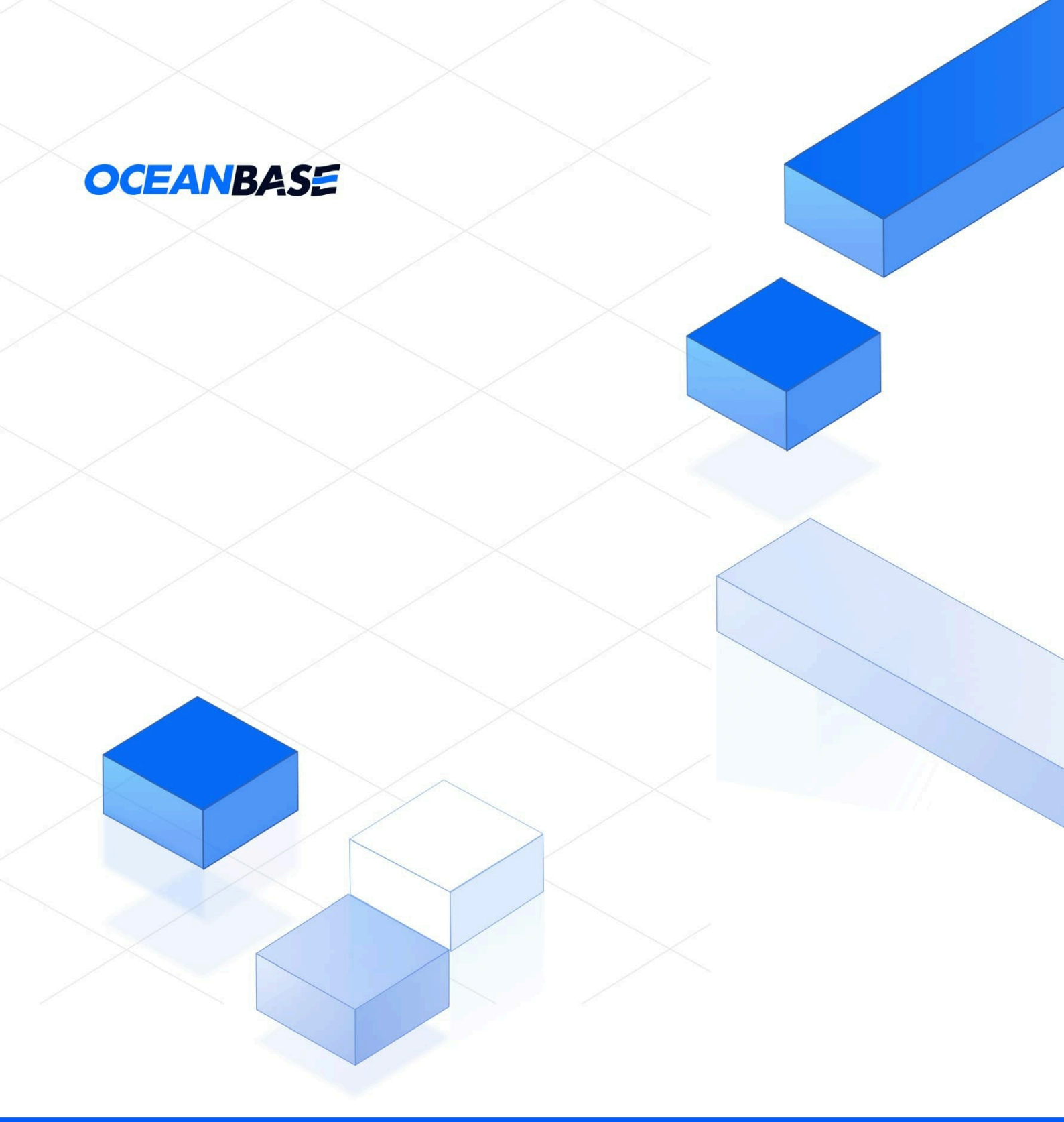

OceanBase 数据库

参考指南--数据库设计规范和约束

| 产品版本：V4.5.0 | 文档版本：20251219

声明

北京奥星贝斯科技有限公司版权所有©2024，并保留一切权利。

商标声明

免责声明

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	**危险** 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	**警告** 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	**注意** 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	**说明** 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1 对象命名规范综述

本文简介 OceanBase 数据库对象命名的通用规范。

在进行数据库开发时，对数据库对象应该有统一的命名方式，同时这种方式应该标准化，使代码的可读性更强，便于其他人阅读、理解和继承。本文旨在给开发者提供一定的命名规范建议。

1.1 通用规范

在数据库中对象的命名有一些通用的规范和注意事项，具体如下所述。

- 统一要求各应用数据库和表的字符集必须一致，而且所有表的字符集都要一致，推荐全部使用 utf8mb4，且在表级不建议自定义字符编码。
- 推荐使用统一的命名规范，并使命名标准化。
- 推荐使用英文名词全称。
- 推荐使用标准通用缩写。
- 不推荐使用中文标识。
- 不推荐混用拼音与英文。
- 不推荐全部使用数字、无实际意义字母、下划线或特殊字符组成无实际意义的一串字符。
- 不推荐在对象名的字符间留空格。
- 不推荐使用系统保留词、关键字。
- 不推荐与数据库系统或常用访问方法冲突。
- 不推荐全部使用拼音。

1.2 标识符长度限制

- MySQL 模式。

数据项	最大长度
集群名	128 字节。
租户名	64 字节。
用户名	64 字节。
数据库名	128 字节。
表名	64 字符。
列名	128 字节。
索引名	64 字节。
视图名	64 字节。
别名	255 字节。
分区名	64 字符。

