALL_LOCAL | O_ENTRY_D             |
|---|--------|--------|-------|--------|--------------|----------|-------------|-----------------------|
| 1 | 2      | 5      | 2,100 | 2,100  | 5            | 12       | 1           | 2020-02-15 18:59:25.0 |
| 2 | 2      | 5      | 2,101 | 4      | [NULL]       | 11       | 1           | 2020-02-15 18:59:25.0 |
| 3 | 2      | 5      | 2,102 | 440    | [NULL]       | 7        | 1           | 2020-02-15 18:59:25.0 |

## 2.6.4 对查询的结果进行排序

查询结果返回的顺序可能是任意顺序，如果想要根据特定字段返回结果集，需要在 SELECT 语句最后面增加 ORDER BY 子句指定返回顺序。

SQL 语句格式如下：

```
SELECT select_list FROM table_list
WHERE query_condition
ORDER BY column_list
```

如查看 2 号仓库第 5 区的客户，按姓名排序，SQL 语句如下：

```
SELECT c_id, c_last, c_first, c_middle, c_w_id, c_d_id, c_credit, c_since
FROM cust
WHERE c_w_id=2 AND c_d_id=5
ORDER BY c_last, c_first;
```

查询结果如下:

```

1= SELECT c_id, c_last, c_first, c_middle, c_w_id, c_d_id, c_credit, c_since
2= FROM cust
3= WHERE c_w_id=2 AND c_d_id=5
4= ORDER BY c_last, c_first;

```

|    | C_ID  | C_LAST         | C_FIRST          | C_MIDDLE | C_W_ID | C_D_ID | C_CREDIT | C_SINCE               |
|----|-------|----------------|------------------|----------|--------|--------|----------|-----------------------|
| 1  | 202   | ABLEBAROUGHT   | omHvv5eG         | OE       | 2      | 5      | GC       | 2020-02-15 18:59:14.0 |
| 2  | 272   | ABLECALLYOUHT  | x0ikkFsuG        | OE       | 2      | 5      | GC       | 2020-02-15 18:59:14.0 |
| 3  | 2,106 | ABLECALLYPRI   | Hksk96bvVRAZ     | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |
| 4  | 2,813 | ABLEPRESEING   | GxtYNqeRZFiaWmN  | OE       | 2      | 5      | GC       | 2020-02-15 18:59:16.0 |
| 5  | 2,360 | ANTIATIONATION | BclWNvikBc       | OE       | 2      | 5      | GC       | 2020-02-15 18:59:16.0 |
| 6  | 2,500 | ANTIEINGCALLY  | WXBmhcRTWe       | OE       | 2      | 5      | GC       | 2020-02-15 18:59:16.0 |
| 7  | 658   | ANTIESECALLY   | ONPLFp6Hti       | OE       | 2      | 5      | BC       | 2020-02-15 18:59:14.0 |
| 8  | 1,085 | ANTIOUGHTTABLE | SstdwQNJJ        | OE       | 2      | 5      | GC       | 2020-02-15 18:59:14.0 |
| 9  | 2,247 | ANTIPRIOUGHT   | R1VEFjdPIoloo    | OE       | 2      | 5      | GC       | 2020-02-15 18:59:16.0 |
| 10 | 2,100 | ATIONABLEESE   | PzFg81YvfBC      | OE       | 2      | 5      | BC       | 2020-02-15 18:59:15.0 |
| 11 | 2,196 | ATIONATIONBAR  | ReFfTMxfo1qG5    | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |
| 12 | 871   | ATIONCALLYBAR  | W7ZNCpPp5sBWNi   | OE       | 2      | 5      | GC       | 2020-02-15 18:59:14.0 |
| 13 | 2,999 | ATIONCALLYPRI  | eQblWEimKcM80xNB | OE       | 2      | 5      | GC       | 2020-02-15 18:59:16.0 |
| 14 | 849   | ATIONPRESATION | Z6V2PdM8B0b9hQ8d | OE       | 2      | 5      | GC       | 2020-02-15 18:59:14.0 |
| 15 | 8     | BARBARCALLY    | jXrw6CCdzcamEi   | OE       | 2      | 5      | GC       | 2020-02-15 18:59:13.0 |
| 16 | 4     | BARBARPRI      | KKvDumYX9AwHuxX  | OE       | 2      | 5      | GC       | 2020-02-15 18:59:13.0 |
| 17 | 1,983 | BAROUGHTANTI   | XeYW3WKpfBITC1aB | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |
| 18 | 1,590 | BAROUGHTCALLY  | Y886Pi3rFsHCSMbF | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |
| 19 | 1,529 | BAROUGHTPRI    | pyP6QJ7FB        | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |
| 20 | 34    | BARPRIPRI      | gXmuxvj2R        | OE       | 2      | 5      | GC       | 2020-02-15 18:59:13.0 |
| 21 | 1,785 | CALLYABLEESE   | LsWK1Zs1pm       | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |
| 22 | 2,765 | CALLYABLEOUGHT | b0maEo4Nm2JgW2j  | OE       | 2      | 5      | GC       | 2020-02-15 18:59:16.0 |
| 23 | 2,102 | CALLYABLEOUGHT | wPS9EgAgztLRvSuZ | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |
| 24 | 798   | CALLYEINGCALLY | Z9T8FivHqeaO     | OE       | 2      | 5      | GC       | 2020-02-15 18:59:14.0 |
| 25 | 2,793 | CALLYEINGOUGHT | eVQqIPDaGOb7N3EN | OE       | 2      | 5      | GC       | 2020-02-15 18:59:16.0 |
| 26 | 1,370 | CALLYESEING    | J8Caketz         | OE       | 2      | 5      | GC       | 2020-02-15 18:59:15.0 |

## 2.6.5 从多个表里查询数据

如果要查询的数据需要从多个表中取时, 需要在 SELECT 语句中 FROM 关键字后用 JOIN... ON 将多个表关联起来查询。通常这多个表在业务上是有联系的, 如某些字段值的定义和数据相同, 这个联系条件就是连接条件, 会体现在 ON 后面的括号里。ON 里也可以包括过滤条件。

SQL 语句格式如下:

```

SELECT select_list FROM table_name1 [INNER] JOIN table_name2 ON ( join_condition )
WHERE query_condition
ORDER BY column_list

```

### 2.6.5.1 返回满足连接条件的多个表数据

默认 JOIN 返回的结果会满足 ON 后面的连接条件, 这个又叫内连接 (INNER JOIN)。通常关键字 INNER 省略了就表示内连接。JOIN 前后的表分别称之为左表和右表, ON

里的条件描述的是左表和右表的连接条件和过滤条件。如果没有 ON 子句, 那 INNER JOIN 返回的就是左表和右表的全部数据, 这个也称为笛卡儿积。

SQL 返回的结果会在 JOIN 结果上再经过 WHERE 后的查询条件过滤 以及根据 ORDER BY 后的列排序。

### 示例：使用 JOIN 从多个表里查询数据

```
obclient> create table t1(id number not null primary key, name varchar(50));
Query OK, 0 rows affected (0.08 sec)

obclient> create table t2(id number not null primary key, name varchar(50));
Query OK, 0 rows affected (0.06 sec)

obclient> insert into t1 values(1,'A1'),(2,'B1'),(4,'D1'),(6,'F1'),(8,'H1'),(10,'J1');
Query OK, 6 rows affected (0.01 sec)
Records: 6 Duplicates: 0 Warnings: 0

obclient> insert into t2 values(1,'B2'),(3,'C2'),(6,'F2'),(9,'I2');
Query OK, 4 rows affected (0.01 sec)
Records: 4 Duplicates: 0 Warnings: 0

obclient> select t1.id, t1.name, t2.id, t2.name from t1 join t2 on (t1.id=t2.id) ;
+----+-----+-----+
| id | name | id | name |
+----+-----+-----+
| 1 | A1 | 1 | B2 |
| 6 | F1 | 6 | F2 |
+----+-----+-----+
2 rows in set (0.01 sec)
```

#### 2.6.5.2 返回包含不满足连接条件的多个表数据

当需要 JOIN 返回的数据除了符合连接条件和过滤条件的数据外, 还包括左表里满足左表的过滤条件但不满足连接条件的数据时, 就可以使用左外连接 (LEFT OUTER JOIN), 也可以简写为左连接 (LEFT JOIN)。左连接返回的结果里属于右表的数据如果不存在, 则该列返回 NULL。

反之, 如果 JOIN 返回的数据除了符合连接条件和过滤条件的数据外, 还包括右表里满足右表的过滤条件但不满足连接条件的数据时, 就可以使用右连接 (RIGHT OUTER JOIN), 或简写为右连接 (RIGHT JOIN)。右连接返回的结果里属于左表的数据如果不存在, 则该列返回 NULL。

### 示例：LEFT JOIN 和 RIGHT JOIN 示例

```
obclient> select t1.id, t1.name, t2.id, t2.name from t1 left join t2 on (t1.id=t2.id) ;
+----+-----+-----+
| id | name | id | name |
+----+-----+-----+
| 1 | A1 | 1 | B2 |
| 2 | B1 | NULL | NULL |
| 4 | D1 | NULL | NULL |
| 6 | F1 | 6 | F2 |
| 8 | H1 | NULL | NULL |
| 10 | J1 | NULL | NULL |
+----+-----+-----+
```



```

6 rows in set (0.00 sec)

obclient> select t1.id, t1.name, t2.id, t2.name from t1 right join t2 on (t1.id=t2.id) ;
+----+-----+-----+
| id | name | id | name |
+----+-----+-----+
| 1 | A1 | 1 | B2 |
| NULL | NULL | 3 | C2 |
| 6 | F1 | 6 | F2 |
| NULL | NULL | 9 | I2 |
+----+-----+-----+
4 rows in set (0.00 sec)

```

## 2.6.6 在查询中使用各种操作符、函数

SQL 查询语句格式中的 `select_list` 除了可以是表的列外，还可以是表达式。如对列使用操作符或者 SQL 函数的表达式。

### 2.6.6.1 查询中使用算术操作符

算术操作符包括：+ (加)、- (减)、\* (乘)、/ (除)、- (取反)、MOD (取模)。这些操作符可以作用在数值列上。

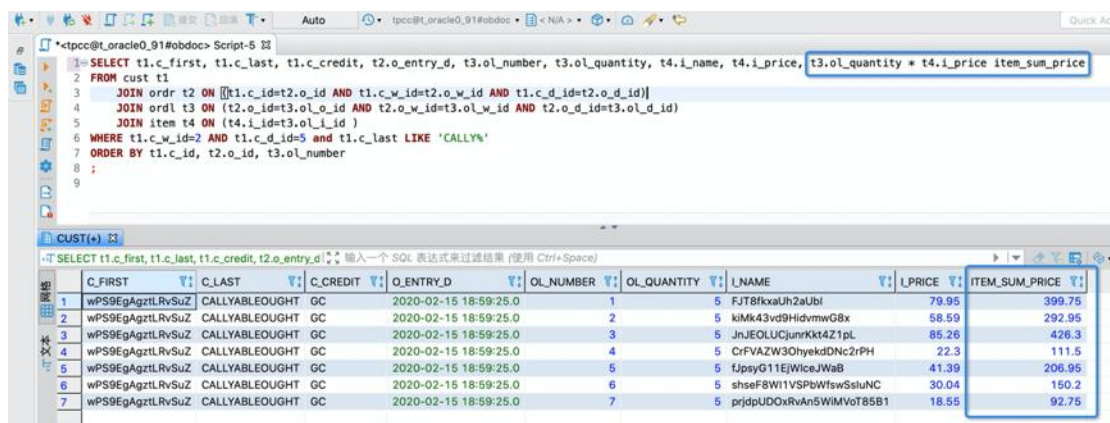
如“从多个表里查询数据”中示例查询出客户购买的每个商品的数量和价格，其中数量乘以价格就是每类商品的支付总额。所以 `select_list` 可以增加一列 `t3.ol_quantity * t4.i_price item_sum_price`。

```

SELECT t1.c_first, t1.c_last, t1.c_credit, t2.o_ol_cnt, t2.o_entry_d, t3.ol_number, t3.ol_quantity, t4.i_name,
t4.i_price, t3.ol_quantity * t4.i_price item_sum_price
FROM cust t1
JOIN ord t2 ON (t1.c_id=t2.o_id AND t1.c_w_id=t2.o_w_id AND t1.c_d_id=t2.o_d_id)
JOIN ord t3 ON (t2.o_id=t3.ol_o_id AND t2.o_w_id=t3.ol_w_id AND t2.o_d_id=t3.ol_d_id)
JOIN item t4 ON (t4.i_id=t3.ol_i_id)
WHERE t1.c_w_id=2 AND t1.c_d_id=5 and t1.c_last LIKE 'CALLY%'
ORDER BY t1.c_id, t2.o_id, t3.ol_number
;

```

查询结果如下：



|   | C_FIRST          | C_LAST       | C_CREDIT | O_ENTRY_D             | OL_NUMBER | OL_QUANTITY | LNAME                    | LPRICE | ITEM_SUM_PRICE |
|---|------------------|--------------|----------|-----------------------|-----------|-------------|--------------------------|--------|----------------|
| 1 | wPS9EgAgztLrvSuZ | CALLYBLEOUGH | GC       | 2020-02-15 18:59:25.0 | 1         | 5           | FJT8fkxUhz2aUbl          | 79.95  | 399.75         |
| 2 | wPS9EgAgztLrvSuZ | CALLYBLEOUGH | GC       | 2020-02-15 18:59:25.0 | 2         | 5           | kiMk43vd9HidvmwG8x       | 58.59  | 292.95         |
| 3 | wPS9EgAgztLrvSuZ | CALLYBLEOUGH | GC       | 2020-02-15 18:59:25.0 | 3         | 5           | JnJEOLUCjunrKkt4Z1pL     | 85.26  | 426.3          |
| 4 | wPS9EgAgztLrvSuZ | CALLYBLEOUGH | GC       | 2020-02-15 18:59:25.0 | 4         | 5           | CrFVAZW3OhyekdDNc2rPH    | 22.3   | 111.5          |
| 5 | wPS9EgAgztLrvSuZ | CALLYBLEOUGH | GC       | 2020-02-15 18:59:25.0 | 5         | 5           | fJpsyG11EjWiceJWaB       | 41.39  | 206.95         |
| 6 | wPS9EgAgztLrvSuZ | CALLYBLEOUGH | GC       | 2020-02-15 18:59:25.0 | 6         | 5           | shseF8W11VSPbWfswSsluNC  | 30.04  | 150.2          |
| 7 | wPS9EgAgztLrvSuZ | CALLYBLEOUGH | GC       | 2020-02-15 18:59:25.0 | 7         | 5           | prjdpUD0xRvAn5WIMVoT85B1 | 18.55  | 92.75          |

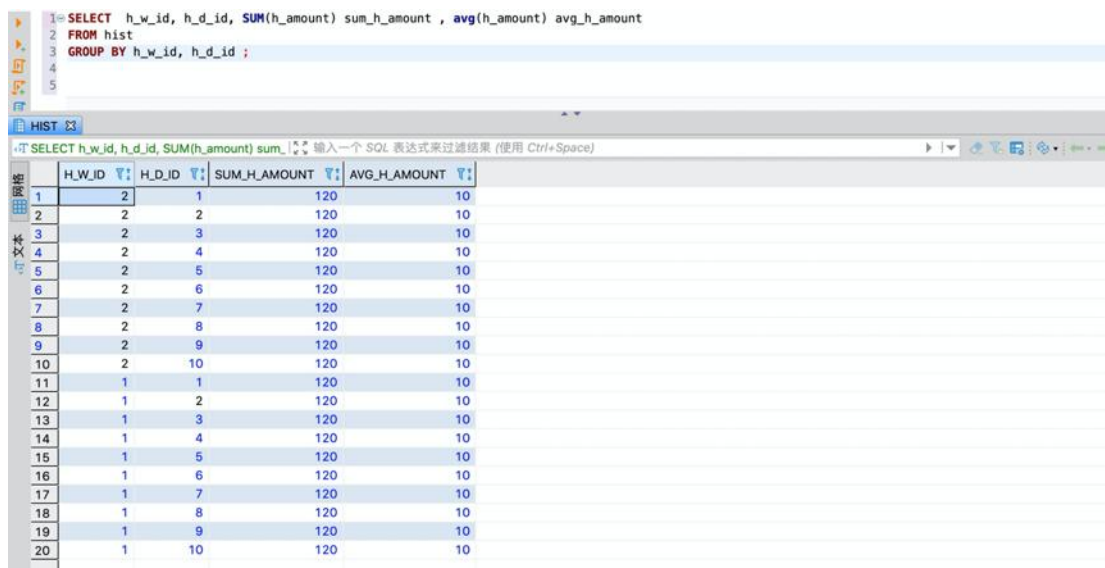
### 2.6.6.2 查询中使用数值函数

常用数值函数有：sum(求和)、avg(求平均)、ceil(向上取整)、floor(向下取整)、trunc(数值取整)、round(n)(四舍五入保留 n 位小数)。

如求历史表中每个仓库和区域的总销售额和平均每单销售额，SQL 如下：

```
SELECT h_w_id, h_d_id, sum(h_amount) sum_h_amount , avg(h_amount) avg_h_amount
FROM hist
GROUP BY h_w_id, h_d_id ;
```

查询结果如下：



|    | H_W_ID | H_D_ID | SUM_H_AMOUNT | AVG_H_AMOUNT |
|----|--------|--------|--------------|--------------|
| 1  | 2      | 1      | 120          | 10           |
| 2  | 2      | 2      | 120          | 10           |
| 3  | 2      | 3      | 120          | 10           |
| 4  | 2      | 4      | 120          | 10           |
| 5  | 2      | 5      | 120          | 10           |
| 6  | 2      | 6      | 120          | 10           |
| 7  | 2      | 7      | 120          | 10           |
| 8  | 2      | 8      | 120          | 10           |
| 9  | 2      | 9      | 120          | 10           |
| 10 | 2      | 10     | 120          | 10           |
| 11 | 1      | 1      | 120          | 10           |
| 12 | 1      | 2      | 120          | 10           |
| 13 | 1      | 3      | 120          | 10           |
| 14 | 1      | 4      | 120          | 10           |
| 15 | 1      | 5      | 120          | 10           |
| 16 | 1      | 6      | 120          | 10           |
| 17 | 1      | 7      | 120          | 10           |
| 18 | 1      | 8      | 120          | 10           |
| 19 | 1      | 9      | 120          | 10           |
| 20 | 1      | 10     | 120          | 10           |

### 2.6.6.3 查询中使用字符串连接符

字符串连接符，在 MySQL 租户中，' || ' 这个默认是表示逻辑运算符‘或’。MySQL 租户的字符串连接函数是 concat 、 concat\_ws 。