- Oracle 模式。

类型	最大长度
集群名	128 字节。
租户名	64 字节。
用户名	64 字节。
表名	128 字节。
列名	128 字节。
索引名	128 字节。
视图名	128 字节。
别名	255 字节。
对象名	128 字节。
分区名	64 字符。

1.3 ODP 最大连接数限制

类型	最大限制
单个 ODP 的连接数	由 ODP 的 client_max_connections 参数控制，默认为 8192。 **说明** 您可以通过增加 ODP 节点数或者修改 client_max_connections 配置项值的方式来增大集群的连接总数限制。

1.4 分区副本数限制

类型	最大限制
每个 OBServer 节点的分区副本数	无严格限制。 **说明** 每个 OBServer 节点的分区副本数可根据租户内存大小来预估，1G 内存约支持约 2 万 tablet。

1.5 单个表的限制

类型	最大限制
行长度	1.5M 字节。
列数	4096 列。
索引个数	128 个。
索引总列数	512 列。
索引长度	1.5M 字节。
主键总列数	64 列。
主键长度	16K。
分区个数	• Oracle 模式：65536 个。 • MySQL 模式：8192 个。

1.6 单列的限制

类型	最大限制
索引单个列长度	262143 字节。

1.7 字符串类型限制

- MySQL 模式。

类型	最大长度
CHAR	256 字节。
VARCHAR	1048576 字节。
BINARY	256 字节。
VARBINARY	1048576 字节。
BLOB	536870911 字节。
TEXT	536870911 字节。

- Oracle 模式。

类型	最大长度
CHAR	2000 字节。
VARCHAR	32767 字节。
VARCHAR2	32767 字节。
NCHAR	2000 字节。
NVARCHAR2	32767 字节。

2 租户命名规范

租户是 OceanBase 独有的对象，从架构上，租户类似实例。本文旨在帮助开发者规范租户的命名规范。

2.1 租户命名建议

- 租户名称必须控制在 64 字节以内，一律采用小写格式。
- 租户名称推荐以业务线的名称或缩写，示例：deposit 业务线租户名称是 "deposit"。
- 若某个业务系统的子系统独立成租户，推荐以业务系统名称或缩写，通过字母、数字和下划线来命名，示例：deposit 业务线的 account 子系统的租户名称是 "deposit_account"。

3 用户命名规范

在进行数据库使用过程中，开发者会创建各种用户，通过这些用户来进行数据库操作，在进行创建用户的时候，对用户命名希望遵循一定的规范，本文旨在帮助开发者规范数据库中用户的命名。本文的命名规则皆为建议，供开发者进行参考。

本文介绍的用户命名规范以用户功能为区分，不同的业务可以根据不同的场景进行区分，本文仅为一种示范。数据库中的非系统用户，大致可以按照功能，分为如下几类：业务对象属主用户、应用访问数据库用户、工具类用户、监控类用户、数据备份用户、数据库管理用户、测试通用用户等。针对每种功能的用户，命名规范分别如下。

- **业务对象属主用户**

  业务对象属主用户负责数据库中的对象的创建和维护等工作。对于这类用户，命名规则为"子系统简称 +data"。示例：子系统简称为 abs，该用户命名为 "absdata"。

- **应用访问数据库用户**

  应用访问数据库用户主要是负责连接数据库对业务数据进行访问和操作。对于这类用户，命名规则为 "子系统简称 +opr"。示例：子系统简称为 abs，该用户命名为 "absopr"。

- **工具类用户**

  数据库中很多的用户是工具类用户，比如用于数据迁移的用户，可以考虑根据迁移的工具类型进行命名。示例如下：

  - kettle 用户，命名规则为 "子系统简称 +ktl"，示例：子系统简称为 abs，该用户命名为 "absktl"。
  - sqoop 用户，命名规则为"子系统简称 +sqp"，示例：子系统简称为 abs，该用户命名为 "abssqp"。

- **监控类用户**

  数据库中可创建单独的监控用户。命名为 dbmonopr。

- **数据备份用户**

  数据库中专门用于备份的用户。命名规则为"子系统简称 +bak"。示例：子系统简称为 abs，该用户命名为 "absbak"。

- **数据库管理用户**

  数据库管理用户为 DBA 用户，用于对数据库进行管理。命名为 "dbmgr"。为区分正式数据库与测试数据库，可以命名测试数据库中的 DBA 用户为 "testmgr"。

- **测试通用用户**

  在测试库中进行数据库操作的，权限较高供测试人员使用的用户。命名规则为"子系统简称 +test"。示例为：子系统简称为 abs，该用户命名为 "abstest"。

4 表命名规范

在进行数据库使用过程中，开发者会创建各种表，来进行数据库操作，本文旨在帮助开发者规范数据库中表的命名。

4.1 表命名

- 表名规范 MySQL 模式和 Oracle 模式如下：
  - OceanBase 数据库 MySQL 模式一般性规范
    - 由小写字母、数字、下划线组成，以小写字母开头，以小写字母或数字结尾。
    - 长度大于等于 3（研发规范）小于数据库表名最大限制字符。
    - 不以 OceanBase 数据库 MySQL 模式中存在的关键字、保留字命名。

4.1.0.1 注意

OceanBase 数据库 MySQL 模式当前表名最大长度限制为 64，具体版本可能有不同限制。

  - OceanBase 数据库 Oracle 模式一般性规范
    - 由大写字母、数字、下划线组成，以大写字母开头，以大写字母或数字结尾。
    - 长度大于等于 3（研发规范）小于数据库表名最大限制字符。
    - 不以 OceanBase 数据库 Oracle 模式中存在的关键字、保留字命名。

4.1.0.2 注意

    - OceanBase 数据库 Oracle 模式规范除字母大小写要求之外，其余与 OceanBase 数据库 MySQL 模式规范相同。
    - OceanBase 数据库 Oracle 模式当前表名最大长度限制为 64，具体版本可能有不同限制。

- 表名推荐使用字母开头。
- 表名推荐望文生义。示例："TEST"。
- 表名不推荐使用下划线开头，不推荐使用下划线结束。
- 表名不推荐使用数字开头。
- 表名不使用系统保留字和关键字。
- 表名不推荐下划线中间只出现数字。
- 表名不推荐使用复数名词。

- 表名推荐使用子系统名称或标准通用缩写开头，后跟功能名称或者标准通用缩写，以 "_" 分隔，最好是 "业务名称_表的作用"，示例："ACCOUNT_USER"。
- 表名后跟数字标号推荐有序递增，编号从 "00" 开始递增。示例："ACCOUNT_USER_00"，"ACCOUNT_USER_01"，"ACCOUNT_USER_02"。
- 以时间进行分表的名称推荐格式是 "表通配名_时间"，时间建议为表中数据的时间四-六位数字缩写，示例："ACCOUNT_USER" 表在 2022 年 1 月的分区表命名为 "ACCOUNT_USER_2201"。
- 中间表用于保留中间结果集，命名规则推荐为："tmp_表名(或缩写)列名（或缩写）创建时间"，示例："tmp_account_tbluser_20220224"。
- 备份表用于备份或抓取源表快照，命名规则推荐为："bak_表名(或缩写)列名（或缩写）创建时间"，示例："bak_ account_tbluser _20220224"。

5 字段命名规范

在数据中建表时，对字段进行命名应该遵循相关的规范，本文旨在帮助开发者规范数据库中字段的命名。

5.1 字段命名

- 字段名推荐英文字母统一采用大写或小写，不混用大小写。
- 字段名推荐使用字母开头。
- 字段名推荐望文生义。示例："NAME"。
- 字段名不推荐使用下划线开头，不推荐使用下划线结束。
- 字段名不推荐使用数字开头。
- 字段名不使用系统保留字和关键字。
- 字段名不推荐下划线中间只出现数字。
- 字段名不推荐使用复数名词。
- 字段名推荐使用字段代表的功能名称或者通用标准缩写，以 "_" 分隔，最好是 "业务名称_字段的作用"，示例："TASK_NUM"、"CREATE_DATE"、"STATION_DESC"、"TASK_ID"。

6 其他命名规范

本文将对 OceanBase 数据库中其他对象的命名方式提供规范性建议，旨在帮助开发者规范相关数据库对象的命名。

6.1 索引命名

这里介绍普通索引、唯一索引、主键和外键的推荐命名规范。

- 普通索引

  普通索引的命名推荐为："idx+_+表名(或缩写)+列名（或缩写）"，示例："idx_dept_deptno"。

- 唯一性约束

  UNIQUE 索引，命名推荐为："uk+_+表名(或缩写)+列名（或缩写）"，示例："uk_dept_deptno"。

- 主键约束

  表主键命名推荐为："pk+*+表名(或缩写)+*+主键字段"，示例："pk_dept_deptno"。

- 外键约束

  表外键命名推荐为："fk+_+表名(或缩写)+*主表名(或缩写)+*+主键字段"，示例："fk_teacher_student_id"。

6.2 视图命名

视图名的命名推荐为："v+_+表名(或缩写)+列名（或缩写）"，示例："v_dept_deptno"。

6.3 同义词命名

同义词的命名推荐为："syn+_+表名(或缩写)"，示例："syn_temp"。

6.4 触发器命名

触发器根据功能分为 AFTER 型触发器、BEFORE 型触发器、INSTEAD OF 型触发器三种触发器，这里推荐根据功能进行命名。

- AFTER 型触发器。

  命名推荐为："tr_+表名(或缩写)+_+aft+列名（或缩写）"，示例："tr_dept_aft_deptno"。

- BEFORE 型触发器。

  命名推荐为："tr_+表名(或缩写)*+bef+*列名（或缩写）"，示例："tr_dept_bef_deptno"。

- INSTEAD OF 型触发器。

  命名推荐为："tr_+表名(或缩写)+_+ins+列名（或缩写）"，示例："tr_dept_ins_deptno"。

6.5 存储过程命名

存储过程的命名推荐为："proc+_+表名(或缩写)" 或 "proc+_+功能名称(或缩写)"，示例："proc_temp" 或 "proc_out"。

6.6 函数命名

函数的命名推荐为："功能名称(或缩写)"，示例："GetLineNo（）"、"SetLineNo（）"。

6.6.0.1 注意

函数命名应遵循驼峰形式。

6.7 序列命名

序列名的命名推荐为： "seq+_+表名(或缩写)+列名（或缩写）"，示例："seq_dept_deptno"。

7 字段设计

开发者在进行字段设计时，主要需要考虑字段的类型，字段的长度。本文给出几个场景字符类型的推荐值，旨在帮助开发者规范字段设计。

7.1 MySQL 模式

- 数值型字段。

  推荐使用 `bigint` 类型替代 `int` 、 `smallint` 等整型类型，防止以后范围超限。

- 字符型字段。
  - 所有动态字符串建议全部使用 `VARCHAR(N)` 类型。
  - 仅仅只有单字符的字段使用 `CHAR(1)` 。表达是与否概念的字段，建议使用 `CHAR(1)` 类型以节省空间（1 代表 TRUE，0 代表 FALSE），值的内容要统一，所有应用值要统一，例如：表达逻辑删除的字段名 `is_deleted` ，1 表示删除，0 表示未删除。