如查看 MySQL 租户下的客户姓名，SQL 语句如下：

```
obclient> SELECT concat_ws(' ', c_first, c_last) full_name FROM cust ORDER BY c_last LIMIT 2;

+-----+
| full_name |
+-----+
| fvBZoeIV2uJh7 ABLEABLEESEE |
| dHmIgRV1IsC ABLEABLEOUGHT |
+-----+
2 rows in set (0.01 sec)
```

但是如果把 MySQL 租户下的变量 `sql_mode` 值增加一个选项 `PIPES_AS_CONCAT` , 则 `' || '` 也会当作字符串连接符。SQL 语句如下:

```
obclient> SET SESSION sql_mode='PIPES_AS_CONCAT,STRICT_TRANS_TABLES,STRICT_ALL_TABLES';
obclient> SELECT c_first || ' ' || c_last full_name FROM cust ORDER BY c_last LIMIT 2;
+-----+
| full_name          |
+-----+
| fvBZoeIV2uJh7 ABLEABLEESE |
| dHmIgRV1IsC ABLEABLEOUGHT |
+-----+
2 rows in set (0.01 sec)
```

#### 2.6.6.4 查询中使用字符串函数

常用的字符串函数有求字符串长度(`length`)、字符串截取(`substr`)、字符串拼接、字符串转大小写(`upper` `lower`)、字符串删除前后缀(`ltrim` `rtrim` `trim`)。

需要注意的是, 在 MySQL 租户里, 字符串长度函数(`length`)长度单位是字节, `char_length` 函数的字符串长度单位是字符。

```
$obclient -h192.168.1.101 -utpcc@obmysql#obdemo -P2883 -p123456 -A tpccdb

obclient> select length('中'), char_length('中');
+-----+
| length('中') | char_length('中') |
+-----+
|          3  |          1        |
+-----+
1 row in set (0.00 sec)
```

#### 2.6.6.5 查询中使用时间函数

MySQL 租户常用的时间类型除了 `date` 和 `timestamp` 外还有很多。如 `time` , `datetime` , `year` 等。更多时间类型用法介绍, 请参考[《OceanBase SQL 参考 \( MySQL 租户\) 》](#)。

MySQL 租户常用的取数据库时间函数是 `now()` , `curdate()` 和 `curtime()` 。

##### 示例：格式化时间显示

MySQL 租户调整时间类型显示的格式, 可以用 `date_format` 函数, SQL 如下:

```
obclient> select now(), date_format(now(), "%Y/%m/%d %T") new_time ;
+-----+
| now()          | new_time          |
+-----+
```



```

+-----+-----+
| 2020-04-03 15:55:37 | 2020/04/03 15:55:37 |
+-----+-----+
1 row in set (0.00 sec)

```

### 示例：提取时间中的年/月/日/时/分/秒

MySQL 租户从时间中提取年/月/日/时/分/秒, 可以用 `extract` 函数, SQL 如下:

```

obclient> SET @dt = now();

obclient> SELECT @dt
, extract(YEAR FROM @dt)      d_year
, extract(MONTH FROM @dt)     d_month
, extract(week FROM @dt)      d_week
, extract(DAY FROM @dt)       d_day
, extract(HOUR FROM @dt)      d_hour
, extract(MINUTE FROM @dt)    d_min
, extract(SECOND FROM @dt)    d_second
, extract(year_month FROM @dt) d_year_month
, extract(hour_minute FROM @dt) d_hour_min
\G
***** 1. row *****
      @dt: 2020-03-27 18:00:52
      d_year: 2020
      d_month: 3
      d_week: 12
      d_day: 27
      d_hour: 18
      d_min: 0
      d_second: 52
      d_year_month: 202003
      d_hour_min: 1800
1 row in set (0.00 sec)

```

### 示例：时间类型加减

MySQL 租户对时间进行加减, 可以使用 `date_add` 或 `date_sub` 函数。

```

obclient> SET @dt = now();

obclient> SELECT @dt
, date_add(@dt, INTERVAL 1 DAY )      t1
, date_add(@dt, INTERVAL 1 HOUR )     t2
, date_add(@dt, INTERVAL -10 MINUTE ) t3
, date_add(@dt, INTERVAL -1 MONTH )   t4
, date_sub(@dt, INTERVAL 1 YEAR )     t5
\G
***** 1. row *****
@dt: 2020-03-27 18:03:44
t1: 2020-03-28 18:03:44
t2: 2020-03-27 19:03:44
t3: 2020-03-27 17:53:44
t4: 2020-02-27 18:03:44
t5: 2019-03-27 18:03:44

```

```
1 row in set (0.01 sec)
```

### 2.6.6.6 查询中使用类型转换函数

类型转换函数可以将一种数据类型转换为另外一种数据类型。如数值类型和时间类型到字符串类型的相互转换。数据库支持的类型转换函数请查看相应的 SQL 帮助手册。

#### 示例：时间字符串转换为时间类型

MySQL 租户中，时间字符串可以直接复制给 `date` 类型，MySQL 可以自动转换为时间类型。同时也可以使用 `convert` 或 `cast` 函数做类型转换。

```
obclient> SELECT CONVERT('2020-02-02 14:30:45', date)      t1
        , CONVERT('2020-02-02 14:30:45', time)            t2
        , CONVERT('2020-02-02 14:30:45', datetime)        t3
        , CAST('2020-02-02 14:30:45' AS date)              t4
        , CAST('2020-02-02 14:30:45' AS time)             t5
        , CAST('2020-02-02 14:30:45' AS datetime)         t6
\G
***** 1. row *****
t1: 2020-02-02
t2: 14:30:45
t3: 2020-02-02 14:30:45
t4: 2020-02-02
t5: 14:30:45
t6: 2020-02-02 14:30:45
1 row in set (0.00 sec)
```

时间类型转换为字符串类型，MySQL 租户可以使用函数 `date_format`。

#### 示例：数值类型和字符串类型互相转换

MySQL 租户中，数值类型和字符串类型互相转换，可以用函数 `convert`、`cast`。

```
obclient> SELECT convert('3.1415926', decimal)  n1
        , cast('3.1415926' AS decimal)         n2
        , convert(3.1415926, char(10))          s1
        , cast(3.1415926 AS char(10))           s2
;

+-----+-----+-----+-----+
| n1    | n2    | s1    | s2    |
+-----+-----+-----+-----+
| 3     | 3     | 3.1415926 | 3.1414926 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

### 2.6.6.7 查询中使用聚合函数

聚合函数扫描一组记录，然后返回单行记录。这组记录可以是一个表或者视图、或者一个子查询的结果。

OceanBase 数据库支持的聚合函数请查看 [OceanBase SQL 参考](#)。

聚合函数通常跟 `GROUP BY` 子句一起使用，按照一个或多个列的值分组，然后每组返回单笔记录。

**示例：分组统计每个仓库的销售额。**

MySQL 租户中，聚合函数跟 `GROUP BY` 子句一起使用的时候，对 `select_list` 里的列没有要求。这个可能会导致结果集很奇怪。如果要求 `select_list` 里的列跟 `GROUP BY` 子句中的列保持一致，需要设置 MySQL 命令行下的 `Sql_mode` 为 `'ONLY_FULL_GROUP_BY'`。SQL 查询如下：

```
obclient> SELECT ol_w_id
, count(*)                order_count
, sum(ol_amount)          sum_amount
, round(avg(ol_amount),2) avg_amount
, min(ol_amount)          min_amount
, max(ol_amount)          max_amount
FROM ordl
GROUP BY ol_w_id
ORDER BY ol_w_id ;
```

| ol_w_id | order_count | sum_amount | avg_amount | min_amount | max_amount |
|---------|-------------|------------|------------|------------|------------|
| 1       | 297         | 917174.33  | 3088.13    | 0.00       | 9876.11    |
| 2       | 329         | 1153354.23 | 3505.64    | 0.00       | 9979.34    |

2 rows in set (0.01 sec)

```
obclient> SELECT ol_w_id, ol_d_id
, count(*)                order_count
, sum(ol_amount)          sum_amount
, round(avg(ol_amount),2) avg_amount
, min(ol_amount)          min_amount
, max(ol_amount)          max_amount
FROM ordl
GROUP BY ol_w_id
ORDER BY ol_w_id
;
```

| ol_w_id | ol_d_id | order_count | sum_amount | avg_amount | min_amount | max_amount |
|---------|---------|-------------|------------|------------|------------|------------|
| 1       | 1       | 297         | 917174.33  | 3088.13    | 0.00       | 9876.11    |
| 2       | 1       | 329         | 1153354.23 | 3505.64    | 0.00       | 9979.34    |

2 rows in set (0.00 sec)

```
obclient> show variables like '%sql_mode%';
```

| Variable_name | Value   |
|---------------|---|
| sql_mode      | PIPES_AS_CONCAT,STRICT_TRANS_TABLES,STRICT_ALL_TABLES |

```

+-----+
1 row in set (0.00 sec)

obclient> SET SESSION sql_mode='STRICT_ALL_TABLES,ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

obclient> SELECT ol_w_id, ol_d_id
, count(*)                order_count
, sum(ol_amount)          sum_amount
, round(avg(ol_amount),2) avg_amount
, min(ol_amount)          min_amount
, max(ol_amount)          max_amount
FROM ordl
GROUP BY ol_w_id
ORDER BY ol_w_id
;

ERROR 1055 (42000): 'tpccdb.ordl.ol_d_id' is not in GROUP BY
obclient>

```

### 2.6.6.8 查询中使用 NULL 相关函数

NULL 相关的函数用于处理 NULL 值。相关的函数在 OceanBase SQL 参考手册中都有列举和详细描述。

NULL 值的特点是任何数值都不能等于 NULL 或不等于 NULL，但可以通过 IS NULL 判断。也可以使用 NVL 函数将 NULL 值转换为可识别的字符串。下面示例如何识别和转换 NULL 值。

#### 示例：NULL 值转换

MySQL 租户中，如果一个列可能有 NULL 值，可以使用 NVL 或 IFNULL 函数探测并转换为特殊字符。SQL 查询如下：

```

CREATE TABLE t_null(id number NOT NULL PRIMARY KEY, name varchar(10));
INSERT INTO t_null(id, name) values(1,'A'), (2,NULL), (3,'NULL');
SELECT id, name, nvl(name, 'NOT APPLICABLE') n_name, IFNULL(name, 'NOT APPLICABLE') n2_name
FROM t_null;

```

```

134 CREATE TABLE t_null(id number NOT NULL PRIMARY KEY, name varchar(10));
135 INSERT INTO t_null(id, name) values(1,'A'),(2,NULL),(3,'NULL');
136 select id, name, nvl(name, 'NOT APPLICABLE') n_name, ifnull(name, 'NOT APPLICABLE') n2_name
137 from t_null;

```

|   | id | name   | n_name         | n2_name        |
|---|----|--------|----------------|----------------|
| 1 | 1  | A      | A              | A              |
| 2 | 2  | [NULL] | NOT APPLICABLE | NOT APPLICABLE |
| 3 | 3  | NULL   | NULL           | NULL           |

### 2.6.6.9 查询中使用 Case 函数

CASE 表达式可以实现类似“IF…ELSE…THEN”的逻辑而不用调用子程序。CASE 表达式有两种使用方法，简单的和带搜索条件的。

#### 示例：在查询中使用简单的 CASE 表达式

用简单的 CASE 表达式将国家代码缩写翻译为全称。  
MySQL 租户 SQL 查询如下：

```
obclient> CREATE TABLE t_case(id number NOT NULL PRIMARY KEY, abbr varchar(5));
Query OK, 0 rows affected (0.08 sec)

obclient> INSERT INTO t_case(id, abbr) VALUES (1,'US'),(2,'UK'),(3,'CN'),(4,'JP');
Query OK, 4 rows affected (0.02 sec)
Records: 4 Duplicates: 0 Warnings: 0

obclient>
obclient> SELECT id, abbr,
CASE abbr
  WHEN 'US' THEN 'America'
  WHEN 'UK' THEN 'English'
  WHEN 'CN' THEN 'China'
  ELSE 'UNKOWN'
END full_name
FROM t_case ;

+---+-----+-----+
| id | abbr | full_name |
+---+-----+-----+
| 1  | US   | America   |
| 2  | UK   | English   |
| 3  | CN   | China     |
| 4  | JP   | UNKOWN    |
+---+-----+-----+
4 rows in set (0.00 sec)

obclient>
```

#### 示例：在查询中使用带搜索条件的 CASE 表达式

MySQL 租户 SQL 查询如下：

```
obclient> DROP TABLE IF EXISTS t_case2;
Query OK, 0 rows affected (0.02 sec)

obclient> CREATE TABLE t_case2(id number NOT NULL PRIMARY KEY, c_date date );
Query OK, 0 rows affected (0.14 sec)

obclient> INSERT INTO t_case2(id,c_date)
VALUES (1,'2019-03-01')
,(2,'2019-05-08')
,(3,'2019-07-07')
,(4,'2019-10-11')
,(5,'2019-12-12')
,(6,'2020-01-05');
Query OK, 6 rows affected (0.01 sec)
Records: 6 Duplicates: 0 Warnings: 0
```

```
obclient>
obclient> SELECT id, c_date,
CASE
  WHEN datediff(now(), c_date) > 12*30 THEN 'More than one year ago'
  WHEN datediff(now(), c_date) > 9*30 THEN 'More than three quarters ago'
  WHEN datediff(now(), c_date) > 6*30 THEN 'More than half a year ago'
  WHEN datediff(now(), c_date) > 3*30 THEN 'More than a quarter ago'
  WHEN datediff(now(), c_date) >= 0 THEN 'Within a quarter'
  ELSE 'Illegal'
END "Duration"
FROM t_case2;
+-----+-----+-----+
| id | c_date | Duration |
+-----+-----+-----+
| 1 | 2019-03-01 | More than one year ago |
| 2 | 2019-05-08 | More than three quarters ago |
| 3 | 2019-07-07 | More than three quarters ago |
| 4 | 2019-10-11 | More than a quarter ago |
| 5 | 2019-12-12 | More than a quarter ago |
| 6 | 2020-01-05 | Within a quarter |
+-----+-----+-----+
6 rows in set (0.01 sec)
```

### 2.6.6.10 锁定查询结果 SELECT FOR UPDATE

OceanBase 支持 MVCC 特性，读是快照读，不阻塞写。但是 SELECT 语句还有个特殊的用法可以阻塞写。示例如下：

```
obclient> select w_name, w_ytd, w_tax from ware where w_id=1 for update;
+-----+-----+-----+
| w_name | w_ytd | w_tax |
+-----+-----+-----+
| n1P4zYo80H | 1200.00 | 0.1868 |
+-----+-----+-----+
1 row in set (0.01 sec)
```

详细使用场景请参考 “[关于 SELECT ... FOR UPDATE 子句](#)”。

## 2.6.7 查看查询执行计划

OceanBase 的 SQL 引擎支持执行计划解析和缓存技术。可以通过分析查询语句的执行计划来分析查询性能。

OceanBase 执行计划查看方式非常简单，语法如下：

```
explain [extended] query_statement ;
```

示例： 查看查询执行计划

```
EXPLAIN select count(*) from TPCC.STOK \G
```

```
***** 1. row *****
Query Plan:
=====
|ID|OPERATOR          |NAME      |EST. ROWS|COST |
=====
|0 |SCALAR GROUP BY    |          |1         |16264|
|1 |EXCHANGE IN DISTR  |          |1         |16264|
|2 |EXCHANGE OUT DISTR|:EX10000|1         |16263|
|3 |MERGE GROUP BY     |          |1         |16263|
|4 |PX PARTITION ITERATOR|        |3732     |15551|
|5 |TABLE SCAN         |STOK      |3732     |15551|
=====

Outputs & filters:
-----
0 - output([T_FUN_COUNT_SUM(T_FUN_COUNT(*))], filter(nil),
  group(nil), agg_func([T_FUN_COUNT_SUM(T_FUN_COUNT(*))])
1 - output([T_FUN_COUNT(*)]), filter(nil)
2 - output([T_FUN_COUNT(*)]), filter(nil), dop=1
3 - output([T_FUN_COUNT(*)]), filter(nil),
  group(nil), agg_func([T_FUN_COUNT(*)])
4 - output([STOK.S_I_ID]), filter(nil)
5 - output([STOK.S_I_ID]), filter(nil),
  access([STOK.S_I_ID]), partitions(p[0-5])

1 row in set (0.00 sec)
```

**EXPLAIN**

```
SELECT /*+ no_use_px parallel(8) */ * FROM (
  SELECT o.o_w_id, o.o_d_id, o.o_id, o.o_ol_cnt, ol.count_ol
    FROM ordr o,
         (SELECT /*+ no_use_px parallel(8) */ ol_w_id, ol_d_id, ol_o_id, COUNT(*) count_ol
          FROM ordl GROUP BY ol_w_id, ol_d_id, ol_o_id) ol
   WHERE o.o_w_id = ol.ol_w_id AND o.o_d_id = ol.ol_d_id AND o.o_id = ol.ol_o_id
) x
WHERE o.ol_cnt != count_ol \G
```

```
***** 1. row *****
Query Plan:
=====
|ID|OPERATOR          |NAME      |EST. ROWS|COST |
=====
|0 |EXCHANGE IN DISTR  |          |88        |2972 |
|1 |EXCHANGE OUT DISTR|          |88        |2815 |
|2 |MERGE JOIN         |          |88        |2815 |
|3 |TABLE SCAN         |O         |180       |148  |
|4 |SUBPLAN SCAN       |OL        |60        |2472 |
|5 |MERGE GROUP BY     |          |60        |2464 |
|6 |TABLE SCAN         |ORDL      |1974      |1591 |
=====

Outputs & filters:
-----
0 - output([O.O_W_ID], [O.O_D_ID], [O.O_ID], [O.O_OL_CNT], [OL.COUNT_OL]), filter(nil)
1 - output([O.O_W_ID], [O.O_D_ID], [O.O_ID], [O.O_OL_CNT], [OL.COUNT_OL]), filter(nil), dop=8
2 - output([O.O_W_ID], [O.O_D_ID], [O.O_ID], [O.O_OL_CNT], [OL.COUNT_OL]), filter(nil),
  equal_conds([O.O_W_ID = OL.OL_W_ID], [O.O_D_ID = OL.OL_D_ID], [O.O_ID = OL.OL_O_ID]), other_conds([O.O_OL_CNT != OL.COUNT_OL])
3 - output([O.O_W_ID], [O.O_D_ID], [O.O_ID], [O.O_OL_CNT]), filter(nil),
  access([O.O_W_ID], [O.O_D_ID], [O.O_ID], [O.O_OL_CNT]), partitions(p[0-5])
4 - output([OL.OL_W_ID], [OL.OL_D_ID], [OL.OL_O_ID], [OL.COUNT_OL]), filter(nil),
  access([OL.OL_W_ID], [OL.OL_D_ID], [OL.OL_O_ID], [OL.COUNT_OL])
5 - output([ORDL.OL_W_ID], [ORDL.OL_D_ID], [ORDL.OL_O_ID], [T_FUN_COUNT(*)]), filter(nil),
  group([ORDL.OL_W_ID], [ORDL.OL_D_ID], [ORDL.OL_O_ID]), agg_func([T_FUN_COUNT(*)])
6 - output([ORDL.OL_W_ID], [ORDL.OL_D_ID], [ORDL.OL_O_ID]), filter(nil),
  access([ORDL.OL_W_ID], [ORDL.OL_D_ID], [ORDL.OL_O_ID]), partitions(p[0-5])