7.1.0.1 注意

NUMBER(1) 也可以表达是与否的概念，但占用的空间更大些。

- 列的类型禁止使用 `NVARCHAR`、`NCLOB` 等类型。

7.1.0.2 注意

字符数据类型的列可以存储所有字母数字值，但是 `NUMBER` 数据类型的列只能存储数字值。

- 日期时间字段。
  - 有时间精度要求的业务，可以使用 `datetime(6)` 。
  - 对精度没要求的，设置为 `datetime` 即可。
  - 如果将来有国际化需求，建议使用 `timestamp` 。
  - 不推荐使用字符作为时间字段的数据类型，在使用的时候容易造成隐式类型转换
- 数据字段的选择推荐。

  特别是在大表上(百万级)，推荐如下使用方法：
  - 业务内各表时间字段务必统一，推荐使用 `DATE` 类型，对于精度较高的业务可以使用 `TIMESTAMP` 类型。

- IP 所在表如果是大表，推荐使用 NUMBER 数值类型存储，可节省存储空间。前端进行转换。使用数值范围大小跟网段数据保持一致。
- 根据业务需要，IPv4 和 IPv6 也可以分字段存储，采用 VARCHAR(N) 存储。

7.2 Oracle 模式

- 数值型字段
  - 推荐使用 NUMBER 类型存储。当 NUMBER 存储变长、十进制精度的定点数时，写法为 NUMBER(p,s)；当 NUMBER 存储浮点数时，写法为 NUMBER。
  - 小数类型的字段推荐使用 DECIMAL 不推荐使用 BINARY_FLOAT 和 BINARY_DOUBLE，BINARY_FLOAT 和 BINARY_DOUBLE 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不正确的结果。
  - 隐式类型转换时的优先级

    BINARY_DOUBLE 的优先级最高，其次是 BINARY_FLOAT，最后是 NUMBER。
  - 数值字段的取值范围

类型	取值范围	长度(字节数)
NUMBER	1.0 E-130F ~ 1.0 E +126 F（不包括1.0 E +126 F）	4~40
BINARY_FLOAT	1.17549E-38F ~ 3.40282E+38F	4
BINARY_DOUBLE	2.22507485850720E-308 ~ 1.79769313486231E+308	8

- 字符型字段。
  - 推荐使用 VARCHAR2。
  - 比较 VARCHAR2 类型时，按照非填充空格的模式进行比较；而比较 CHAR 类型时，按照填充空格的模式比较。
  - 日期时间字段。
    - TIMESTAMP WITH TIME ZONE 和 TIMESTAMP WITH LOCAL TIME ZONE 两个类型是感知时区的，请注意时区差。
    - 不推荐使用字符作为时间字段的数据类型，在使用的时候容易造成隐式类型转换。

7.3 其他

- **自增列字段**：必须使用 bigint 类型，禁止使用 int 类型，以防止存储溢出。
- 禁止使用外键自引用和级联删除更新的表字段约束定义，以避免重复删除的问题。
- 尽量避免使用枚举列类型：`enum('x','y','z')`，应使用字符串类型替代。
- 如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。
- 字段允许适当冗余，以提高性能，但是必须考虑数据同步的情况。冗余字段应遵循：
  - 不是频繁修改的字段
  - 不是超长字段
- 发生隐式类型转换时，数值类型的优先级低于时间类型，高于字符和所有其他数据类型。
- 合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。

  无符号值可以避免误存负数，且扩大了表示范围，不同的数值范围推荐不同的数据类型，示例如下：

对象	年龄区间	类型	表示范围
人	150 岁之内	unsigned tinyint	无符号值：0 到 255。
龟	数百岁	unsigned smallint	无符号值：0 到 65535。
恐龙化石	约数千万年	unsigned int	无符号值：0 到约 43 亿。
太阳	约 50 亿年	unsigned bigint	无符号值：0 到约 10的 19 次方。

8 表结构设计

在数据库使用过程中，开发者会创建各种表，来进行数据库操作，本文旨在帮助开发者规范数据库中表的结构设计。

8.1 三大范式

在推荐表结构设计之前，先来了解一个数据库概念，数据库设计三大范式。为了建立出冗余更小、结构更合理的数据库，在进行数据库创建的时候要遵循一定的原则，在关系型数据库中这种规范被称为范式。下面简单介绍下三大范式。

8.1.1 第一范式（字段原子值）

第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该表满足了第一范式。

示例：学生表，具体字段的定义如下。

sno	sname	联系方式
01	赵浅	1******1@qq.com
02	孙理	138****1234
03	周吾	135****1234

从表中明显可以看出，联系方式值有邮箱和手机号码，因此不是原子值，不满足第一范式。这里将联系方式拆分为邮箱和手机号码，则每一列都是原子值，从而满足了第一范式。修改后的学生表的字段定义如下。

sno	sname	邮箱	手机号码
01	赵浅	1******1@qq.com	
02	孙理		138****1234
03	周吾		135****1234

8.1.2 第二范式（无部分依赖）

第二范式在第一范式的基础之上对字段的定义要求更进一层。第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

示例：学生教师表，具体字段定义如下：

（sno	tno）~pk~	sname	tname
01	01	赵浅	郑望
01	02	赵浅	冯沉
02	01	孙礼	郑望
02	03	孙礼	楚卫
03	02	周吾	冯沉
03	03	周吾	楚卫

这张表中的主键为（ sno ， tno ），很明显 每个字段都是原子值不可再分，满足第一范式，但 sname 依赖于 sno ， tname 依赖于 tno ，不满足第二范式，因此需要对表进行拆分。将 1 张表拆分为 3 张表具体拆分后的字段如下。