1 row in set (0.01 sec)
```

## 2.6.8 在查询中使用 SQL Hint

### 2.6.8.1 关于 SQL Hint

OceanBase SQL 的执行性能跟 SQL 的执行计划有关。执行计划跟表的连接方式、查询条件和表的索引都有关系。通常这是数据库的 SQL 引擎内部逻辑。但

是通过在 SQL 里添加注释，您可能改变执行计划的内容，从而改变 SQL 的执行性能。

SQL Hint 的格式是：

```
/*+ hint_text */
```

常用的 SQL Hint 有：

- **read\_consistency(weak)**：弱一致性读，指引 SQL 读取相关表的分区的备副本。有关分区的备副本概念请查看 [“关于表和分区”](#)。
- **index(table\_name, index\_name)**：指引 SQL 使用某个表的某个索引读取数据。
- **query\_timeout(int\_num)**：设置当前 SQL 的查询超时时间，单位是 us。

常用的 SQL Hints 请参考附录 [“OceanBase 常用 SQL Hints”](#)。

SQL Hint 的使用属于 OceanBase SQL 性能调优专题，详细使用方法请参考官网文档的“OceanBase SQL 调优指南”。

### 2.6.8.2 使用 SQL Hint

SQL Hint 通常用在 SQL 里，并不限于查询 SQL。这里以查询 SQL 为例，简单的语法格式如下：

```
SELECT /*+ hint_text [, hint_text] */ select_items FROM table_name
```

多个 SQL Hint 可以叠加使用，注意功能不要冲突。

**注意：**

在 obclient 命令行环境下，默认会忽略注释语法，SQL Hint 会不起作用。启动 obclient 时需要增加参数“-c”。有关 obclient 用法请参考 [“通过 obclient 连接 OceanBase 的租户”](#)。

**示例：在查询 SQL 中使用 SQL Hint**

下面 SQL 指定查询超时时间为 10 秒。

```
obclient> select /*+ query_timeout(10000000) */ o_id,o_c_id,o_carrier_id,o_ol_cnt,o_all_local,o_entry_d
from ordr
where o_w_id=1 and o_d_id=2 and o_id=2100;
+-----+-----+-----+-----+-----+-----+
| o_id | o_c_id | o_carrier_id | o_ol_cnt | o_all_local | o_entry_d |
+-----+-----+-----+-----+-----+-----+
| 2100 |      8 |           8 |       11 |           1 | 2020-02-15 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

obclient> SELECT /*+ no_use_px parat1el(8) */ * FROM(
SELECT /*+ no_use_px parallel(8) */ no_w_id, no_d_id, MAX(no_o_id) max_no_o_id, MIN(no_o_id) min_no_o_id,
```



```
COUNT(*) count_no
  FROM nord
  GROUP BY no_w_id, no_d_Id
) x
WHERE max_no_o_id - min_no_o_id+ 1!= count_no;
Empty set (0.01 sec)

obclient>
```

## 2.6.9 关于查询超时设计

OceanBase 租户内部执行查询的工作线程数跟租户的 CPU 个数相关, 所以工作线程是很稀有的资源。如果查询长时间不返回, 就会一直占有这个工作线程资源。OceanBase 为避免长时间不返回的查询长期占有工作线程资源以及可能占有 CPU 资源, 设计了查询超时功能。这个超时时间默认由租户参数(变量)

`ob_query_timeout` 控制, 默认值是 100000000(单位微秒)。当查询时间超过这个变量值后, 会返回错误 -4012(HY000): Timeout。

默认的超时参数对于 OLTP 类业务来说是合理的, 但是对于 OLAP 类业务就不一定。此时可以选择在会话级别调整租户变量的值。或者使用 SQL Hint 设置超时参数。查询超时的 Hint 格式是:

### 示例: 查询超时示例

```
#obclient -h127.1 -uroot@obmysql#obdemo -P3883 -p123456 -c -A

obclient> show variables like 'ob_query_timeout';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| ob_query_timeout | 100000000 |
+-----+-----+
1 row in set (0.01 sec)

obclient> set session ob_query_timeout=10000000;
Query OK, 0 rows affected (0.00 sec)

obclient> select sleep(11);
ERROR 4012 (HY000): Timeout
obclient>
obclient> select /*+ query_timeout(100000000) */ sleep(11);
+-----+
| sleep(11) |
+-----+
| 0 |
+-----+
1 row in set (11.00 sec)

obclient>
```

### 注意:

请不要把这个超时变量的值设置的太小或者为 0!

## 第 3 章 关于 DML 语句和事务

数据操作语言(DML)语句可以增加、修改和删除 OceanBase 的表里的数据。**事务**指的是一序列 SQL 语句, OceanBase 将这组 SQL 语句当作一个整体, 要么全部执行成功, 要么全部不成功; 不存在部分 SQL 语句执行成功, 或者部分 SQL 没有执行成功的情形。

通常事务中的 SQL 会包含 DML 语句，也会包含查询语句。如果一个事务中的 SQL 只有查询语句，这个事务通常称为只读事务。

本章话题包含：

- [关于数据操作语言 \(DML\) 语句](#)  
DML 语句访问和修改表中的数据。
- [关于事务控制语句](#)  
事务是一序列被 OceanBase 数据库当作整体的多个 SQL 语句，要么全部执行成功，要么全部不成功。当业务流程要求多个操作被当作一个整体执行时，您就需要事务来对这个业务流程建模。
- [提交事务](#)  
提交事务时事务的修改会永久保存、相关的锁和保存点会被释放和清除。
- [回滚事务](#)  
回滚事务时事务的修改会被撤销。您可以回滚整个事务，或者只回滚到事务中间某个保存点。事务回滚后，相关的锁和保存点会被释放和清除；如果是回滚到某个保存点，保存点之后的锁和其他保存点会被释放和清除。
- [在事务期间设置事务保存点](#)  
使用 `SAVEPOINT` 语句在事务中创建保存点——事务可以单独回滚到某个保存点。保存点在事务中是可选的，并且可以有多个。

## 3.1 关于数据操作语言 (DML) 语句

数据操作语言(DML)语句能够访问和操作表中的数据。

在 `obclient` 命令行环境下，您可以在 SQL 提示符 `>` 输入 DML 语句。

在 DBeaver 的 SQL 编辑器窗口，您也可以输入 DML 语句。

DML 语句对数据的修改效果只有在您提交事务时才永久生效。一个事务是一序列被 OceanBase 数据库当作整体的多个 SQL 语句，要么全部执行成功，要么全部不成功。单个 DML 语句也可以是一个事务。事务在没有提交之前，您还可以回滚事务。有关事务更多信息，请查看“[关于事务控制语句](#)”。

### 3.1.1 关于 INSERT 语句

INSERT 语句向表中插入行记录。

最简单的 INSERT 语句语法格式如下：

```
INSERT INTO table_name (list_of_columns) VALUES (list_of_values);
```

`list_of_columns` 指定的是表 `table_name` 的列, `list_of_values` 是 `list_of_columns` 提到的列的对应的值, 必须一一对应。因此, 在向一个表插入行记录之前, 您需要了解这个表所有的列信息, 以及列类型和有效值、是否允许为空等。在 `obclient` 命令行环境下, 您可以直接用 `DESC` 命令查看列属性。

```
obclient> desc ordl;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ol_w_id        | int(11)       | NO   | PRI | NULL    |      |
| ol_d_id        | int(11)       | NO   | PRI | NULL    |      |
| ol_o_id        | int(11)       | NO   | PRI | NULL    |      |
| ol_number      | int(11)       | NO   | PRI | NULL    |      |
| ol_delivery_d  | date          | YES  |     | NULL    |      |
| ol_amount      | decimal(6,2)  | YES  |     | NULL    |      |
| ol_i_id        | int(11)       | YES  |     | NULL    |      |
| ol_supply_w_id | int(11)       | YES  |     | NULL    |      |
| ol_quantity    | int(11)       | YES  |     | NULL    |      |
| ol_dist_info   | char(24)      | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

在 `INSERT` 语句中, 您不需要知道表的所有列的值, 但需要知道所有列属性为 `NOT NULL` 的列的值。如果列属性为 `NOT NULL` 有默认值时, 您可以不指定该列的值。如果列为 `NULL`, 您也可以不指定该列的值, OceanBase 会在该列上插入一个 `NULL` 值。

当插入多条件记录时, 可以分多笔 `INSERT` 语句, 也可以用一个 `INSERT` 多个 `VALUES` 语句。

### 示例：当所有列信息都知道时，使用 INSERT 语句

下面示例创建有默认值列的表, SQL 插入两笔记录, 所有字段信息都有值。

```
obclient> CREATE TABLE t_insert(
  id number NOT NULL PRIMARY KEY
  , name varchar(10) NOT NULL, value number
  , gmt_create timestamp NOT NULL DEFAULT current_timestamp
);
Query OK, 0 rows affected (0.07 sec)

obclient> INSERT INTO t_insert(id, name, value, gmt_create) values(1,'CN',10001, sysdate);
Query OK, 1 row affected (0.01 sec)
```

### 示例：当不是所有列信息都知道时，使用 INSERT 语句

下面 SQL 插入两笔记录, `gmt_create` 字段没有提供。两笔记录使用一个 `INSERT` 多个 `VALUES` 子句。

```
obclient> INSERT INTO t_insert(id, name, value)
VALUES (2,'US', 10002) ,(3,'EN', 10003);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

### 示例：使用 INSERT 语句报错：违反唯一约束冲突

当表上有唯一性约束的时候，插入相同的记录，数据库会报报错，提示 key ‘xxx’ violated。

```
obclient> INSERT INTO t_insert(id, name, value)
VALUES (3,'UK', 10003) ,(4, 'JP', 10004);
ERROR-00001: unique constraint '3' for key 'PRIMARY' violated
```

这个报错可以通过 INSERT IGNORE INTO (MySQL 租户)、MERGE INTO (MySQL 租户)、INSERT INTO ON DUPLICATE KEY UPDATE (MySQL 租户) 避免。

### 3.1.1.1 关于 INSERT IGNORE INTO 语句 (MySQL 租户)

下面示例是 MySQL 租户下 INSERT 用法，以及使用 INSERT IGNORE INTO 避免约束冲突。IGNORE 关键字可以忽略由于约束冲突导致的 INSERT 失败的影响。

```
obclient> CREATE TABLE t_insert(
  id number NOT NULL PRIMARY KEY
  , name varchar(10) NOT NULL
  , value number
  ,gmt_create timestamp NOT NULL DEFAULT current_timestamp
);

Query OK, 0 rows affected (0.68 sec)

obclient> INSERT INTO t_insert(id, name, value, gmt_create)
VALUES (1,'CN',10001, current_timestamp);

Query OK, 1 row affected (0.01 sec)

obclient> INSERT INTO t_insert(id, name, value)
VALUES (2,'US', 10002),(3,'EN', 10003);

Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

obclient> INSERT INTO t_insert(id, name, value)
VALUES (3,'UK', 10003),(4, 'JP', 10004);

ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY'
obclient>
obclient> INSERT IGNORE INTO t_insert(id, name, value)
VALUES (3,'UK', 10003) ,(4, 'JP', 10004);
Query OK, 1 row affected (0.00 sec)

obclient> select * from t_insert;
+----+-----+-----+-----+
| id | name | value | gmt_create |
+----+-----+-----+-----+
| 1 | CN   | 10001 | 2020-04-03 16:05:45 |
| 2 | US   | 10002 | 2020-04-03 16:05:54 |
| 3 | EN   | 10003 | 2020-04-03 16:05:54 |
| 4 | JP   | 10004 | 2020-04-03 16:06:08 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
```

**示例：使用查询语句充当 INSERT 的 values 子句**

当需要备份一个表的备份或者全部记录时，可以使用 `INSERT INTO ... SELECT ... FROM` 语句。

```
obclient> create table ware_bak(
  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)
, primary key(w_id)
);
Query OK, 0 rows affected (0.17 sec)

obclient> insert into ware_bak select * from ware;
Query OK, 2 rows affected (0.02 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

### 3.1.2 关于 UPDATE 语句

UPDATE 语句可以更新表的行记录。

简单的 UPDATE 语句语法如下：

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

`column_name` 是要更新的列，后面等号后面的 `value` 是这列要更新的目标值，必须符合列的类型定义。`WHERE` 条件子句指定要更新的行记录必须满足的条件，没有 `WHERE` 条件子句就是更新表所有记录。

**示例：更新所有记录，SQL 查询如下：**

```
obclient> update t_insert set value=value+1 ;
Query OK, 3 rows affected (0.01 sec)
Rows matched: 3 Changed: 3 Warnings: 0
```

不带条件更新的时候，如果记录数达到几十万或者几百万，会有大事务产生，可能会失败。所以 UPDATE 注意控制事务不要太大。

**示例：更新部分记录，违反约束报错：**

```
obclient> create unique index uk_name on t_insert(name);
Query OK, 0 rows affected (1.99 sec)

obclient> update t_insert set name='US' where id=3;
ERROR-00001: unique constraint 'US' for key 'UK_NAME' violated
```

除了显式的 UPDATE 语句外，还有几类语句也可以更新数据。比如说 INSERT 因为约束冲突失败的时候，可以使用 ON DUPLICATE KEY UPDATE 子句转变为 UPDATE 语句更新相关字段。

### 3.1.2.1 关于 INSERT ON DUPLICATE KEY UPDATE 子句 (MySQL 租户)

使用 ON DUPLICATE KEY UPDATE 子句有个要求，就是表上面要有主键或唯一约束(索引)。

示例：使用 INSERT ON DUPLICATE KEY UPDATE 避免数据插入冲突。

```
obclient> INSERT INTO t_insert(id, name, value) VALUES (3,'UK', 10003);
ERROR 1062 (23000): Duplicate entry '3' for key 'PRIMARY'

obclient>
obclient> INSERT INTO t_insert(id, name, value) VALUES (3,'UK', 10003)
ON DUPLICATE KEY UPDATE name='UK', value=10003 ;
Query OK, 2 rows affected (0.01 sec)

obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN | 10001 | 2020-04-03 16:05:45 |
| 2 | US | 10002 | 2020-04-03 16:05:54 |
| 3 | UK | 10003 | 2020-04-03 16:05:54 |
| 4 | JP | 10004 | 2020-04-03 16:06:08 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

### 3.1.2.2 关于 SELECT ... FOR UPDATE 子句

使用 SELECT ... FOR UPDATE 可以在读取记录的时候就对记录加锁，避免其他 DML 语句对该笔记录进行同时修改。这种设计通常也称为“悲观锁策略”

示例：使用 SELECT ... FOR UPDATE 先锁定记录后修改

```
obclient> select id,name,value from account where id=2 for update;
+-----+-----+-----+
| id | name | value |
+-----+-----+-----+
| 2 | B | 1000 |
+-----+-----+-----+
1 row in set (0.00 sec)

obclient> update account set value=value+100 where id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

obclient> commit;
Query OK, 0 rows affected (0.00 sec)
```

### 3.1.3 关于 DELETE 语句

DELETE 语句可以删除表中的记录。最简单的 DELETE 语句形式如下：

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

WHERE 条件是可选的，如果没有提供就是全表删除。如果表记录数多达几百万以上，会形成大事务，可能会有性能问题。建议带上 WHERE 条件分批删除；或者使用 TRUNCATE TABLE 语句。

示例 3-1-3-1 删除表中数据

```
delete from ordr where o_w_id=2;
delete from ordr ;
```

除了 DELETE 语句外，REPLACE INTO 语句也可能会删除数据。

### 3.1.4 关于 REPLACE INTO 语句 (MySQL 租户)

REPLACE INTO 语句会判断行记录是否存在(根据主键索引或唯一索引判断)。如果不存在，则插入记录；如果存在，则删除已有记录，并查询新行记录。目标表建议有主键或者唯一索引，否则容易插入重复的记录。

```
obclient> CREATE TABLE t_replace(
  id number NOT NULL PRIMARY KEY
  , name varchar(10) NOT NULL
  , value number
  ,gmt_create timestamp NOT NULL DEFAULT current_timestamp
);
Query OK, 0 rows affected (0.06 sec)

obclient> REPLACE INTO t_replace values(1,'CN',2001, current_timestamp ());
Query OK, 1 row affected (0.00 sec)

obclient> REPLACE INTO t_replace
  SELECT id,name,value,gmt_create FROM t_insert;
Query OK, 5 rows affected (0.00 sec)
Records: 4 Duplicates: 1 Warnings: 0

obclient> REPLACE INTO t_replace(id, name, value) values(6,'DE',20006);
Query OK, 1 row affected (0.01 sec)

obclient> select * from t_replace;
+----+-----+-----+-----+
| id | name | value | gmt_create          |
+----+-----+-----+-----+
| 1 | CN   | 10001 | 2020-04-03 16:05:45 |
| 2 | US   | 10002 | 2020-04-03 16:05:54 |
| 3 | UK   | 10003 | 2020-04-03 16:05:54 |
| 4 | JP   | 10004 | 2020-04-03 16:06:08 |
| 6 | DE   | 20006 | 2020-04-03 16:09:19 |
```



```
+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

## 3.2 关于事务控制语句

**事务**指的是一序列 SQL 语句，OceanBase 将这组 SQL 语句当作一个整体，要么全部执行成功，要么全部不成功；不存在部分 SQL 语句执行成功，或者部分 SQL 没有执行成功的情形。

举个例子，当一个客户下单购买了一批商品时，订单表会新增几行记录购买信息和金额，库存表会更新这批商品的库存数量(扣减库存)。应用程序中为这个业务做设计时，就需要把 INSERT 和 UPDATE 语句放到一个事务里。

基本的事务控制语句有：

- BEGIN，显式开启一个事务。这个命令是可选的，如果租户会话的参数 `autocommit` 值是 `off` (关闭自动提交)，就不需要显式发出这个命令；如果参数值是 `on` (开启自动提交)，那每句 SQL 就是一个独立的事务。如果要多个 SQL 组成一个事务，需要显式发起 `BEGIN` 命令。
- SAVEPOINT，在事务过程中标记一个“保存点”——事务可以事后选择回滚到这个点。保存点是可选的，一个事务过程中也可以有多个保存点。
- COMMIT，结束当前事务，让事务所有修改持久化并生效，清除所有保存点和释放事务持有的锁。
- ROLLBACK，回滚整个事务已做的修改或者只回滚某个保存点之后事务已做的修改。清除回滚部分包含的所有保存点和释放事务持有的锁。

在 `obclient` 命令环境下，你可以在 SQL 提示符后发起事务控制命令，也可以修改会话级别的 `autocommit` 参数。如果是修改租户级别的 `autocommit` 参数，需要断开会话重新连接才会生效。

在图形化客户端工具中，如 DBeaver 中，SQL 编辑窗口里可以发出命令，或者工具栏上有提交和回滚的图标。

### 注意：

如果会话的 `autocommit` 参数值是 `off` 时，并且您没有显式的提交事务，程序异常中断时，OceanBase 数据库会自动回滚最后一个未提交的事务。

OceanBase 建议显式的提交事务或者回滚事务。

## 3.3 提交事务

提交一个事务会让事务的修改持久化生效，清除保存点并释放事务持有的所有锁。

要显式的提交事务，使用 `COMMIT` 语句或者使用提交按钮(图形化客户端工具里)。

备注: OceanBase 会在 DDL 语句前和后隐式的发起一个 `COMMIT` 语句, 这个也会提交事务。

在您提交事务之前:

- 您的修改只对当前会话可见, 对其他数据库会话是不可见的。
- 您的修改没有持久化, 所以不是最终的——您可以用 `ROLLBACK` 语句回滚这些修改。

在您提交事务之后:

- 您的修改对所有数据库会话(包括自己)可见。
- 您的修改持久化成功——您不可以用 `ROLLBACK` 语句回滚这些修改。

**示例: obclient 提交事务.**

```
obclient> insert into t_insert(id,name) values(4,'JP');
Query OK, 1 row affected (0.01 sec)

obclient> commit;
Query OK, 0 rows affected (0.00 sec)

obclient> ^DBye

[qing.meiq@h07gl2088.sqa.eu95 /home/qing.meiq/bmsql]
$obclient -h192.168.1.101 -utpcc@obmysql#obdemo -P2883 -p123456 TPCC
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

obclient> select * from t_insert;
+----+-----+-----+-----+
| id | name | value | gmtime_create |
+----+-----+-----+-----+
| 1 | CN | 10001 | 2020-04-03 16:05:45 |
| 2 | US | 10002 | 2020-04-03 16:05:54 |
| 3 | UK | 10003 | 2020-04-03 16:05:54 |
| 4 | JP | 10004 | 2020-04-03 16:06:08 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
```

### 3.4 回滚事务

回滚一个事务指将事务的修改全部撤销。您可以回滚当前整个未提交事务, 或者也可以回滚到事务中任意一个保存点。

要回滚到某个保存点, 您必须使用 `ROLLBACK` 和 `TO SAVEPOINT` 语句结合使用。

如果是回滚整个事务:

- 事务会结束。
- 所有的修改会被丢弃。
- 清除所有保存点。
- 释放事务持有的所有锁。

如果是回滚到某个保存点:

- 事务不会结束。
- 保存点之前的修改被保留，保存点之后的修改被丢弃。
- 清除保存点之后的保存点(不包括保存点自身语句)。
- 释放保存点之后事务持有的所有锁

**示例：ROLLBACK 回滚掉事务全部修改。**

```
obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN   | 10001 | 2020-04-03 16:05:45 |
| 2 | US   | 10002 | 2020-04-03 16:05:54 |
| 3 | UK   | 10003 | 2020-04-03 16:05:54 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

obclient> insert into t_insert(id, name, value) values(4, 'DE', 10004);
Query OK, 1 row affected (0.00 sec)

obclient> insert into t_insert(id, name, value) values(5, 'FR', 10005), (6, 'RU', 10006);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN   | 10001 | 2020-04-03 16:05:45 |
| 2 | US   | 10002 | 2020-04-03 16:05:54 |
| 3 | UK   | 10003 | 2020-04-03 16:05:54 |
| 4 | DE   | 10004 | 2020-04-03 16:11:54 |
| 5 | FR   | 10005 | 2020-04-03 16:12:00 |
| 6 | RU   | 10006 | 2020-04-03 16:12:00 |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)

obclient> rollback;
Query OK, 0 rows affected (0.00 sec)

obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN   | 10001 | 2020-04-03 16:05:45 |
| 2 | US   | 10002 | 2020-04-03 16:05:54 |
| 3 | UK   | 10003 | 2020-04-03 16:05:54 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

### 3.5 事务保存点

SAVEPOINT，在事务过程中标记一个“保存点”——事务可以事后选择回滚到这个点。保存点是可选的，一个事务过程中也可以有多个保存点。

下面示例展示了一个事务中包含多个 DML 语句和多个保存点，然后回滚到其中一个保存点，只丢弃了保存点后面的那部份修改。

## 示例：将一个事务回滚到一个保存点

查看表当前记录。

```
obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN | 10001 | 2020-04-03 16:05:45 |
| 2 | US | 10002 | 2020-04-03 16:05:54 |
| 3 | UK | 10003 | 2020-04-03 16:05:54 |
| 4 | JP | NULL | 2020-04-03 16:16:46 |
| 5 | DE | NULL | 2020-04-03 16:16:46 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

开启一个事务，设置多个保存点信息。

```
obclient> set session autocommit=off;
Query OK, 0 rows affected (0.00 sec)

obclient> insert into t_insert(id, name) values(6,'FR');
Query OK, 1 row affected (0.00 sec)

obclient> savepoint fr;
Query OK, 0 rows affected (0.00 sec)

obclient> insert into t_insert(id, name) values(7,'RU');
Query OK, 1 row affected (0.00 sec)

obclient> savepoint ru;
Query OK, 0 rows affected (0.00 sec)

obclient> insert into t_insert(id, name) values(8,'CA');
Query OK, 1 row affected (0.00 sec)

obclient> savepoint ca;
Query OK, 0 rows affected (0.00 sec)
```

当前会话能看到事务未提交的所有修改。

```
obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN | 10001 | 2020-04-03 16:05:45 |
| 2 | US | 10002 | 2020-04-03 16:05:54 |
| 3 | UK | 10003 | 2020-04-03 16:05:54 |
| 4 | JP | NULL | 2020-04-03 16:16:46 |
| 5 | DE | NULL | 2020-04-03 16:16:46 |
| 6 | FR | NULL | 2020-04-03 16:26:22 |
| 7 | RU | NULL | 2020-04-03 16:26:32 |
| 8 | CA | NULL | 2020-04-03 16:26:42 |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```

回滚事务到其中一个保存点。

```
obclient> rollback to savepoint ru;
Query OK, 0 rows affected (0.00 sec)
```

```
obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN | 10001 | 2020-04-03 16:05:45 |
| 2 | US | 10002 | 2020-04-03 16:05:54 |
| 3 | UK | 10003 | 2020-04-03 16:05:54 |
| 4 | JP | NULL | 2020-04-03 16:16:46 |
| 5 | DE | NULL | 2020-04-03 16:16:46 |
| 6 | FR | NULL | 2020-04-03 16:26:22 |
| 7 | RU | NULL | 2020-04-03 16:26:32 |
+-----+-----+-----+-----+
7 rows in set (0.01 sec)
```

提交事务，确认表最新修改包含保存点之前的修改。

```
obclient> commit;
Query OK, 0 rows affected (0.00 sec)
obclient> select * from t_insert;
+-----+-----+-----+-----+
| id | name | value | gmtime_create |
+-----+-----+-----+-----+
| 1 | CN | 10001 | 2020-04-03 16:05:45 |
| 2 | US | 10002 | 2020-04-03 16:05:54 |
| 3 | UK | 10003 | 2020-04-03 16:05:54 |
| 4 | JP | NULL | 2020-04-03 16:16:46 |
| 5 | DE | NULL | 2020-04-03 16:16:46 |
| 6 | FR | NULL | 2020-04-03 16:26:22 |
| 7 | RU | NULL | 2020-04-03 16:26:32 |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)

obclient>
```

### 3.6 关于事务超时

OceanBase 为了避免事务长时间不提交持有锁影响其他会话，设计了两个超时逻辑。一个是事务未提交超时、一个是事务空闲超时。分别由租户变量 `ob_trx_idle_timeout` 和 `ob_trx_timeout` 控制。默认值分别是 120 秒和 100 秒。通常只会有一个超时机制被触发，具体是哪个先超时。

```
obclient> show variables where variable_name in ('ob_trx_idle_timeout','ob_trx_timeout');
+-----+-----+
| Variable_name | Value |
+-----+-----+
| ob_trx_idle_timeout | 120000000 |
| ob_trx_timeout | 100000000 |
+-----+-----+
2 rows in set (0.00 sec)
```

### 3.6.1 关于事务空闲超时

OceanBase 的事务空闲时间超过一段时间还没有提交时，会自动断开连接并回滚事务。此时会话需要重新连接。

会话事务空闲超时时间阈值是由租户变量 `ob_trx_idle_timeout` 控制。这个参数值建议不要小于 100 秒。实际空闲会话断开的的时间会是在 `[100s, 100s+ob_trx_idle_timeout]` 之间。

#### 示例：事务空闲超时报错

下面示例先设置 事务空闲超时时间为 120 秒，事务未提交超时时间为 1000 秒。当事务空闲时间超过 120 秒后，在 100 秒内会连接被自动断开，事务也自动被 ROLLBACK 了。

```
obclient> DROP TABLE IF EXISTS t_insert;
Query OK, 0 rows affected (0.01 sec)

obclient> CREATE TABLE t_insert(
  id bigint NOT NULL PRIMARY KEY auto_increment
  , name varchar(10) NOT NULL
  , value bigint
  ,gmt_create timestamp NOT NULL DEFAULT current_timestamp
);
Query OK, 0 rows affected (0.05 sec)

obclient> INSERT INTO t_insert(name, value) values('CN',NULL),('UK',NULL),('US',NULL);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

obclient> commit;
Query OK, 0 rows affected (0.00 sec)

obclient> select now(), * from t_insert t;
+-----+-----+-----+-----+-----+
| now() | id | name | value | gmt_create |
+-----+-----+-----+-----+-----+
| 2020-04-03 16:54:51 | 1 | CN | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:54:51 | 2 | UK | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:54:51 | 3 | US | NULL | 2020-04-03 16:54:49 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

obclient> set session ob_trx_timeout=1000000000;
Query OK, 0 rows affected (0.00 sec)

obclient> set session ob_trx_idle_timeout=120000000;
Query OK, 0 rows affected (0.00 sec)

obclient> update t_insert set gmt_create=now() where id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

obclient> select now(), t.* from t_insert t;
+-----+-----+-----+-----+-----+
| now() | id | name | value | gmt_create |
+-----+-----+-----+-----+-----+
| 2020-04-03 16:55:30 | 1 | CN | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:55:30 | 2 | UK | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:55:30 | 3 | US | NULL | 2020-04-03 16:55:25 |
+-----+-----+-----+-----+-----+
```

```

3 rows in set (0.00 sec)

<<等100秒不操作>>

obclient> select now(), t.* from t_insert t;
ERROR-02013: Lost connection to MySQL server during query
obclient> select now(), * from t_insert t;
ERROR-02006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 53246
Current database: TPCC

+-----+-----+-----+-----+-----+
| now() | id | name | value | gmtime |
+-----+-----+-----+-----+-----+
| 2020-04-03 16:57:41 | 1 | CN | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:57:41 | 2 | UK | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:57:41 | 3 | US | NULL | 2020-04-03 16:54:49 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

### 3.6.2 关于事务未提交超时

OceanBase 的事务持续时间超过一段时间还没有提交，会报超时错误。此时会话需要明确发出 ROLLBACK 命令才可以继续在会话里执行 SQL。

#### 示例：事务未提交超时报错

会话事务的超时时间阈值是由租户变量 `ob_trx_timeout` 控制。下面示例先设置事务空闲超时时间为 120 秒，事务超时时间为 100 秒。当一个事务未提交时间持续到 100 秒时，事务内部状态就变为超时状态，同时锁会释放。此后会话需要显式发出 ROLLBACK 语句。

```

obclient> set session ob_trx_timeout=100000000;
Query OK, 0 rows affected (0.00 sec)

obclient> set session ob_trx_idle_timeout=120000000;
Query OK, 0 rows affected (0.00 sec)

obclient> update t_insert set gmtime=sysdate() where id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

obclient> select now(), t.* from t_insert t;
+-----+-----+-----+-----+-----+
| now() | id | name | value | gmtime |
+-----+-----+-----+-----+-----+
| 2020-04-03 16:59:56 | 1 | CN | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:59:56 | 2 | UK | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 16:59:56 | 3 | US | NULL | 2020-04-03 16:59:51 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

<<等120秒不操作>>

obclient> select now(), t.* from t_insert t;
ERROR-00600: internal error code, arguments: -6210, Transaction is timeout

```

```
obclient> commit;
ERROR-00600: internal error code, arguments: -6210, Transaction is timeout
obclient> rollback;
Query OK, 0 rows affected (0.00 sec)

obclient> select now(), t.* from t_insert t ;
+-----+-----+-----+-----+-----+
| now() | id | name | value | gmt_create |
+-----+-----+-----+-----+-----+
| 2020-04-03 17:04:13 | 1 | CN | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 17:04:13 | 2 | UK | NULL | 2020-04-03 16:54:49 |
| 2020-04-03 17:04:13 | 3 | US | NULL | 2020-04-03 16:54:49 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

### 注意:

请不要把事务未提交超时参数设置的非常小!



## 第 4 章 创建和管理数据库对象

如果要创建、修改和删除数据库对象，您可以用数据操作语言( DDL )。本章主要介绍 MySQL 租户里创建和管理数据库对象的方法。

本章话题:

- [关于数据定义语言 \(DDL\) 语句](#)  
数据操作语言( DDL )语句创建、修改和删除数据库对象。在 DDL 开始之前和之后，OceanBase 数据库会发出一个隐式的 COMMIT 语句。因此您不可以回滚 DDL 语句。
- [创建和管理表](#)  
表是最基础的数据存储单元。表包含所有用户可以访问的数据，每个表包含很多行记录，每个记录有多个列组成。
- [创建和管理表分组](#)  
表分组是表的属性，表分组会影响多个表在 OceanBase 机器上的分布特征。
- [创建和管理视图](#)  
视图代表表的一个查询结果。大部分可以使用表的场景里，您也可以使用视图。视图还有个好处就是可以将多个表的信息组合在一起返回。

### 4.1 关于数据定义语言 (DDL) 语句

数据操作语言( DDL )语句创建、修改和删除数据库对象。在 DDL 开始之前和之后，OceanBase 数据库会发出一个隐式的 COMMIT 语句。因此您不可以回滚 DDL 语句。

在 obclient 命令环境下，您可以在 SQL 提示符(obclient>)后发起 DDL 语句。

在 DBeaver 图形化环境下，您可以在 SQL 编辑器里输入 DDL 语句，分号结尾，然后点击“执行当前 SQL”图标。

DDL 语句主要是创建、修改和删除数据库对象。OceanBase DDL 支持的数据库对象有：

- 表，包括约束、索引。
- 视图
- 自定义函数
- 自定义类型

## 4.2 创建和管理表

表是最基础的数据存储单元。表包含所有用户可以访问的数据，每个表包含很多行记录，每个记录有多个列组成。

### 4.2.1 关于 SQL 数据类型

当您创建表的时候，你必须指定表记录行的每一列的数据类型，数据类型定义了该列存储数据的合法格式。比如说一个 DATE 类型的列，能存储值 ‘2020-02-20’，但是不能存储值为 2 的数字或者字符串 ‘hello’。

SQL 的数据类型分为两类：数据库内部自带的类型和用户自定义的类型。(STORED PROCEDURE 有额外新增的类型)。

有关数据库自带的类型，请参考《OceanBase SQL 参考》。这里介绍一些常用的数据类型。

| 分类                | 类型                                  | 备注  |
|-------------------|-------------------------------------|---|
| 数值类型-整形           | bigint                              | 有符号: $[-2^{63}, 2^{63} - 1]$<br>无符号: $[0, 2^{64} - 1]$  |
|                   | int<br>integer                      | 有符号: $[-2^{31}, 2^{31} - 1]$<br>无符号: $[0, 2^{32} - 1]$  |
|                   | smallint                            | 有符号: $[-2^{15}, 2^{15} - 1]$<br>无符号: $[0, 2^{16} - 1]$  |
|                   | bool<br>boolean<br>tinyint          | 有符号: $[-2^7, 2^7 - 1]$<br>无符号: $[0, 2^8 - 1]$   |
| 数值类型-定点           | decimal(p, s)                       | decimal 等同于 numeric   |
| 数值类型-浮点           | float                               | 有符号: $[-2^{128}, 2^{128}]$<br>无符号: $[-2^{1024}, 2^{1024}]$<br>精度 7 位  |
|                   | double                              | 有符号: $[-2^{1024}, 2^{1024}]$<br>无符号: $[0, 2^{1024}]$<br>精度 15 位   |
| 数值类型-<br>整形/定点/浮点 | number<br>number(p)<br>number(p, s) | p 和 s 都有表示定点数。p(precision)为精度，s(scale)表示小数点右边的数字个数，精度最大值为 38，scale 的取值范围为-84 到 127。<br>s 为 0 表示整形。<br>p 和 s 都不指定，表示浮点数，最大精度 38。 |
| 字符类型-变长           | varchar(N)                          | 最长 256K，字符集 UTF8MB4   |
|                   | varbinary                           | 最初 256K，字符集 BINARY  |
|                   | enum                                | 最多 65535 个元素，每个元素最长 255 个字符，字符集 UTF8MB4   |
|                   | set                                 | 最多 64 个元素，每个元素最长 255 个字符，   |

|         |                    |   |
|---------|--------------------|---|
|         |                    | 字符集 UTF8MB4                                   |
| 字符类型-定长 | char(N)            | 最大 256, 字符集 UTF8MB4                           |
|         | binary             | 最大 256, 字符集 BINARY                            |
| 时间类型    | date               | YYYY-MM-DD, 只包含日期                             |
|         | time               | HH:MM:SS[fraction], 只包含时间。                    |
|         | datetime           | YYYY-MM-DD HH:MM:SS[fraction], 包含日期时间(不考虑时区)。 |
|         | Timestamp          | 日期时间(考虑时区)。                                   |
|         | year               | YYYY,[1901, 2155]                             |
| 大对象     | Text / blob        | 最大 64K  |
|         | Longtext /longblob | 最大 48M  |

## 4.2.2 创建表

要创建表, 通过 DBeaver 或者 obclient 命令行都可以执行 DDL 语句 CREATE TABLE。

### 4.2.2.1 使用 CREATE TABLE 语句建表

本节演示如何使用 CREATE TABLE 语句创建订单表 ware 和 cust 表。