学生表：

sno~pk~	sname
01	赵浅
02	孙礼
03	周吾

教师表：

tno~pk~	tname
01	郑望
02	冯沉
03	楚卫

学生教师关系表：

id~pk~	sno~fk~	tno~fk~
1	01	01
2	01	02
3	02	01
4	02	03
5	03	02
6	03	03

上面3张表字段皆为原子值，且不存在部分依赖，满足第二范式。

8.1.3 第三范式（无传递依赖）

第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

示例：学生教室表，具体字段定义如下

tno~pk~	tname	cno	cname
01	赵浅	01	唐朝班
02	孙礼	01	唐朝班
03	周吾	02	周朝班

从表中明显看出表中各字段皆为原子值，且不存在部分依赖，满足第一、第二范式，但 `cname` 依赖于 `cno`，`cno` 依赖于 `tno`，存在传递依赖，不满足第三范式。将这张表进行拆分为 2 张表，具体如下：

班级表：

cno~pk~	cname
01	唐朝班
02	周朝班

学生班级关系表：

tno~pk~	tname	cno~fk~
01	赵浅	01
02	孙礼	01
03	周吾	02

经过拆分后，表中不存在传递依赖，满足了第三范式。

8.2 普通表结构设计规范

- 表结构设计不应该简单遵循三大范式，应该以业务性能为指导，适当进行数据冗余存储，以减少表的关联从而提升业务性能。冗余字段应遵循：
  - 不是频繁修改的字段。
  - 不是 varchar 超长字段。
- 建表时应该设定主键。
  - 建议使用业务字段做主键或做联合主键，不建议使用自增列做主键。
  - OceanBase 数据库的表存储模型为索引聚集表模型(IOT)，如果用户未指定主键，系统会自动生成一个隐藏主键。
- 表必备两字段： gmt_create ， gmt_modified 。

8.2.3.1 说明

gmt_create ， gmt_modified 的类型选择 DATE （精确到秒）或 TIMESTAMP WITH TIME ZONE （精确到微秒，且带当前时区信息），可以使用 sysdate 或 systimestamp 函数。

- 表、字段需要有 COMMENT 属性。
- 表中所有字段推荐是 NOT NULL 属性，业务可以根据需要定义 DEFAULT 值。
- 多表中的相同列，必须保证列定义一致。
- 进行 join 的关联字段，数据类型保证一致，避免隐式转换。
- 不推荐使用复杂的数据类型，如 blob 或者 json 。
- 表达是与否概念的字段，推荐数据类型是 unsigned tinyint （1 表示是，0 表示否），值的内容要统一。 示例：表达逻辑删除的字段名 is_deleted ，1 表示删除，0 表示未删除。
- 如果修改字段含义或对字段表示的状态追加时，建议及时更新字段注释。

9 分区表设计

本文介绍在 OceanBase 中进行分区表结构设计的推荐设计。

- 关于分区表创建时的注意事项。
  - 如果数据量很大并且访问比较集中时，可以在创建表时使用分区表。
  - 分区表约束注意事项。
    - 建分区表时，表上的每一个主键、唯一键所对应的字段里都必须至少有一个字段包含在表的分区键字段中。
    - 分区表中的全局唯一性建议能通过主键实现的都通过主键实现。
- 关于分区策略，推荐从表的实际用途和应用场景方面进行设计。
  - 实际用途：历史表，流水表。
  - 应用场景：存在明显访问热点的表。
- 关于分区键的选择，使用分区表时要选择合适的拆分键以及拆分策略。这里仅做选择分区类型的基本推荐，更详细的内容请参见 创建和管理分区表。
  - hash 分区：选择区分度较大、在查询条件中出现频率最高的字段作为 hash 分区的分区键。
  - range 和 list 分区：根据业务规则选择合适的字段作为分区键，但分区数量不宜过少。示例：如果是日志类型的大表，根据时间类型的列做 range 分区。
- 关于分区的使用限制。

  hash 分区下，不适合基于分区字段进行范围查询。

10 索引设计

本文介绍在 OceanBase 数据库中进行索引设计的推荐设计。

10.1 索引简介

OceanBase 数据库支持主键索引，唯一索引，也支持二级索引，构成以上索引的可以是单一列，也可以是多个列（复合索引）。在 OceanBase 数据库中，索引可以分为两种类型：本地索引和全局索引。两者之间的区别在于：本地索引与分区数据共用分区，全局索引为单独分区。

10.1.0.1 说明

OceanBase 4.0 版本中，MySQL 默认创建的是本地索引（local），Oracle 模式默认创建的是全局索引（global）。

以 MySQL 模式创建索引的示例如下：

- 创建本地索引

```
obclient> CREATE TABLE t1(id NUMBER PRIMARY KEY,name1 VARCHAR(10),name2
VARCHAR(10));
obclient> CREATE INDEX idx_t1_name1 ON t1(name1);
```

10.1.0.2 说明

如果不指定 `LOCAL` 或 `GLOBAL`，MySQL 模式默认创建本地索引（local）。

- 创建全局索引

```
obclient> CREATE INDEX idx_t1_name2 ON t1(name2) GLOBAL;
```

10.2 索引设计

10.2.1 单值索引要求和建议

- 索引建立需满足最左前缀原则。
- 表中所有带索引的字段建议都是 NOT NULL 属性，通常建议业务根据需要定义 DEFAULT 值。
- 业务上具有唯一特性的字段，即使是组合字段，建议创建为主键。

10.2.1.1 说明

即使唯一索引影响了 INSERT 速度，但这个速度损耗可以忽略，唯一索引提高查找速度是明显的；另外，即使在应用层做了非常完善的校验和控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

- 需要 join 的字段，数据类型需保持一致；多表关联查询时，保证被关联的字段建有索引。

10.2.1.2 说明

在多表关联（join）的场景中，也要注意表索引的使用可以很好地提升 SQL 性能。。

- 页面搜索尽量避免左模糊或者全模糊，如果需要的话可以通过搜索引擎来解决。同时尽量将过滤后的字段作为输入条件，可以避免后台产生大量对结果集的查找。

10.2.1.3 说明

索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

反例：表存在索引 idx_t2_abc(a,b,c)，但 WHERE 条件为： `b = ? and c = ?` ，因条件中并未包含 a，无法使用该索引。t2 表和索引 idx_t2_abc(a,b,c) 的创建语句如下。

```
obclient> CREATE TABLE t2(a NUMBER PRIMARY KEY, b INT, c VARCHAR(10));
obclient> CREATE INDEX idx_t2_abc ON t2(a,b,c);
```

现在看下 where 条件为： `b = ? and c = ?` 的语句的执行计划。从执行计划明显可以看出，name 列为 t2，该语句没有走到该索引，走了全表扫，cost 为 409。

```
obclient> EXPLAIN SELECT a,b,c FROM t2 WHERE b=8889 AND c='a(mbmtwm';
+----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------+
| Query Plan |
+----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
```

```
---------------------+
| ==================================
|ID|OPERATOR |NAME|EST. ROWS|COST|
---------------------------------
|0 |TABLE SCAN|t2 |1 |409 |
==================================


Outputs & filters:
-------------------------------------
0 - output([t2.a], [t2.b], [t2.c]), filter([t2.b = 8889], [t2.c = 'a(mbmtwm']),
access([t2.b], [t2.c], [t2.a]), partitions(p0)
|
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
---------------------+
1 row in set (0.01 sec)
```

正例：上述索引调整字段顺序为 idx_bca(b,c,a) 或 idx_cba(c,b,a)，过滤条件仍为 `b = ? and c = ?`，则该复合索引会被使用。t2 表和索引 idx_t2_bca(b,c,a) 的创建语句如下。

```
obclient> CREATE TABLE t2(a NUMBER PRIMARY KEY, b INT, c VARCHAR(10));
obclient> CREATE INDEX idx_t2_bca ON t2(b,c,a) ;
```

现在看下 where 条件为：`b = ? and c = ?` 的语句的执行计划。从执行计划明显可以看出，name 列为 t2(idx_t2_bca)，该语句走了正确的索引，cost 为 46，明显下降了很多。

```
obclient> EXPLAIN SELECT a,b,c FROM t2 WHERE b=8889 AND c='a(mbmtwm';
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------
-----------------------------------------+
| Query Plan |
+------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----------------------------------------+
| ============================================
|ID|OPERATOR |NAME |EST. ROWS|COST|
--------------------------------------------
|0 |TABLE SCAN|t2(idx_t2_bca)|1 |46 |
============================================


Outputs & filters:
-------------------------------------
0 - output([t2.a], [t2.b], [t2.c]), filter(nil),
access([t2.b], [t2.c], [t2.a]), partitions(p0)
|
+------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----------------------------------------+
1 row in set
```

- 如果有 ORDER BY 的场景，请注意利用索引的有序性，避免出现 file_sort。ORDER BY 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file_sort 的情况，影响查询性能。

  正例：`where a=? and b=? order by c;` 索引：idx_t3_abc(a,b,c)。t3 表和索引 idx_t3_abc(a,b,c) 的创建语句如下。

```
obclient> CREATE TABLE t3(a NUMBER PRIMARY KEY, b INT, c INT);
obclient> CREATE INDEX idx_t3_abc ON t3(a,b,c);
```

现在来看下 `where a=? and b=? order by c;` 语句的执行计划，OPERATOR 为 TABLE GET，COST 为 46。

```
obclient> EXPLAIN SELECT a,b,c FROM t3 WHERE a=117 AND b=67176 ORDER BY c;
+----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+
| Query Plan |
+----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+
| ==================================
|ID|OPERATOR |NAME|EST. ROWS|COST|
---------------------------------
|0 |TABLE GET|t3 |1 |46 |
==================================


Outputs & filters:
-------------------------------------
0 - output([t3.a], [t3.b], [t3.c]), filter([t3.b = 67176]),
access([t3.a], [t3.b], [t3.c]), partitions(p0)
|
+----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------+
1 row in set
```

反例：索引中有范围查找，那么索引有序性无法利用，如：`WHERE a>10 ORDER BY b;` 索引 a_b 无法排序。因为索引中可能有以下数据已经按升序排好： {a = 11, b = 2} {a = 11, b = 3} {a = 12, b = 1} 此时若直接按序输出，则 b=1 排在 b=2 和 b=3 之后，顺序错误。

- 利用覆盖索引进行查询操作，可以避免回表操作。

10.2.2 组合索引要求和建议

- 如果索引包含多个列，要明智选择列的顺序，一般来说，索引的第一列应当是一个高基数列（基数：数据列所包含不同值的数量）即区分度最高的在最左边。

  示例：如果 `where a=? and b=?`，a 列的几乎接近于唯一值，那么只需要单建 idx_a 索引即可。

- 对 ORDER BY、GROUP BY、DISTINCT 子句中频繁使用的列添加到索引后面，形成覆盖索引避免回表。

- 若存在 `where a=? and b=?` 的查询条件，使用组合索引 idx_ab(a,b)，而不要分别在 a、b 字段上建立 idx_a(a)，idx_b(b) 两个索引，后者将无法同时用到两个索引

- 合理设计组合索引同时满足多场景的查询，减少索引的冗余性：
  - 多条语句可以共用同一个索引，其中某些语句只要覆盖索引前缀即可。如有两条语句条件分别为：`a = ? and b = ?`，和 `b = ?`，那么组合索引 idx_ba(b,a) 就可以同时为两条语句使用，不必再创建索引 idx_ab(a,b) 和 idx_b(b) 上分别建立索引。
  - 非必要情况下不要使用全局索引，尤其要杜绝 NDV 值小而且使用频率不高的全局索引。
  - where 条件中如包含有分区键字段，则可以在分区键字段和条件里的其它字段上建立联合本地索引。
  - 全局分区索引适合查询返回数据量较少的场景，对于返回数据量较大的场景建议在全局索引与本地索引间进行比较测试。
  - 选择 NDV 值大的字段作为全局分区索引的分区键。
  - 建议不要对分区表分区键、全局分区索引分区键所在字段进行 update 操作，若业务确有需要务必打开分区表的 row movement 功能。