```
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)
);

Query OK, 0 rows affected (0.09 sec)

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)
);
```

Query OK, 0 rows affected (0.10 sec)

### 注意:

由于 ALTER TABLE 语法不支持后期增加主键，所以在建表的时候设置主键就更加有必要了。

#### 4.2.2.2 使用 CREATE TABLE 复制表结构

在 MySQL 租户里，可以使用 CREATE TABLE AS SELECT 复制表的数据，但是结构并不完全一致，会丢失约束、索引、默认值、分区等信息。使用 CREATE TABLE LIKE 可以复制表结构，但是不包括数据。

#### 示例：MySQL 租户的 CREATE TABLE 复制表结构和数据的区别

```
obclient> create table t1(
    id bigint not null primary KEY
    , name varchar(50) not NULL
    , gmt_create timestamp not null default current_timestamp
) partition by hash(id) partitions 8;
Query OK, 0 rows affected (0.10 sec)

obclient> insert into t1(id,name) values(1,'A'),(2,'B'),(3,'C');
Query OK, 3 rows affected (0.03 sec)
Records: 3 Duplicates: 0 Warnings: 0

obclient> create table t1_like like t1;
Query OK, 0 rows affected (0.11 sec)

obclient> create table t1_copy as select * from t1;
Query OK, 3 rows affected (0.12 sec)

obclient> show create table t1_like\G
***** 1. row *****
      Table: t1_like
Create Table: CREATE TABLE `t1_like` (
  `id` bigint(20) NOT NULL,
  `name` varchar(50) NOT NULL,
  `gmt_create` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = DYNAMIC COMPRESSION = 'zstd_1.0' REPLICA_NUM = 3 BLOCK_SIZE = 16384
USE_BLOOM_FILTER = FALSE TABLET_SIZE = 134217728 PCTFREE = 10
  partition by hash(id) partitions 8

1 row in set (0.00 sec)

obclient> show create table t1_copy\G
***** 1. row *****
      Table: t1_copy
```

```

Create Table: CREATE TABLE `t1_copy` (
  `id` bigint(20) DEFAULT NULL,
  `name` varchar(50) DEFAULT NULL,
  `gmt_create` timestamp NULL DEFAULT NULL
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = DYNAMIC COMPRESSION = 'zstd_1.0' REPLICA_NUM = 3 BLOCK_SIZE = 16384
USE_BLOOM_FILTER = FALSE TABLET_SIZE = 134217728 PCTFREE = 10
1 row in set (0.00 sec)

obclient> show create table t1\G
***** 1. row *****
      Table: t1
Create Table: CREATE TABLE `t1` (
  `id` bigint(20) NOT NULL,
  `name` varchar(50) NOT NULL,
  `gmt_create` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = DYNAMIC COMPRESSION = 'zstd_1.0' REPLICA_NUM = 3 BLOCK_SIZE = 16384
USE_BLOOM_FILTER = FALSE TABLET_SIZE = 134217728 PCTFREE = 10
  partition by hash(id) partitions 8

1 row in set (0.00 sec)

```

#### 4.2.2.3 关于表和分区

前面提到，您使用表存储数据。但是在 OceanBase 里，数据表示的最小粒度是分区。普通的非分区表，就只有一个分区；而分区表，通常有多个分区，分区名默认以 p 开头，按数字顺序从 0 开始编号。所以分区是表的子集。

通常分区对您的应用是透明的，应用只需要使用 SQL 读写表即可。只有某些场景下，为了提升分区表的查询性能，应用也可以使用 SQL 直接访问某个具体的分区，SQL 语法格式是

```
SELECT ... FROM parted_table PARTITION (pN) WHERE query_condition ;
```

#### 示例：通过 SQL 直接访问分区表的分区

```

obclient> select o_id,o_c_id,o_carrier_id,o_ol_cnt,o_all_local,o_entry_d from odr partition (p1) where o_w_id=1
and o_d_id=2 and o_id=2100;
+-----+-----+-----+-----+-----+-----+
| o_id | o_c_id | o_carrier_id | o_ol_cnt | o_all_local | o_entry_d |
+-----+-----+-----+-----+-----+-----+
| 2100 |      8 |           8 |      11 |           1 | 2020-02-15 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)

obclient> select ol_o_id, ol_number,ol_delivery_d,ol_amount,ol_i_id,ol_supply_w_id,ol_quantity from ordl
partition (p1) where ol_w_id=1 and ol_d_id=2 and ol_o_id=2100;
+-----+-----+-----+-----+-----+-----+-----+
| ol_o_id | ol_number | ol_delivery_d | ol_amount | ol_i_id | ol_supply_w_id | ol_quantity |
+-----+-----+-----+-----+-----+-----+-----+
| 2100 | 1 | 2020-02-15 | 0.00 | 87133 | 1 | 5 |
| 2100 | 2 | 2020-02-15 | 0.00 | 47413 | 1 | 5 |
| 2100 | 3 | 2020-02-15 | 0.00 | 9115 | 1 | 5 |
| 2100 | 4 | 2020-02-15 | 0.00 | 42985 | 1 | 5 |
| 2100 | 5 | 2020-02-15 | 0.00 | 43621 | 1 | 5 |
| 2100 | 6 | 2020-02-15 | 0.00 | 5787 | 1 | 5 |
| 2100 | 7 | 2020-02-15 | 0.00 | 62576 | 1 | 5 |
| 2100 | 8 | 2020-02-15 | 0.00 | 91592 | 1 | 5 |
| 2100 | 9 | 2020-02-15 | 0.00 | 34452 | 1 | 5 |

```

```

| 2100 | 10 | 2020-02-15 | 0.00 | 13792 | 1 | 5 |
| 2100 | 11 | 2020-02-15 | 0.00 | 94326 | 1 | 5 |
+-----+-----+-----+-----+-----+-----+
11 rows in set (0.01 sec)

```

在 OceanBase 里，节点间的数据迁移的最小粒度是分区，每个分区在集群里有三个副本，内容保持同步，角色上有区分。三副本会有一个主副本( Leader 副本)和两个备副本( Follower 副本)，只有主副本可以提供写服务，默认也只有主副本可以提供读服务。主副本上的事务提交时会把事务日志同步到两个备副本，三副本使用 Paxos 协议表决事务是否提交成功。有时候为了不影响主副本，可以让备副本承担部分读请求，这就是应用常用的**读写分离**的解决方案，在 OceanBase 里，这种读备称为**弱一致性读**。使用这种方案，应用读需要承担读延时的风险，这个延时最大允许值会通过参数( max\_stale\_time\_for\_weak\_consistency )控制。

### 示例：使用 SQL Hint 实现读写分离。

弱一致读的 Hint 语法是 `/*+ read_consistency(weak) */`。通常的读默认是强一致性读，就不用 Hint 了。有关 SQL Hint 用法请参考“[在查询中使用 SQL Hint](#)”。

```

obclient> select /*+ read_consistency(weak) */ o_id,o_c_id,o_carrier_id,o_ol_cnt,o_all_local,o_entry_d from order
where o_w_id=1 and o_d_id=2 and o_id=2100;
+-----+-----+-----+-----+-----+-----+
| o_id | o_c_id | o_carrier_id | o_ol_cnt | o_all_local | o_entry_d |
+-----+-----+-----+-----+-----+-----+
| 2100 | 8 | 8 | 11 | 1 | 2020-02-15 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

obclient>

```

#### 4.2.2.4 特殊的表：复制表

复制表是分布式数据库 OceanBase 的高级优化手段。

通常 OceanBase 集群是三副本架构，默认每个表的每个分区在 OceanBase 中会有三个副本数据，角色上分为一个主副本( Leader 副本)和两个备副本( Follower 副本)。默认提供读写服务的是主副本。

复制表的特别之处可以指定在租户的每台机器上都有一个备副本，并且这个主副本跟所有备份的数据使用全同步策略保持强同步。这样做的目的是为了业务有些 SQL 关联查询时能在同一节点内部执行，以获取更好的性能。

复制表的语法是在 CREATE TABLE 语句后增加下列选项：

#### 示例 4-2-2-4-1：创建复制表

```

create table item (i_id int
, i_name varchar(24)

```

```
, i_price decimal(5,2)
, i_data varchar(50)
, i_im_id int
, primary key(i_id)) COMPRESS FOR QUERY pctfree=0 BLOCK_SIZE=16384
duplicate_scope='cluster' locality='F,R{all_server}@doc_1,
F,R{all_server}@doc_2,F,R{all_server}@doc_3' primary_zone='doc_1';
```

## 4.2.3 关于列的自增

如果创建表时需要某个数值列的值不重复并且保持递增，这就是自增列。

MySQL 租户的表支持自增列类型。

### 4.2.3.1 CREATE TABLE 使用自增列 ( AUTO\_INCREMENT )

在 MySQL 租户里，列的类型可以定义为 `AUTO_INCREMENT`，这就是 MySQL 租户的自增列。

自增列有三个重要属性：自增起始值、自增步长、自增列缓存大小。这是通过三个租户变量参数控制。

```
obclient> show variables where variable_name in
('auto_increment_increment','auto_increment_offset','auto_increment_cache_size');
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_cache_size | 1000000 |
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
+-----+-----+
3 rows in set (0.01 sec)
```

#### 示例：CREATE TABLE 使用自增列

下面创建了一个自增列，在使用 `INSERT` 插入记录的时候不需要指定自增列，OceanBase 数据库会自动为该列填充值时。

```
obclient> create table t1(id bigint not null auto_increment primary key, name varchar(50), gmt_create timestamp
not null default current_timestamp);
Query OK, 0 rows affected (0.08 sec)

obclient> insert into t1(name) values('A'),('B'),('C');
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

obclient> select * from t1;
+-----+-----+
| id | name | gmt_create |
+-----+-----+
```

```

| 1 | A | 2020-04-03 17:09:55 |
| 2 | B | 2020-04-03 17:09:55 |
| 3 | C | 2020-04-03 17:09:55 |
+-----+-----+-----+
3 rows in set (0.01 sec)

```

如果在 INSERT 时指定了自增列的值，如果这个值是 0，则 OceanBase 数据库会用自增列的下一个值填充列的值；如果这个值比当前最大值小，则不影响自增列的下一个值的计算；如果这个值比当前值最大值要大，则自增列会把插入值和自增列缓存值的和作为下次自增的起始值。

```

obclient> insert into t1(id, name) values(0, 'D');
Query OK, 1 row affected (0.00 sec)

obclient> insert into t1(id, name) values(-1, 'E');
Query OK, 1 row affected (0.00 sec)

obclient> insert into t1(id, name) values(10, 'F');
Query OK, 1 row affected (0.01 sec)

obclient> insert into t1(name) values('G');
Query OK, 1 row affected (0.00 sec)

obclient> select * from t1;
+-----+-----+-----+
| id      | name |gmt_create      |
+-----+-----+-----+
| -1      | E    | 2020-04-03 17:10:24 |
| 1       | A    | 2020-04-03 17:09:55 |
| 2       | B    | 2020-04-03 17:09:55 |
| 3       | C    | 2020-04-03 17:09:55 |
| 4       | D    | 2020-04-03 17:10:19 |
| 10      | F    | 2020-04-03 17:10:29 |
| 1000011 | G    | 2020-04-03 17:10:34 |
+-----+-----+-----+
7 rows in set (0.00 sec)

```

## 4.2.4 关于列的约束类型

为了确保表里的数据符合业务规则，您可以在列上定义约束。

约束定义在列上，限制了列里存储的值。当尝试在该列上写入或更新为违反约束定义的值时，会触发一个错误并回滚这个操作。反之，当尝试在已有的表的列上加上一个跟现有数据相冲突的约束时，也会触发一个错误并回滚这个操作。

约束可以被启用或禁用。默认情况下，创建后状态会是启用的。

约束的类型有：

- **非空约束 (NOT NULL)**，不允许约束包含的列的值包含 NULL。  
如 ware 表的 w\_name 列类型后面有 not null 约束，表示业务约束每个仓库必须有个名称。有非空约束的列，在 INSERT 语句中必须指明该列的值，除非该列还定义了默认值。如 cust 表的列 c\_discount 定义了默认值 0.99，即业务上每个人默认折扣是 0.99。



- **唯一约束( UNIQUE )**，不允许约束包含的列的值有重复值，但是可以有多个 NULL 值。  
如 ware 表的 (w\_name, w\_city) 列上有个唯一约束，表示每个城市里仓库的名称必须是不重复的。
- **主键约束( PRIMARY KEY )**，是 NOT NULL 约束和唯一约束的组合。  
如 ware 表和 cust 表都有个主键 w\_id 和 c\_id，这两列不允许为 NULL 并且必须是不重复的。
- **外键约束( FOREIGN KEY )**，要求约束的列的值取自于另外一个表的主键列。  
如 cust 表的 c\_w\_id 上有个外键约束引用了 ware 表的 w\_id 列，表示业务上顾客归属的仓库必须是属于仓库表里仓库。  
OceanBase 租户默认是关闭外键约束的，通过租户变量 foreign\_key\_checks 控制。

#### 注意：

目前 OceanBase 版本不支持通过 ALTER TABLE 语句事后增加或修改约束，所以您需要在 CREATE TABLE 的事后就能确定好表的约束。

#### 4.2.4.1 关于时间列的默认时间设置

当列上有 NOT NULL 约束时，通常建议设置默认值。当列类型是日期或时间类型时，可以设置默认值为数据库当前时间。

##### 示例 4-2-4-1-2：MySQL 租户表的时间列设置默认值

OceanBase 的 MySQL 租户可以设置多个时间列有默认值。

```
obclient> create table t1(
  id bigint not null primary KEY
  , gmt_create datetime not null default current_timestamp
  , gmt_modified datetime not null default current_timestamp
);
Query OK, 0 rows affected (0.07 sec)

obclient> insert into t1(id) values(1),(2),(3);
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

obclient> select * from t1;
+-----+-----+-----+
| id | gmt_create          | gmt_modified          |
+-----+-----+-----+
| 1 | 2020-02-27 17:09:23 | 2020-02-27 17:09:23 |
| 2 | 2020-02-27 17:09:23 | 2020-02-27 17:09:23 |
| 3 | 2020-02-27 17:09:23 | 2020-02-27 17:09:23 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

## 4.2.5 关于表的索引

您可以在表的一个或多个列上创建索引以加速表上的 SQL 语句执行速度。索引使用正确的话，可以减少物理 IO 或者逻辑 IO。

当您创建表时同时设置了主键时，OceanBase 数据库会默认创建一个唯一索引。您可以通过下面 SQL 观察到这个。

```
obclient> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.01 sec)

obclient> CREATE TABLE t1(id bigint not null primary key, name varchar(50));
Query OK, 0 rows affected (0.05 sec)

obclient> show indexes from t1\G
***** 1. row *****
      Table: t1
    Non_unique: 0
      Key_name: PRIMARY
Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
     Comment: available
Index_comment:
       Visible: YES
1 row in set (0.01 sec)
```

### 4.2.5.1 教程：新增索引

向表增加索引可以通过 obclient 命令行环境，或者通过 DBeaver 工具的 SQL 编辑器。OceanBase 能在普通表和分区表上创建索引，索引可以是本地索引或者全局索引。同时索引可以是唯一索引或者普通索引。如果是分区表的唯一索引，唯一索引必须包含表分区的拆分键。

创建索引的 SQL 语法格式如下：

```
CREATE [UNIQUE] INDEX index_name ON table_name ( column_list ) [LOCAL | GLOBAL] [ PARTITION
BY column_list PARTITIONS N ] ;
```

MySQL 里索引名称只要求在表内不能重复。

MySQL 租户下查看索引通过命令 SHOW INDEXES 。

### 4.2.5.2 教程：新增和删除索引(MySQL 租户)

在 MySQL 租户里，新增索引还有一种 SQL 语法格式，如下：

```
ALTER TABLE table_name
    ADD|DROP INDEX|KEY index_name ( column_list ) ;
```

MySQL 租户可以一次加多个索引。索引关键字用 INDEX 或 KEY 都可以。

### 示例：对分区表新增索引(MySQL 租户)

```
obclient> create table t1(id bigint not null primary key ,name varchar(50) not null);
Query OK, 0 rows affected (0.06 sec)

obclient> show indexes from t1;
***** 1. row *****
      Table: t1
      Non_unique: 0
      Key_name: PRIMARY
      Seq_in_index: 1
      Column_name: id
      Collation: A
      Cardinality: NULL
      Sub_part: NULL
      Packed: NULL
      Null:
      Index_type: BTREE
      Comment: available
      Index_comment:
      Visible: YES
1 row in set (0.00 sec)

obclient> create table t3(
    id bigint not null primary KEY
    , name varchar(50)
    , gmt_create timestamp not null default current_timestamp
) partition by hash(id) partitions 8;
Query OK, 0 rows affected (0.14 sec)

obclient> alter table t3 add unique key t3_uk (name) local;
ERROR 1503 (HY000): A UNIQUE INDEX must include all columns in the table's partitioning function

obclient> alter table t3
    add unique key t3_uk (name, id) LOCAL
    , add key t3_ind3(gmt_create) global;
Query OK, 0 rows affected (18.03 sec)

obclient> show indexes from t3;
```

```
obclient> show indexes from t3;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| t3    | 0          | PRIMARY | 1            | id          | A         | NULL       | NULL    | NULL  | NULL | BTREE      | available |               | YES    |
| t3    | 0          | t3_uk   | 1            | name        | A         | NULL       | NULL    | NULL  | YES  | BTREE      | available |               | YES    |
| t3    | 0          | t3_uk   | 2            | id          | A         | NULL       | NULL    | NULL  | NULL | BTREE      | available |               | YES    |
| t3    | 1          | t3_ind2 | 1            | gmt_create  | A         | NULL       | NULL    | NULL  | NULL | BTREE      | available |               | YES    |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

MySQL 租户下删除索引语法格式如下：

```
ALTER TABLE table_name DROP key|index index_name ;
```

### 示例 4-2-4-3-2：删除索引(MySQL 租户)

```
obclient> alter table t3 drop key t3_uk, drop key t3_ind3;
Query OK, 0 rows affected (0.07 sec)
```

## 4.2.6 闪回被删除的表

OceanBase 支持回收站功能，默认回收站是开启的。回收站开启/关闭是由租户的变量 `recyclebin` 控制。

开启/关闭回收站的语法是：