    ```
    obclient> alter table XXX enable row movement;
    ```

  - 避免重复索引：冗余的索引影响数据的增删改效率，同时浪费存储成本。
  - 主键默认创建索引和唯一性约束。

- 索引 idx_abc(a,b,c) 已创建的情况下不需要再创建索引 `idx_a(a)` 和 `idx_ab(a,b)` 。

10.2.3 分区表索引建议

- 按照本地索引->全局分区索引->全局索引的顺序进行选择，非必要情况下不建议使用全局索引。
- 减少不必要的全局索引定义。

  全局索引维护代价很高，数据的增删改都需要维护全局索引， 不适宜大量使用，数据的查询要尽量使用主键或只要数据能在分区内保证唯一，则没必要使用全局索引。

10.2.3.4 说明

全局索引会降低 DML 的性能，可能会因此产生分布式事务。对于分区表的索引定义，如果默认不加 local 关键字即为全局索引定义，所以如果定义本地索引语法为 `CREATE INDEX <index_name> ON <table_name> (column, column) LOCAL;`

- 分区表上的主键必须包括表的分区键。
- 分区表上的本地唯一键的交集必须包括表的分区键。

10.2.4 索引使用注意事项

- 上线 SQL 前确认新建索引已生效。
- 索引修改流程为：先新建索引待新索引生效后，确保旧索引无用再删除。
- 索引过多时，请根据需要删除不使用的索引，避免索引越建越多。

10.2.4.5 注意

这个操作风险很大，必须确认删除的索引没有其他 SQL 使用。

- 禁止 `ALTER TABLE ADD INDEX/DROP INDEX` 同时混用在一条 DDL。
- 创建索引时避免有如下极端误解：
  - 误认为一个查询就需要建一个索引。
  - 误认为索引会消耗空间、严重拖慢更新和新增速度。
  - 误认为唯一索引一律需要在应用层通过"先查后插"方式解决。
- 全局索引使用注意。

  对 `PARTITION` 进行运维时，请注意 `DROP` 或 `TRUNCATE` 操作会导致全局索引失效。

11 其他结构设计

11.1 租户结构设计

从 OceanBase 数据库整体架构上，租户的概念类似实例。

一般一条业务线或者子业务单独建一个租户。

12 字符集规范

本文介绍在 OceanBase 中进行字符集选择的规范。

用户可以在租户级、Database 级、表级、字段级、session 级设置字符集，目前 OceanBase 支持 `utf8mb4` 、 `gbk` 、 `gb18030` 、 `binary` 、 `utf16` 等字符集。

12.0.0.1 说明

- 为支持无缝迁移，OceanBase 在语法上将 `UTF8` 视为 `UTF8MB4` 的同义词。
- 数据库字符集暂不支持修改。

以 gbk 字符集为例：

- 在创建租户时设置字符集
  - 可以在 create tenant 语句添加 charset 设置，添加 `"charset=gbk"` 。

```
create tenant oracle replica_num = 1,
resource_pool_list =('pool1'),
charset = gbk
set
ob_tcp_invited_nodes = '%',
ob_compatibility_mode = 'oracle',
parallel_servers_target = 10,
ob_sql_work_area_percentage = 20,
secure_file_priv = "";
```

  - 可以在 OCP 中创建租户时，选择字符集为 gbk。

12.0.0.1 注意

- Oracle 模式的字符集是租户级别，在 gbk 租户中，所有用户表的 char、varchar2、clob 字段都是 gbk 字符集，系统表的 char 、varchar2 等字段仍然保留 utf8 字符集。
- Oracle 模式的租户字符集是不可修改的，即不能通过 `alter` 语句修改租户、库、表、列的字符集。

- 设置客户端（链路）字符集
  - 客户端（链路）字符集是配置 Client 和 Server 之间交互使用的字符集设置。

Client 将 SQL 字符串的发给 Server 执行，Server 将执行结果返回给 Client，在这个过程中，Server 需要明确知道 Client 使用的字符集是什么，才能正确的解析，执行和返回结果。在不同环境下，Client 可以是 OBClient、jdbc 或 OCI 等，因此有时候也会叫做链路字符集。

- 租户字符集与客户端字符集没有直接关系，是相互独立配置的。

  GBK 租户可以被 GBK 的客户端连接，也可以被 UTF8 的客户端连接。

  - 当客户端字符集为 GBK 时，Server 会按照 GBK 解析和执行接收到的 SQL 语句。
  - 当客户端字符集为 UTF8 时，Server 会按照 UTF8 解析和执行收到的 SQL 语句。
- 配置方法
  - 永久性修改

    ```
    set global character_set_client = gbk;
    set global character_set_connection = gbk;
    set global character_set_results = gbk;
    ```

    - character_set_client：客户端字符集。
    - character_set_connection：连接字符集。在 oracle 租户下应设置为与 character_set_client 一致的值。
    - character_set_results：Server 返回给 Client 结果的字符集。

      一般来说，客户端发给服务器，和服务器返回给客户端字符串的字符集是统一的，因此在 Oracle 模式的租户下，这三个值应该保持一致；MySQL 模式使用 3 个变量的作用是可以灵活进行配置，一般使用场景将三个变量都配置成客户端的字符集即可。
  - 临时修改（仅对本 Session 生效）
    - 方法一：