```
set global recyclebin = ON | OFF ;
```

只对后续新连接会话生效。

**示例：闪回被删除的表**

```
obclient> drop table if exists t1;
obclient> create table t1(id bigint not null primary key, gmt_create datetime not null default current_timestamp);
Query OK, 0 rows affected (0.09 sec)

obclient> insert into t1(id) values(1),(2),(3);
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

obclient> select * from t1;
+-----+
| ID | GMT_CREATE |
+-----+
| 1 | 2020-02-28 09:47:07 |
| 2 | 2020-02-28 09:47:07 |
| 3 | 2020-02-28 09:47:07 |
+-----+
3 rows in set (0.00 sec)

obclient> drop table t1;
Query OK, 0 rows affected (0.03 sec)

obclient> show recyclebin;
***** 1. row *****
OBJECT_NAME: __recycle_$_20200102_1585650066255592
ORIGINAL_NAME: t1
TYPE: TABLE
CREATETIME: 2020-03-31 18:21:06.255716
1 row in set (0.03 sec)

obclient> flashback table __recycle_$_20200102_1585650066255592 to before drop rename to t1;
Query OK, 0 rows affected (0.02 sec)

obclient> select * from t1;
+-----+
| ID | GMT_CREATE |
+-----+
| 1 | 2020-02-28 09:47:07 |
| 2 | 2020-02-28 09:47:07 |
| 3 | 2020-02-28 09:47:07 |
+-----+
```

## 4.3 创建和管理分区表

分区技术(Partitioning)是 OceanBase 非常重要的分布式能力之一,它能帮助您解决大表的容量问题和高并发访问时性能问题,主要思想就是将大表拆分为更多更小的结构相同的独立的对象,即分区。普通的表只有一个分区,可以看作分区表的特例。每个分区只能存在于一个节点内部,分区表的不同分区可以分散在不同节点内部。

### 4.3.1 分区路由

OceanBase 的分区表是内建功能,您只需要在建表的时候指定分区策略和分区数即可。分区表的查询 SQL 跟普通表是一样的, OceanBase 的 OBProxy 或 OBCServer 会自动将用户 SQL 路由到相应节点内,因此,分区表的分区细节对业务是透明的。

当然,您也可以通过 SQL 直接访问分区表的某个分区,前提是您知道要读取的数据所在的分区号。

### 4.3.2 分区策略

OceanBase 支持多种分区策略:

- 范围(RANGE)分区
- RANGE COLUMNS 分区
- 列表(LIST)分区
- LIST COLUMNS 分区
- 哈希(HASH)分区
- 组合分区

#### 4.3.2.1 范围(RANGE)分区

范围分区根据分区表定义时为每个分区建立的分区键值范围,将数据映射到相应的分区中。它是常见的分区类型,经常跟日期类型一起使用。比如说,您可能期望将业务日志表按日/周/月分区。

范围分区的语法形式简单如下:

```
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) ]
);

```

当使用范围分区时，您需要遵守如下几个规则：

- PARTITION BY RANGE ( expr ) 里的 expr 表达式的结果必须为整形。
- 每个分区都有一个 VALUES LESS THAN 子句，它为分区指定一个非包含的上限值。分区键的任何值等于或大于这个值时将被映射到下一个分区中。
- 除第一个分区外，所有分区都隐含一个下限值，即上一个分区的上限值。
- 允许且只允许最后一个分区上限定义为 MAXVALUE，这个值没有具体的数值，比其他所有分区上限都要大，也包含空值。

### 示例：创建一个范围分区表

```

CREATE TABLE t_log_part_by_range (
  log_id      bigint NOT NULL
  , log_value varchar(50)
  , log_date  timestamp NOT NULL
) PARTITION BY RANGE(UNIX_TIMESTAMP(log_date))
(
  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'))
);

```

范围分区可以新增、删除分区。如果最后一个范围分区指定了 MAXVALUE，则不能新增分区，必须是分裂分区(SPLIT PARTITION)。OceanBase 当前版本还不支持分裂分区操作。

RANGE 分区要表拆分键表达式的结果必须为整型，如果要按时间类型列做 RANGE 分区，则必须使用 timestamp 类型，并且使用函数 UNIX\_TIMESTAMP 将时间类型转换为数值。不过这个需求我们也可以使用 RANGE COLUMNS 分区实现，就没有整型这个要求。

### 4.3.2.2 RANGE COLUMNS 分区

RANGE COLUMNS 分区作用跟 RANGE 分区基本类似，不同之处在于：

- range columns 拆分列结果不要求是整型，可以是任意类型。
- range columns 拆分列不能使用表达式。
- range columns 拆分列可以写多个列(即列向量)。

RANGE COLUMNS 分区的语法形式简单如下：

```
CREATE TABLE table_name (
    column_name1          column_type
    [, column_nameN          column_type]
) PARTITION BY RANGE ( column_name1 [, column_name2] )
(
    PARTITION p0          VALUES LESS THAN ( expr )
    [, PARTITION pN          VALUES LESS THAN ( expr ) ]
    [, PARTITION pX      VALUES LESS THAN ( maxvalue ) ]
);
```

示例：创建一个 RANGE COLUMNS 分区

```
CREATE TABLE t_log_part_by_range_columns (
    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
);
```

### 4.3.2.3 列表(LIST)分区

列表分区使得您可以显式的控制记录行如何映射到分区，具体方法是为每个分区的分区键指定一组离散值列表，这点跟范围分区和哈希分区都不同。列表分区的优点是可以自然的对无序或无关的数据集进行分区。

LIST 分区的语法形式简单如下：

```
CREATE TABLE table_name (
  column_name1      column_type
  [, column_nameN      column_type]
) PARTITION BY LIST ( expr(column_name1) )
(
  PARTITION p0      VALUES IN ( v01 [, v0N] )
  [, PARTITION pN      VALUES IN ( vN1 [, vNN] ) ]
  [, PARTITION pX      VALUES IN (default) ]
);
```

当使用列表分区时，需要遵守几个规则：

- 分区表达式结果必须是整型。
- 分区表达式只能引用一列，不能有多列(即列向量)。

**示例：创建一个列表分区表**

```
CREATE TABLE t_part_by_list (
  c1 BIGINT PRIMARY KEY
  , c2 VARCHAR(50)
) PARTITION BY list(c1)
(
  PARTITION p0 VALUES IN (1, 2, 3)
  , PARTITION p1 VALUES IN (5, 6)
  , PARTITION p2 VALUES IN (DEFAULT)
);
```

列表分区可以新增分区，指定新的不重复的列表。也可以删除分区。

#### 4.3.2.4 LIST COLUMNS 分区

LIST COLUMNS 分区作用跟 LIST 分区基本相同，不同之处在于：

- LIST COLUMNS 的拆分列不能是表达式。
- LIST COLUMNS 的拆分列可以是多列(即列向量)。

LIST COLUMNS 分区的语法形式简单如下：

```
CREATE TABLE table_name (
  column_name1      column_type
  [, column_nameN      column_type]
) PARTITION BY LIST COLUMNS ( column_name1 [, column_nameN ] )
(
  PARTITION p0      VALUES IN ( v01 [, v0N] )
  [, PARTITION pN      VALUES IN ( vN1 [, vNN] ) ]
);
```



```
[, PARTITION pX VALUES IN (default) ]
);
```

示例： 创建 LIST COLUMNS 分区。

#### 4.3.2.5 哈希(HASH)分区

哈希分区适合于对不能用范围分区、列表分区方法的场景，它的实现方法简单，通过对分区键上的 HASH 函数值来散列记录到不同分区中。如果您的数据符合下列特点，使用哈希分区是个很好的选择：

- 您不能指定数据的分区键的列表特征。
- 不同范围内的数据大小相差非常大，并且很难手动调整均衡。
- 使用范围分区后数据聚集会非常厉害。
- 并行 DML、分区剪枝和分区连接等性能非常重要。

示例： 创建一个哈希分区表

```
CREATE TABLE ware(
  w_id number
  , w_ytd number(12,2)
  , w_tax number(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)
  , primary key(w_id)
)tablegroup tpcc_group
PARTITION by hash(w_id) partitions 60;
```

哈希分区不适合做删除操作。当前版本还不支持分区分裂操作( split partition )。

#### 4.3.2.6 组合分区

组合分区通常是先使用一种分区策略，然后在子分区再使用另外一种分区策略，适合于业务表的数据量非常大时。使用组合分区能发挥多种分区策略的优点。

在指定二级分区分区策略细节时，可以使用 SUBPARTITION TEMPLATE 子句。OceanBase 当前版本不支持不同一级分区下使用不同定义的二级分区。

示例： 创建组合分区表

```
CREATE TABLE t_ordr_part_by_hash_range (o_w_id int
, o_d_id int,
```

```

, o_id int
, o_c_id int
, o_carrier_id int
, o_ol_cnt int
, o_all_local int
, o_entry_d date
, index idx_ordr(o_w_id, o_d_id, o_c_id, o_id) LOCAL
, primary key ( o_w_id, o_d_id, o_id, o_entry_d )
)
PARTITION BY hash(o_w_id)
SUBPARTITION BY RANGE(o_entry_d)
SUBPARTITION template
(
    SUBPARTITION M202001 VALUES LESS THAN(TO_DATE('2020/02/01','YYYY/MM/DD'))
    , SUBPARTITION M202002 VALUES LESS THAN(TO_DATE('2020/03/01','YYYY/MM/DD'))
    , SUBPARTITION M202003 VALUES LESS THAN(TO_DATE('2020/04/01','YYYY/MM/DD'))
    , SUBPARTITION M202004 VALUES LESS THAN(TO_DATE('2020/05/01','YYYY/MM/DD'))
    , SUBPARTITION M202005 VALUES LESS THAN(TO_DATE('2020/06/01','YYYY/MM/DD'))
    , SUBPARTITION M202006 VALUES LESS THAN(TO_DATE('2020/07/01','YYYY/MM/DD'))
    , SUBPARTITION M202007 VALUES LESS THAN(TO_DATE('2020/08/01','YYYY/MM/DD'))
    , SUBPARTITION M202008 VALUES LESS THAN(TO_DATE('2020/09/01','YYYY/MM/DD'))
    , SUBPARTITION M202009 VALUES LESS THAN(TO_DATE('2020/10/01','YYYY/MM/DD'))
    , SUBPARTITION M202010 VALUES LESS THAN(TO_DATE('2020/11/01','YYYY/MM/DD'))
    , SUBPARTITION M202011 VALUES LESS THAN(TO_DATE('2020/12/01','YYYY/MM/DD'))
    , SUBPARTITION M202012 VALUES LESS THAN(TO_DATE('2021/01/01','YYYY/MM/DD'))
    , SUBPARTITION MMAX VALUES LESS THAN MAXVALUE
)
partitions 16;

CREATE TABLE t_log_part_by_range_hash (
    log_id          number NOT NULL
    , log_value varchar2(50)
    , log_date date NOT NULL DEFAULT sysdate
    , PRIMARY key(log_id, log_date)
) PARTITION BY RANGE(log_date)
SUBPARTITION BY HASH(log_id) SUBPARTITIONS 16
(
    PARTITION M202001 VALUES LESS THAN(TO_DATE('2020/02/01','YYYY/MM/DD'))
    , PARTITION M202002 VALUES LESS THAN(TO_DATE('2020/03/01','YYYY/MM/DD'))
    , PARTITION M202003 VALUES LESS THAN(TO_DATE('2020/04/01','YYYY/MM/DD'))
    , PARTITION M202004 VALUES LESS THAN(TO_DATE('2020/05/01','YYYY/MM/DD'))
    , PARTITION M202005 VALUES LESS THAN(TO_DATE('2020/06/01','YYYY/MM/DD'))
    , PARTITION M202006 VALUES LESS THAN(TO_DATE('2020/07/01','YYYY/MM/DD'))
    , PARTITION M202007 VALUES LESS THAN(TO_DATE('2020/08/01','YYYY/MM/DD'))
    , PARTITION M202008 VALUES LESS THAN(TO_DATE('2020/09/01','YYYY/MM/DD'))
    , PARTITION M202009 VALUES LESS THAN(TO_DATE('2020/10/01','YYYY/MM/DD'))
    , PARTITION M202010 VALUES LESS THAN(TO_DATE('2020/11/01','YYYY/MM/DD'))
    , PARTITION M202011 VALUES LESS THAN(TO_DATE('2020/12/01','YYYY/MM/DD'))
    , PARTITION M202012 VALUES LESS THAN(TO_DATE('2021/01/01','YYYY/MM/DD'))
    , PARTITION MMAX VALUES LESS THAN MAXVALUE
);

```

由于 OceanBase 里二级分区必须使用 SUBPARTITION TEMPLATE，如果二级分区是 RANGE 则不支持对二级分区进行 ADD/DROP 操作。所以通常强烈建议使用 RANGE-HASH 这种组合分区方式而不是 HASH-RANGE 组合分区。

### 4.3.3 分区表的索引

分区表的查询性能跟 SQL 中条件有关。当 SQL 中带上拆分键时，OceanBase 会根据条件做分区剪枝，只用搜索特定的分区即可。如果没有拆分键时，则要扫描所有分区。

分区表也可以通过创建索引来提升性能。跟分区表一样，分区表的索引也可以分区或者不分区。

如果分区表的索引不分区，就是一个全局索引(GLOBAL)，是一个独立的分区，索引数据覆盖整个分区表。

如果分区表的索引分区了，根据分区策略又可以分为两类。一是跟分区表保持一致的分区策略，则每个索引分区的索引数据覆盖相应的分区表的分区，这个索引又叫本地索引(LOCAL)。

OceanBase 建议尽可能的使用本地索引，只有在有必要的时候才使用全局索引。其原因是全局索引会降低 DML 的性能，DML 可能会因此产生分布式事务。

通常创建索引时默认都是全局索引，本地索引需要在后面增加关键字 LOCAL。

示例 4-3-3-1：创建分区表的本地索引

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

### 4.3.4 分区表使用建议

通常当表的数据量非常大，以致于可能使数据库空间紧张，或者由于表非常大导致相关 SQL 查询性能变慢时，可以考虑使用分区表。

使用分区表时要选择合适的拆分键(列)以及拆分策略。如果是日志类型的大表，根据时间类型的列做范围分区是最合适的。

如果是并发访问非常高的表，结合业务特点选择能满足绝大部分核心业务查询的列作为拆分键是最合适的。无论选哪个列做为分区键，都不大可能满足所有的查询性能。

分区表中的全局唯一性需求可以通过主键约束和唯一约束实现。OceanBase 的分区表的主键约束和唯一键约束必须包含拆分键。唯一约束也是一个全局索引。全局唯一的需求也可以通过本地唯一索引实现，只需要在唯一索引里包含拆分键。

示例：创建有唯一性需求的分区表

```

CREATE TABLE account(
  id bigint NOT NULL PRIMARY KEY
  , name varchar(50) NOT NULL UNIQUE
  , value number NOT NULL
  , gmt_create timestamp DEFAULT current_timestamp NOT NULL
  , gmt_modified timestamp DEFAULT current_timestamp NOT NULL
) PARTITION BY HASH(id) PARTITIONS 16;

CREATE TABLE account2(
  id bigint NOT NULL PRIMARY KEY
  , name varchar(50) NOT NULL
  , value number NOT NULL
  , gmt_create timestamp DEFAULT current_timestamp NOT NULL
  , gmt_modified timestamp DEFAULT current_timestamp NOT NULL
) PARTITION BY HASH(id) PARTITIONS 16;

CREATE UNIQUE INDEX account2_uk ON account2(name, id) LOCAL ;

show indexes from account;
show indexes from account2;

SELECT * FROM information_schema.`TABLE_CONSTRAINTS` WHERE table_schema='TPCCDB' AND table_name LIKE 'ACCOUNT%';

```

```

obclient> show indexes from account;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| account | 0 | PRIMARY | 1 | id | A | NULL | NULL | NULL | NULL | BTREE | available | | YES |
| account | 0 | | 1 | name | A | NULL | NULL | NULL | NULL | BTREE | available | | YES |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

obclient> show indexes from account2;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| account2 | 0 | PRIMARY | 1 | id | A | NULL | NULL | NULL | NULL | BTREE | available | | YES |
| account2 | 0 | account2_uk | 1 | name | A | NULL | NULL | NULL | NULL | BTREE | available | | YES |
| account2 | 0 | account2_uk | 2 | id | A | NULL | NULL | NULL | NULL | BTREE | available | | YES |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

obclient> SELECT * FROM information_schema.`TABLE_CONSTRAINTS` WHERE table_schema='TPCCDB' AND table_name LIKE 'ACCOUNT%';
+-----+-----+-----+-----+-----+-----+
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME | TABLE_SCHEMA | TABLE_NAME | CONSTRAINT_TYPE |
+-----+-----+-----+-----+-----+-----+
| def | tpccdb | PRIMARY | tpccdb | account | PRIMARY KEY |
| def | tpccdb | name | tpccdb | account | UNIQUE |
| def | tpccdb | PRIMARY | tpccdb | account2 | PRIMARY KEY |
| def | tpccdb | account2_uk | tpccdb | account2 | UNIQUE |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

```

## 4.4 创建和管理表分组

### 4.4.1 关于表分组

表分组是表的属性，表分组会影响多个表的分区在 OceanBase 机器上的分布特征。有关分区概念请参考“[关于表和分区](#)”。

不同表的分区有可能分布在不同的节点上，当两个表做表连接查询时，OceanBase 会跨节点请求数据，执行时间就跟节点间请求延时有关系。在 SQL 调优指南里，OceanBase 建议对业务上关系密切的表，设置相同的表分组。OceanBase 对于同一个表分组中的表的同号分区会管理为一个分区组。同一个分区组中的分区，OceanBase 会尽可能的分配同一个节点内部，这样就可以规避跨节点的请求。

创建表分组时，首先要规划好表分组的用途。如果是用于普通表的属性，表分组就不用分区。如果是用于分区表的属性，表分组就要指定分区策略，并且要跟分区表的分区策略保持一致。

### 示例：创建表分组

```
obclient> create tablegroup tpcc_group partition by hash partitions 6 ;
Query OK, 0 rows affected (0.03 sec)

obclient>
```

## 4.4.2 创建表时指定表分组

表分组可以在创建表的时候指定，SQL 查询格式如下：

```
CREATE TABLE table_name (
    column_name data_type [, column_name data_type]
) TABLEGROUP tablegroup_name ;
```

### 示例：创建表时指定分区组

如下面创建订单表和订单明细表，业务上这两个表经常要关联查询，所以建议放到同一个表分组中。

```
create table odr (
    o_w_id int
    , o_d_id int
    , o_id int
    , o_c_id int
    , o_carrier_id int
    , o_ol_cnt int
    , o_all_local int
    , o_entry_d date
    , index iodr(o_w_id, o_d_id, o_c_id, o_id) local
    , primary key ( o_w_id, o_d_id, o_id )
)tablegroup tpcc_group partition by hash(o_w_id) partitions 6;
create table ordl (
    ol_w_id int
    , ol_d_id int
    , ol_o_id int
    , ol_number int
    , ol_delivery_d date
    , ol_amount decimal(6, 2)
    , ol_i_id int
    , ol_supply_w_id int
    , ol_quantity int
    , ol_dist_info char(24)
    , primary key (ol_w_id, ol_d_id, ol_o_id, ol_number )
)tablegroup tpcc_group partition by hash(ol_w_id) partitions 6;
```

### 4.4.3 查看表分组信息

查看表分组信息，使用命令 `SHOW TABLEGROUPS` 和 `SHOW CREATE TABLE`。

#### 示例：查看表分组信息和定义

```
obclient> show tablegroups;
+-----+-----+-----+
| Tablegroup_name | Table_name | Database_name |
+-----+-----+-----+
| oceanbase      | NULL      | NULL          |
| tpcc_group     | cust      | tpccdb        |
| tpcc_group     | dist      | tpccdb        |
| tpcc_group     | hist      | tpccdb        |
| tpcc_group     | nord      | tpccdb        |
| tpcc_group     | ordl      | tpccdb        |
| tpcc_group     | ordl      | tpccdb        |
| tpcc_group     | stok      | tpccdb        |
| tpcc_group     | ware      | tpccdb        |
+-----+-----+-----+
9 rows in set (0.01 sec)

obclient> show create tablegroup tpcc_group\G
***** 1. row *****
      Tablegroup: tpcc_group
Create Tablegroup: CREATE TABLEGROUP IF NOT EXISTS `tpcc_group` BINDING = FALSE
partition by hash partitions 6

1 row in set (0.00 sec)
```

### 4.4.4 向表分组中增加表

如果表创建的时候没有指定表分组，也可以事后加入到表分组中。有两种方法。一是修改表的表分组属性，二是使用 `ALTER TABLEGROUP ADD` 语法。

#### 示例：修改表的表分组属性

注意：修改表的表分组属性名字不用单引号。否则，会区分大小写。

```
obclient> alter table ordl tablegroup=tpcc_group;
Query OK, 0 rows affected (0.04 sec)

obclient> alter table ordl tablegroup=tpcc_group;
Query OK, 0 rows affected (0.04 sec)
```

#### 示例：向表分组中加入表

向表分组中加入表的前提是表的分区策略跟表分组的分区策略保持一致。普通表就是不分区。

```
obclient> alter table ordl tablegroup='';
Query OK, 0 rows affected (0.04 sec)
```

```
obclient> alter table ordl tablegroup='';
Query OK, 0 rows affected (0.03 sec)

obclient> alter tablegroup tpcc_group add ordr, ordl ;
Query OK, 0 rows affected (0.03 sec)

obclient>
```

#### 4.4.5 删除表分组

如果要删除表分组，得先确认表分组中没有表。具体方法就是从表分组中删除表，或者将相关表的表分组属性清空。

```
obclient> show tablegroups;
+-----+-----+-----+
| Tablegroup_name | Table_name | Database_name |
+-----+-----+-----+
| oceanbase       | NULL      | NULL          |
| tpcc_group      | cust      | tpccdb        |
| tpcc_group      | dist      | tpccdb        |
| tpcc_group      | hist      | tpccdb        |
| tpcc_group      | nord      | tpccdb        |
| tpcc_group      | ordl      | tpccdb        |
| tpcc_group      | ordr      | tpccdb        |
| tpcc_group      | stok      | tpccdb        |
| tpcc_group      | ware      | tpccdb        |
+-----+-----+-----+
9 rows in set (0.01 sec)

obclient> drop tablegroup tpcc_group;
ERROR 4615 (HY000): tablegroup is not empty
obclient>
```

### 4.5 创建和管理视图

视图表示表的查询结果。大部分能使用表的地方，都可以使用视图。当您经常访问的数据分布在多个表里时，使用视图是最好的方法。

#### 4.5.1 创建视图

要创建视图，您可以在 obclient 命令行环境下使用 DDL 语句来创建视图，也可以在 DBeaver 图形化客户端的 SQL 编辑器里执行 DDL 语句。

本节展示如何使用 CREATE VIEW 语句创建视图 stock\_item，这个视图内容是取自于表 stock 和 item 两张表，两张表做表连接就得到这个视图。有关表连接使用，请参考“[从多个表里查询数据](#)”。

**示例：使用 CREATE VIEW 创建视图 stock\_item**

```
CREATE VIEW stock_item
AS
SELECT /*+ leading(s) use_merge(i) */
i_price, i_name, i_data, s_i_id, s_w_id, s_order_cnt, s_ytd, s_remote_cnt, s_quantity, s_data, s_dist_01, s_dist_02,
s_dist_03, s_dist_04, s_dist_05, s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10
FROM stok s, item i
WHERE s.s_i_id = i.i_id;
Query OK, 0 rows affected (0.03 sec)
```

请参考 [OceanBase SQL 参考](#) 查看更多创建视图的选项。

## 4.5.2 修改视图的查询语句

本节展示如何使用 `CREATE OR REPLACE VIEW` 语句修改视图 `stock_item`。

```
CREATE OR REPLACE VIEW stock_item
AS
SELECT /*+ leading(s) use_merge(i) */
i_price, i_name, i_data, s_i_id, s_w_id, s_order_cnt, s_ytd, s_remote_cnt, s_quantity, s_data, s_dist_01, s_dist_02,
s_dist_03, s_dist_04, s_dist_05, s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10
FROM stok s, item i
WHERE s.s_i_id = i.i_id;
Query OK, 0 rows affected (0.03 sec)
```

## 4.5.3 删除视图

删除视图使用 `DROP VIEW` 语句。删除视图并不会删除视图引用的表。如果视图被其他视图所引用，视图删除后将会导致依赖当前视图的其他视图查询失败。





## 第 5 章 向 OceanBase 迁移数据

### 5.1 关于数据迁移和同步

数据从传统数据库迁移到 OceanBase 数据库上，可以选择将数据导出为 CSV 文件、SQL 文件，然后再导入到 OceanBase 中。

### 5.2 通用数据同步框架 DataX

#### 5.2.1 DataX 简介

DataX 是阿里巴巴集团内被广泛使用的离线数据同步工具/平台，实现包括 MySQL、Oracle、SqlServer、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、DRDS 和 OceanBase 等各种异构数据源之间高效的数据同步功能。

DataX 本身作为数据同步框架，将不同数据源的同步抽象为从源头数据源读取数据的 Reader 插件，以及向目标端写入数据的 Writer 插件，理论上 DataX 框架可以支持任意数据源类型的数据同步工作。同时 DataX 插件体系作为一套生态系统，每接入一套新数据源该新加入的数据源即可实现和现有的数据源互通。

DataX 已经在 github 开源，开源地址是 [github.com/Alibaba/datax](https://github.com/Alibaba/datax)，开源的产品不支持 OceanBase 和 DB2，OceanBase 产品团队提供针对 OceanBase 和 DB2 的读写插件。

#### 5.2.2 DataX 的使用示例

DataX 安装后，默认目录在 /home/admin/datax3。目录下有个文件夹 job，默认存放数据迁移任务的配置文件，当然也可以自定义目录。

每个任务的参数文件是一个 json 格式，主要由一个 reader 和一个 writer 组成。job 文件夹下有个默认的示例任务配置文件 job.json

```
[admin /home/admin/datax3/job]
$cat job.json
{
  "job": {
    "setting": {
      "speed": {
        "byte":10485760
      },
      "errorLimit": {
```

```

        "record": 0,
        "percentage": 0.02
    }
},
"content": [
    {
        "reader": {
            "name": "streamreader",
            "parameter": {
                "column": [
                    {
                        "value": "DataX",
                        "type": "string"
                    },
                    {
                        "value": 19890604,
                        "type": "long"
                    },
                    {
                        "value": "1989-06-04 00:00:00",
                        "type": "date"
                    },
                    {
                        "value": true,
                        "type": "bool"
                    },
                    {
                        "value": "test",
                        "type": "bytes"
                    }
                ]
            },
            "sliceRecordCount": 100000
        },
        "writer": {
            "name": "streamwriter",
            "parameter": {
                "print": false,
                "encoding": "UTF-8"
            }
        }
    }
]
}
}

```

这个任务的配置文件的 reader 和 writer 类型是一个 stream。这个任务会检测 DataX 安装正确。运行之前确保安装 JDK 运行环境。

```

[admin@h07g12092.sqa.eu95 /home/admin/datax3/job]
$cd ../

[admin@h07g12092.sqa.eu95 /home/admin/datax3]
$bin/datax.py job/job.json

```

```

2020-03-24 17:34:24.719 [job-0] INFO JobContainer -
=== total summarize info ===

1. all phase average time info and max time task info:

PHASE | AVERAGE USED TIME | ALL TASK NUM | MAX USED TIME | MAX TASK ID | MAX TASK INFO
TASK_TOTAL | 0.402s | 1 | 0.402s | 0-0-0 | null
READ_TASK_INIT | 0.001s | 1 | 0.001s | 0-0-0 | null
READ_TASK_PREPARE | 0.000s | 1 | 0.000s | 0-0-0 | null
READ_TASK_DATA | 0.058s | 1 | 0.058s | 0-0-0 | null
READ_TASK_POST | 0.000s | 1 | 0.000s | 0-0-0 | null
READ_TASK_DESTROY | 0.000s | 1 | 0.000s | 0-0-0 | null
WRITE_TASK_INIT | 0.001s | 1 | 0.001s | 0-0-0 | null
WRITE_TASK_PREPARE | 0.000s | 1 | 0.000s | 0-0-0 | null
WRITE_TASK_DATA | 0.258s | 1 | 0.258s | 0-0-0 | null
WRITE_TASK_POST | 0.000s | 1 | 0.000s | 0-0-0 | null
WRITE_TASK_DESTROY | 0.000s | 1 | 0.000s | 0-0-0 | null
WAIT_READ_TIME | 0.033s | 1 | 0.033s | 0-0-0 | null
WAIT_WRITE_TIME | 0.017s | 1 | 0.017s | 0-0-0 | null

2. record average count and max count task info :

PHASE | AVERAGE RECORDS | AVERAGE BYTES | MAX RECORDS | MAX RECORD'S BYTES | MAX TASK ID | MAX TASK INFO
READ_TASK_DATA | 100000 | 2.60M | 100000 | 2.60M | 0-0-0 | null

2020-03-24 17:34:24.720 [job-0] INFO MetricReportUtil - reportJobMetric is turn off
2020-03-24 17:34:24.721 [job-0] INFO StandAloneJobContainerCommunicator - Total 100000 records, 2600000 bytes | Speed 2.48MB/s, 100000 records/s
| Error 0 records, 0 bytes | All Task WaitWriterTime 0.017s | All Task WaitReaderTime 0.033s | Percentage 100.00%
2020-03-24 17:34:24.722 [job-0] INFO LogReportUtil - report datax log is turn off
2020-03-24 17:34:24.722 [job-0] INFO JobContainer -
任务启动时刻 : 2020-03-24 17:34:23
任务结束时刻 : 2020-03-24 17:34:24
任务总计耗时 : 1s
任务平均流量 : 2.48MB/s
记录写入速度 : 100000rec/s
读出记录总数 : 100000
读写失败总数 : 0

```

注：这个默认的任务参数关闭了数据流输出，所以看不到输出结果。

## 5.3 不同数据源的 DataX 读写插件示例

DataX 官网支持绝大部分主流数据源的读写插件，并且有详细的使用文档。

### 5.3.1 CSV 文件的读写插件

csv 文件就是文本文件，用 `txtreader` 和 `txtwriter` 读写。配置文件详细语法请参见 “[DataX 官网说明](#)”。

`txtreader` 配置示例：

```

"reader":{
  "name":"txtfilereader",
  "parameter":{
    "path":["文件全路径"],
    "encoding":"UTF-8",
    "column":[
      { "index":0, "type":"long" },
      { "index":1, "type":"long" },
      { "index":2, "type":"string" },
      { "index":3, "type":"double" },
      { "index":4, "type":"string" }
    ],
    "fieldDelimiter":"||",
    "fileFormat":"text"
  }
}

```

**txtwriter 配置示例:**

```
"writer":{
  "name": "txtfilewriter",
  "parameter":{
    "path": "文件全路径",
    "fileName": "文件名",
    "writeMode": "truncate",
    "dateFormat": "yyyy-MM-dd",
    "charset": "UTF-8",
    "nullFormat": "",
    "fileDelimiter": "||"
  }
}
```

### 5.3.2 MySQL 数据库的读写插件

针对 MySQL 数据库，用 mysqlreader 和 mysqlwriter 插件读写。

**mysqlreader 配置示例:**

```
"reader": {
  "name": "mysqlreader",
  "parameter": {
    "username": "root",
    "password": "root",
    "column": [
      "id",
      "name"
    ],
    "splitPk": "db_id",
    "connection": [
      {
        "table": [
          "table"
        ],
        "jdbcUrl": [
          "jdbc:mysql://127.0.0.1:3306/database"
        ]
      }
    ]
  }
}
```

**mysqlwriter 配置示例:**

```
"writer": {
  "name": "mysqlwriter",
  "parameter": {
    "writeMode": "insert",
    "username": "root",
    "password": "root",
    "column": [
      "id",
      "name"
    ],
    "session": [
```

```

        "set session sql_mode='ANSI'"
    ],
    "preSql": [
        "delete from test"
    ],
    "connection": [
        {
            "jdbcUrl":
"jdbc:mysql://127.0.0.1:3306/datax?useUnicode=true&characterEncoding=gbk",
            "table": [
                "test"
            ]
        }
    ]
}

```

### 5.3.3 ORACLE 数据库的读写插件

针对 ORACLE 数据库，用 oraclereader 和 oraclewriter 插件来读写。

**oraclereader 配置示例：**

```

"reader": {
    "name": "oraclereader",
    "parameter": {
        // 数据库连接用户名
        "username": "root",
        // 数据库连接密码
        "password": "root",
        "column": [
            "id", "name"
        ],
        // 切分主键
        "splitPk": "db_id",
        "connection": [
            {
                "table": [
                    "table"
                ],
                "jdbcUrl": [
"jdbc:oracle:thin:@[HOST_NAME]:PORT:[DATABASE_NAME]"
                ]
            }
        ]
    }
}

```

**oraclewriter 配置示例：**

```

"writer": {
    "name": "oraclewriter",
    "parameter": {
        "username": "root",
        "password": "root",
        "column": [

```

```

        "id",
        "name"
    ],
    "preSql": [
        "delete from test"
    ],
    "connection": [
        {
            "jdbcUrl": "jdbc:oracle:thin:@[HOST_NAME]:PORT:[DATABASE_NAME]",
            "table": [
                "test"
            ]
        }
    ]
}

```

### 5.3.4 DB2 数据库的读写插件

**dbreader 配置示例:**

```

"reader":{
    "name":"db2reader",
    "parameter":{
        "username":"SRC_DB_USERNAME",
        "password":"SRC_DB_PASSWORD",
        "column":[
            "SRC_COLUMN_LIST"
        ],
        "connection":[
            {
                "table":[
                    "SRC_TABLE_NAME"
                ],
                "jdbcUrl":[
                    "jdbc:db2://SRC_DB_IP:SRC_DB_PORT/SRC_DB_NAME"
                ]
            }
        ]
    }
}

```

**db2writer 配置示例:**

### 5.3.5 OceanBase 数据库的读写插件

OceanBase 数据库使用插件 `oceanbasev10reader` 和 `oceanbasev10writer` 来读写。该插件由 OceanBase 产品团队单独提供。

## ■ oceanbasev10reader 配置示例

```

    "reader":{
        "name":"oceanbasev10reader",
        "parameter":{
            "where":"",
            "timeout":10000,
            "readBatchSize":100000,
            "readByPartition":"true",
            "column": [
                "列名 1", "列名 2"
            ],
            "connection":[
                {
                    "jdbcUrl":["||_dsc_ob10_dsc_||集群名:租户名
||_dsc_ob10_dsc_||jdbc:oceanbase://连接 IP:连接端口/模式名或数据库名"],
                    "table":["表名"]
                }
            ],
            "username":"租户内用户名",
            "password":"密码"
        }
    }
}

```

### 示例 9-3-5-1：OceanBase 表 ware 导出到 csv 文件。

```

[admin@*** /home/admin/datax3]
$cat job/ob_tpcc_ware_2_csv.json
{
    "job":{
        "setting":{
            "speed":{
                "channel":10
            },
            "errorLimit":{
                "record":0, "percentage": 0.02
            }
        },
        "content":[
            {
                "reader":{
                    "name":"oceanbasev10reader",
                    "parameter":{
                        "where":"",
                        "timeout":10000,
                        "readBatchSize":100000,
                        "readByPartition":"true",
                        "column": [
"W_ID", "W_YTD", "W_TAX", "W_NAME", "W_STREET_1", "W_STREET_2", "W_CITY", "W_STATE", "W_ZIP"
                        ],
                        "connection":[
                            {
"WjdbcUrl":["||_dsc_ob10_dsc_||obdemo:obmysql||_dsc_ob10_dsc_||jdbc:oceanbase://127.1:2883/tpcc"],
                                "table":["ware"]
                            }
                        ],
                        "username":"tpcc",
                        "password":"123456"
                    }
                },
                "writer":{
                    "name":"txtfilewriter",
                    "parameter":{
                        "path":"/home/admin/csvdata/",

```



```

        "fileName": "ware",
        "writeMode": "truncate",
        "dateFormat": "yyyy-MM-dd",
        "charset": "UTF-8",
        "nullFormat": "",
        "fileDelimiter": "||"
    }
}
]
}
}

[admin@*** /home/admin/datax3]
$bin/datax.py job/ob_tpcc_ware_2_csv.json

```

```

=== total summarize info ===

1. all phase average time info and max time task info:

PHASE | AVERAGE USED TIME | ALL TASK NUM | MAX USED TIME | MAX TASK ID | MAX TASK INFO
TASK_TOTAL | 0.202s | 1 | 0.202s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
READ_TASK_INIT | 0.008s | 1 | 0.008s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
READ_TASK_PREPARE | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
READ_TASK_DATA | 0.058s | 1 | 0.058s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
READ_TASK_POST | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
READ_TASK_DESTROY | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
WRITE_TASK_INIT | 0.001s | 1 | 0.001s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
WRITE_TASK_PREPARE | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
WRITE_TASK_DATA | 0.066s | 1 | 0.066s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
WRITE_TASK_POST | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
WRITE_TASK_DESTROY | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
SQL_QUERY | 0.010s | 1 | 0.010s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
RESULT_NEXT_ALL | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
WAIT_READ_TIME | 0.062s | 1 | 0.062s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc
WAIT_WRITE_TIME | 0.000s | 1 | 0.000s | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc

2. record average count and max count task info:

PHASE | AVERAGE RECORDS | AVERAGE BYTES | MAX RECORDS | MAX RECORD'S BYTES | MAX TASK ID | MAX TASK INFO
READ_TASK_DATA | 2 | 147B | 2 | 147B | 0-0-0 | ware,jdbc:oceanbase://127.1:2883/tpcc

2020-03-25 17:33:41.286 [job-0] INFO MetricReportUtil - reportJobMetric is turn off
2020-03-25 17:33:41.287 [job-0] INFO StandAloneJobContainerCommunicator - Total 2 records, 147 bytes | Speed 147B/s, 2 records/s | Error 0 records, 0 bytes | All Task WaitWriterTime 0.000s | All Task WaitReaderTime 0.062s | Percentage 100.00%
2020-03-25 17:33:41.288 [job-0] INFO LogReportUtil - report datax log is turn off
2020-03-25 17:33:41.288 [job-0] INFO JobContainer -
任务启动时刻 : 2020-03-25 17:33:39
任务结束时刻 : 2020-03-25 17:33:41
任务总耗时 : 1s
任务平均流量 : 147B/s
记录写入速度 : 2rec/s
读出记录总数 : 2
读写失败总数 : 0

```

## ■ oceanbasev10writer 配置示例

使用 DataX 向 OceanBase 里写入时，要避免写入速度过快导致 OceanBase 的增量内存耗尽。通常建议 DataX 配置文件里针对写入做一个写入限速设置。关键字是：

```

"writer": {
  "name": "oceanbasev10writer",
  "parameter": {
    "username": "租户内的用户名",
    "password": "密码",
    "writeMode": "insert",
    "column": [
      "列名 1", "列名 2"
    ],
    "preSql": [
      ""
    ],
    "connection": [
      {
        "jdbcUrl": "||_dsc_ob10_dsc_||集群名:租户名||_dsc_ob10_dsc_||jdbc:oceanbase://
连接 IP:连接端口(默认 2883)/模式名或数据库名",
        "table": [
          "表名"
        ]
      }
    ]
  }
}

```

```

    ]
  },
  "batchSize": 1024,
  "memstoreThreshold": "90"
}
}

```

### 示例 9-3-5-2：从 csv 文件导入到 OceanBase 表中。

```

[admin@*** /home/admin/datax3]
$cat job/csv_2_ob_tpcc_ware2.json
{
  "job":{
    "setting":{
      "speed":{
        "channel":32
      },
      "errorLimit":{
        "record":0, "percentage": 0.02
      }
    },
    "content":[
      {
        "reader":{
          "name":"txtfilereader",
          "parameter":{
            "path":["/home/admin/csvdata/ware*"],
            "encoding":"UTF-8",
            "column":[
              { "index":0, "type":"long" },
              { "index":1, "type":"long" },
              { "index":2, "type":"long" },
              { "index":3, "type":"string" },
              { "index":4, "type":"string" },
              { "index":5, "type":"string" },
              { "index":6, "type":"string" },
              { "index":7, "type":"string" },
              { "index":8, "type":"string" }
            ],
            "fieldDelimiter":",",
            "fileFormat":"text"
          }
        },
        "writer":{
          "name":"oceanbasev10writer",
          "parameter":{
            "writeMode":"insert",
            "column":[
              "W_ID","W_YTD","W_TAX","W_NAME","W_STREET_1","W_STREET_2","W_CITY","W_STATE","W_ZIP"
            ],
            "connection":[
              {
                "jdbcUrl":["||_dsc_ob10_dsc_||obdemo:obmysql||_dsc_ob10_dsc_||jdbc:oceanbase://127.1:2883/tpcc",
                  "table":["WARE2"]
                ]
              },
              {
                "username":"tpcc",
                "password":"123456",
                "batchSize":256,
                "memstoreThreshold":"90"
              }
            ]
          }
        }
      }
    ]
  }
}

```

```
[admin@*** /home/admin/datax3]
$bin/datax.py job/csv_2_ob_tpcc_ware2.json
```

```
1. all phase average time info and max time task info:
```

| PHASE              | AVERAGE USED TIME | ALL TASK NUM | MAX USED TIME | MAX TASK ID | MAX TASK INFO |
|--------------------|-------------------|--------------|---------------|-------------|---------------|
| TASK_TOTAL         | 0.202s            | 1            | 0.202s        | 0-0-0       | null          |
| READ_TASK_INIT     | 0.001s            | 1            | 0.001s        | 0-0-0       | null          |
| READ_TASK_PREPARE  | 0.000s            | 1            | 0.000s        | 0-0-0       | null          |
| READ_TASK_DATA     | 0.013s            | 1            | 0.013s        | 0-0-0       | null          |
| READ_TASK_POST     | 0.000s            | 1            | 0.000s        | 0-0-0       | null          |
| READ_TASK_DESTROY  | 0.000s            | 1            | 0.000s        | 0-0-0       | null          |
| WRITE_TASK_INIT    | 0.019s            | 1            | 0.019s        | 0-0-0       | null          |
| WRITE_TASK_PREPARE | 0.008s            | 1            | 0.008s        | 0-0-0       | null          |
| WRITE_TASK_DATA    | 0.078s            | 1            | 0.078s        | 0-0-0       | null          |
| WRITE_TASK_POST    | 0.000s            | 1            | 0.000s        | 0-0-0       | null          |
| WRITE_TASK_DESTROY | 0.001s            | 1            | 0.001s        | 0-0-0       | null          |
| WAIT_READ_TIME     | 0.000s            | 1            | 0.000s        | 0-0-0       | null          |
| WAIT_WRITE_TIME    | 0.000s            | 1            | 0.000s        | 0-0-0       | null          |

```
2. record average count and max count task info :
```

| PHASE          | AVERAGE RECORDS | AVERAGE BYTES | MAX RECORDS | MAX RECORD'S BYTES | MAX TASK ID | MAX TASK INFO |
|----------------|-----------------|---------------|-------------|--------------------|-------------|---------------|
| READ_TASK_DATA | 2               | 147B          | 2           | 147B               | 0-0-0       | null          |

```
2020-03-25 17:35:39.328 [job-0] INFO MetricReportUtil - reportJobMetric is turn off
2020-03-25 17:35:39.329 [job-0] INFO StandAloneJobContainerCommunicator - Total 2 records, 147 bytes | Speed 147B/s, 2 records/s | Error 0 records, 0 bytes | All Task
2020-03-25 17:35:39.330 [job-0] INFO LogReportUtil - report datax log is turn off
2020-03-25 17:35:39.330 [job-0] INFO JobContainer -
任务启动时刻      : 2020-03-25 17:35:37
任务结束时刻      : 2020-03-25 17:35:39
任务总计耗时      : 1s
任务平均速度      : 147B/s
记录写入速度      : 2rec/s
读出记录总数      : 2
读写失败总数      : 0
```

```
obclient> truncate table ware2;
Query OK, 0 rows affected (0.04 sec)
```

```
obclient> select * from ware2;
```

| W_ID | W_YTD | W_TAX | W_NAME     | W_STREET_1          | W_STREET_2          | W_CITY       | W_STATE | W_ZIP     |
|------|-------|-------|------------|---------------------|---------------------|--------------|---------|-----------|
| 2    | 1200  | 0     | L6xwRsbDk  | xEdT1jkEntbLwoI1Zb0 | NT0j4RCQ40qrS       | vlwzndw2FPr0 | XR      | 063311111 |
| 1    | 1200  | 0     | n1P4zYo8OH | jTNk0XwX0dh         | lf9QXTXGoF04IZBkCP7 | srRq15uvxe5  | GQ      | 506811111 |

```
2 rows in set (0.00 sec)
```

## 5.4 OceanBase 数据加载技术

OceanBase 支持命令 `LOAD DATA` 用于加载外部文本文件的内容到数据库表中。

`LOAD DATA` 语法格式如下：

```
LOAD DATA
  [/*+ parallel(N)*/]
  INFILE 'file_name'
  [REPLACE | IGNORE]
  INTO TABLE tbl_name
  [{FIELDS | COLUMNS}
    [TERMINATED BY 'string']
    [[OPTIONALLY] ENCLOSED BY 'char']
    [ESCAPED BY 'char']
  ]
  [LINES
    [STARTING BY 'string']
    [TERMINATED BY 'string']
  ]
  [IGNORE number {LINES | ROWS}]
  [(col_name_or_user_var
    [, col_name_or_user_var] ...)]
  [SET col_name={expr | DEFAULT},
    [, col_name={expr | DEFAULT}] ...]
```

其中 REPLACE 选项只适用于 MySQL 租户。

### 注意:

要加载的文件必须在该表的主副本所在的 OBServer 上, 当前版本不支持从远程客户端加载数据。

### 示例: 通过 load data 导入 csv 文件到表中

```
[admin@h07g12092.sqa.eu95 /home/admin/csvdata]
$more ware__df8f30ac_64e0_474c_9cc4_9919d64c5e4c
2,1200,.0862,L6xwRsbDk,xEdT1jkENtLwoI1Zb0,NT0j4RCQ40qrS,vlwzndw2FPPrO,XR,063311111
1,1200,.1868,n1P4zYo8OH,jTNkXKWX0dh,1f9QXTXXGoF04IZBkCP7,srRq15uvxe5,GQ,506811111

$obclient -h192.168.1.101 -utpcc@obmysql -P2881 -p123456

obclient> load data infile '/home/admin/csvdata/ware__df8f30ac_64e0_474c_9cc4_9919d64c5e4c' into table ware2
fields terminated by ',' lines terminated by '\n';
Query OK, 2 rows affected (0.02 sec)
Records: 2 Deleted: 0 Skipped: 0 Warnings: 0

obclient> select * from ware2\G
***** 1. row *****
      W_ID: 2
      W_YTD: 1200
      W_TAX: .0862
      W_NAME: L6xwRsbDk
W_STREET_1: xEdT1jkENtLwoI1Zb0
W_STREET_2: NT0j4RCQ40qrS
      W_CITY: vlwzndw2FPPrO
      W_STATE: XR
      W_ZIP: 063311111
***** 2. row *****
      W_ID: 1
      W_YTD: 1200
      W_TAX: .1868
      W_NAME: n1P4zYo8OH
W_STREET_1: jTNkXKWX0dh
W_STREET_2: 1f9QXTXXGoF04IZBkCP7
      W_CITY: srRq15uvxe5
      W_STATE: GQ
      W_ZIP: 506811111
2 rows in set (0.00 sec)
obclient>
```

## 附录

### Java 连接 OceanBase 示例

#### 示例：OceanBase Java 连接示例

##### 1. 添加 Maven 依赖

```
<dependency>
  <groupId>com.alipay.oceanbase</groupId>
  <artifactId>oceanbase-client</artifactId>
  <version>1.0.9</version>
</dependency>
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <!-- 推荐版本 18, 经过测试 14~18 都可以 -->
  <version>18.0</version>
</dependency>
```

##### 2. 修改连接字符串

连接串的前缀需要设置为 jdbc:oceanbase，其他部分的使用方式与原生的 MySQL 使用方式保持一致。

```
String url = "jdbc:oceanbase://192.168.1.101/TPCC?useUnicode=true&characterEncoding=utf-8";
String username = "TPCC@obmysql#obdemo";
String password = "123456";
Connection conn = null;
try {
    Class.forName("com.alipay.oceanbase.obproxy.mysql.jdbc.Driver");
    conn = DriverManager.getConnection(url, username, password);
    PreparedStatement ps = conn.prepareStatement("select to_char(sysdate,'yyyy-MM-dd HH24:mi:ss') from dual;");
    ResultSet rs = ps.executeQuery();
    rs.next();
    System.out.println("sysdate is:" + rs.getString(1));
    rs.close();
    ps.close();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    if (null != conn) {
        conn.close();
    }
}
```

#### 注意事项：

1. 目前驱动与服务端交互使用的是文本写协议部分，驱动在 PreparedStatement 中 setTimestamp 类型中都会在前面增加 timestamp 的字面量，因此针对 PreparedStatement 模式下，timestamp 参数不支持字面量

2. ServerPreparedStatement 支持还不完善，因此请不要设置 useServerPrepStmts 和 cachePrepStmts 参数
3. 对于 Druid 框架，如果没有使用 DriverManager，需要直接指定 DrvierClass，如下：

```
<property name="driverClassName" value="com.alipay.oceanbase.obproxy.mysql.jdbc.Driver">
```

## OceanBase 常用参数变量

以下是常用的跟开发有关的参数( parameters )

| 参数名                                   | 参数值   | 参数含义                                    | 范围 |
|---------------------------------------|-------|---|----|
| minor_freeze_times                    | 0     | 指定多少次minor freeze后会触发major freeze。      | 集群 |
| enable_major_freeze                   | True  | 指定集群是否开启自动major freeze。                 | 集群 |
| major_freeze_duty_time                | 02:00 | 指定集群自动 major freeze的开始时间。               | 集群 |
| enable_rebalance                      | True  | 指定是否开启分区负载均衡。True开启，False关闭。            | 集群 |
| enable_auto_leader_switch             | True  | 指定是否开启分区自动切主。                           | 集群 |
| large_query_threshold                 | 100ms | 指定一个查询执行时间被判定为大查询的阈值。                   | 集群 |
| large_query_worker_percentage         | 30    | 指定大查询要使用的CPU工作线程比例。                     | 集群 |
| data_copy_concurrency                 | 20    | 指定分区迁移时集群最大的迁移任务数。                      | 集群 |
| server_data_copy_out_concurrency      | 2     | 指定分区迁移时每个节点最大的迁出任务数。                    | 集群 |
| server_data_copy_in_concurrency       | 2     | 指定分区迁移时每个节点最大的迁入任务数。                    | 集群 |
| freeze_trigger_percentage             | 70    | 指定增量内存使用比例的一个阈值，达到这个值会触发 minor freeze 。 | 集群 |
| enable_perf_event                     | True  | 指定是否开启性能事件信息收集，默认是False。                | 集群 |
| enable_sql_audit                      | True  | 指定是否开启SQL日志，默认是True。                    | 集群 |
| sql_work_area                         | 1G    | 指定租户的SQL工作内存。                           | 租户 |
| writing_throttling_maximum_duration   | 1h    | 指定触发租户限流时增量内存预估最大使用时间。                  | 租户 |
| writing_throttling_trigger_percentage | 100   | 指定触发租户限流时增量内存的使用比例阈值。                   | 租户 |
| max_stale_time_for_weak_consistency   | 5s    | 指定弱一致读能接受备副本的最大时延。                      | 租户 |

以下是常用的租户变量( variables )

| 变量名                         | 变量值   | 变量含义  |
|-----------------------------|---|---|
| auto_increment_increment    | 1   | 指定 MySQL 租户的自增列单次自增大小。  |
| auto_increment_offset       | 1   | 指定 MySQL 租户的自增列自增起始值。   |
| auto_increment_cache_size   | 1000000   | 指定 MySQL 租户的自增列内部缓存大小。  |
| autocommit                  | ON  | 指定租户是否开启事务自动提交。   |
| ob_compatibility_mode       | ORACLE/MYSQL  | 显示当前租户的兼容类型。<br>ORACLE/MYSQL。只读变量。  |
| ob_tcp_invited_nodes        | %   | 指定租户访问的 IP 白名单,逗号分隔。如: 127.1,192.168.0.0/16                               |
| ob_timestamp_service        | GTS   | 指定租户时间服务是用 GTS 还是 LTS。  |
| ob_query_timeout            | 10000000  | 指定 SQL 执行默认超时时间,单位微秒(us)。   |
| ob_trx_idle_timeout         | 120000000   | 指定租户里事务最大空闲时间,单位微秒(us)。   |
| ob_trx_timeout              | 100000000   | 指定租户里事务最大持续时间,单位微秒(us)。   |
| ob_read_consistency         | STRONG  | 指定租户里读 SQL 的默认一致性级别。STRONG 是强一致读, WEAK 是弱一致性读, FORZEN 是读上次 major freeze 的 |
| ob_sql_audit_percentage     | 3   | 指定租户 SQL 日志占用最大内存百分比。   |
| ob_sql_work_area_percentage | 5   | 指定租户 SQL 工作内存占内存最大百分比。  |
| recyclebin                  | ON  | 指定是否开启回收站。ON 开启, OFF 关闭。  |
| sql_mode                    | STRICT_TRANS_TABLES,STRICT_ALL_TABLES,PAD_CHAR_TO_FULL_LENGTH | 指定 SQL 遵守的模式。   |
| tx_isolation                | READ-COMMITTED  | 指定租户事务默认隔离级别。   |
| time_zone                   | +8:00   | 指定租户默认时区。   |
| system_time_zone            | +08:00  | 显示系统默认时区。   |
| version_comment             |   | 显示 OceanBase 版本, 只读变量。  |

## OceanBase 常用 SQL Hints

SQL Hint 的使用语法:

```
/*+ HINT_NAME */
```

或

```
/*+ HINT_NAME ( HINT_PARA )
```

如果是在命令行 obclient 或 mysql 下连接 OceanBase，注意需要指定参数 -c，这样 Hint 文本才会发送到 OBServer 端生效。

| Hint 名称                 | Hint 参数                             | Hint 语义   |
|-------------------------|-------------------------------------|---|
| NO_REWRITE              |                                     | 不改写SQL。   |
| READ_CONSISTENCY        | weak strong frozen                  | weak:弱一致性读<br>strong:强一致性读<br>frozen:读最近一次冻结点的数据                    |
| INDEX_HINT              | [qb_name] table_name index_name     | 指定查询表时选择的索引。  |
| QUERY_TIMEOUT           | int64                               | 指定语句执行的超时时间, 单位是微秒(us)。   |
| LEADING                 | [qb_name] table_name [, table_name] | 指定多表连接时的顺序。   |
| ORDERED                 |                                     | 指定多表连接顺序按SQL中表出现的顺序。  |
| FULL                    | [qb_name] table_name                | 指定表的访问方式为全表扫描(有主键时会读主键)。  |
| USE_MERGE               | [qb_name] table_name [, table_name] | 指定多表连接时使用MERGE算法。   |
| USE_NL                  | [qb_name] table_name [, table_name] | 指定多表连接时使用NEST LOOP算法。   |
| USE_BNL                 | [qb_name] table_name [, table_name] | 指定多表连接时适用BLOCK NEST LOOP算法。   |
| USE_HASH_AGGREGATION    | [qb_name]                           | 指定 aggregate 方法使用HASH AGGREGATE, 例如HASH GROUP BY, HASH DISTINCT。    |
| NO_USE_HASH_AGGREGATION | [qb_name]                           | 指定 aggregate 方法不使用HASH AGGREGATE, 使用MERGE GROUP BY, MERGE DISTINCT。 |
| QB_NAME                 | qb_name                             | 指定 query block的名称。  |
| PARALLEL                | int64                               | 指定分布式执行的并行度。  |
|                         |                                     |   |