      ```
      set character_set_client = gbk;
      set character_set_connection = gbk;
      set character_set_results = gbk;
      ```

    - 方法二：

      ```
      set names gbk;
      ```

- 设置客户端字符集
  - 使用 jdbc 连接 OceanBase 数据库，GBK 链路一般在 url 里修改参数设置，添加 `characterEncoding=gbk` 。

```
String url = "jdbc:oceanbase://xxx.xxx.xxx.xxx:xxxx?useSSL=false&useUnicode=true&characterEncoding=gbk&connectTimeout=30000&rewriteBatchedStatements=true";
```

- 使用 OBClient 客户端连接数据库，GBK 链路 bash 环境变量推荐使用 `zh_CN.GBK` 的超集 `zh_CN.GB18030` 。
  - 修改 bash 环境变量

    ```
    export LANG=zh_CN.GB18030
    export LC_ALL=zh_CN.GB18030
    ```

  - 修改终端的编码设置，将当前窗口设置为 gbk 编码。请根据终端界面指示进行操作。

12.0.0.1 注意

除了将数据库（observer 进程）配置成 GBK 链路之外，客户端、驱动也要做相应的配置，若环境配置错误，可能会显示乱码。

13 数据库连接规范

本文介绍在 OceanBase 中进行数据库连接的规范。

- 使用连接池的时候需要设置一个连接最大的空闲时间。
- 前端程序连接数据库或者数据库代理层，对于 jdbc 建议连接超时设置为 1 秒，要具备失败重连机制，且失败重连必须有间隔时间。
- 程序端日志必须记录连接数据库的标准 OceanBase 错误码以及所连接的数据库信息（比如 IP 和 PORT，数据库用户名），用于 DBA 排查错误
- 对于有连接池的前端程序，必须根据业务需要配置初始、最小、最大连接数，建议超时时间设置为 30 秒，且要设置连接检测或空闲超时，以及连接回收机制（最大 3600 秒）。
- 程序端使用的连接数据库的 so 库包、jar 库包以及客户端数据库版本，必须与线上数据库服务器的版本兼容。
- 切换 DB，不推荐执行 `use <dbname>`，推荐使用 `Connection.setCatalog(dbname)` 接口。
- 设置 ReadOnly，不推荐执行 `set session transaction readonly`，推荐使用 `Connection.setReadOnly(xx)` 接口。
- 执行单个事务语句（无论是单 SQL 事务语句，还是多 SQL 的事务语句）之前，必须重新 `getConnection`，执行完后必须 `close connection`。不能在一个过程中执行多个事务，即需按照以下方式使用：`getConnection，执行单事务语句，close connection`。需要特殊说明的原因是，OceanBase 当前限制单个事务只能在单个 server上 完成整个生命周期，即在单个 server上开始事务，执行事务，结束事务。不保证在 A server 上开启的事务，在 B server上同样有效。具体影响到 JAVA Connector 中，就是当前连接处于事务中时，下一条 SQL 只能路由到上一个 server 上，不能路由切换到其它 server，当前连接不在事务中时，可以根据下一条SQL路由到最佳 server 上。

14 注释的使用

14.1 表注释的使用

给表的每个字段加上注释，有助于更好的通过建表语句理解表及其各个字段所起到的作用，有助于后续开发人员共同开发，是对数据库设计文档的一个有利的补充，是一个天然的功能说明。

表加注释有 2 种方式。

- 在创建表的同时加注释。

```
obclient> CREATE TABLE student(
id INT PRIMARY KEY AUTO_INCREMENT COMMENT '学号',
name VARCHAR(200) COMMENT '姓名',
age INT COMMENT '年龄') COMMENT='学生信息';
```

- 创建表后，使用 `alter` 语句加注释。

```
obclient> CREATE TABLE student( id INT PRIMARY KEY AUTO_INCREMENT , name VARCHAR(200) , age INT);
obclient> ALTER TABLE student COMMENT '学生信息';
obclient> ALTER TABLE student MODIFY COLUMN id INT COMMENT '学号';
obclient> ALTER TABLE student MODIFY COLUMN name VARCHAR(200) COMMENT '姓名';
obclient> ALTER TABLE student MODIFY COLUMN age INT COMMENT '年龄';
```

这里推荐第二种在创建表之后，使用 `ALTER` 语句加注释，便于后续对表进行维护。

14.2 SQL 语句注释的使用

对于普通 SQL 语句，OceanBase 数据库支持以下三种注释方法：

- 从 `#` 到行尾，可以注释一行。
- 从 `--` 到行尾，可以注释一行。
- 从 `/*` 到 `*/`，可以注释多行。

14.2.0.1 说明

SQL 语句不支持嵌套注释。

特别地，OceanBase 数据库支持将 `/*! */` 内部的语句当成 SQL 处理并执行注释中的语句。如果 `/*! */` 里出现语法错误，数据库会默认当作注释处理。

注释的格式如下：

```
/*![表示版本号的连续数字字符]<空格><任意 SQL 语句 >*/
```

示例：

- 为表 `t1` 创建 Hash 分区，分区数为 `8` 。

```
obclient> CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 INT) /*!50100
PARTITION BY HASH(c1) PARTITIONS 8*/;
Query OK, 0 rows affected
```

- 使用 `ENABLE KEYS` 更新非唯一索引，批量插入的时候可以使用 `DISABLE KEYS` 先不更新，所有索引插入完之后再更新索引。

```
obclient> /*!ALTER TABLE t1 DISABLE KEYS */;
Query OK, 0 rows affected



obclient> /*!ALTER TABLE t1 ENABLE KEYS */;
Query OK, 0 rows affected
```

- 支持执行 PL 的 `DROP IF EXISTS` 语法。

```
obclient> /*!50003 DROP PROCEDURE IF EXISTS */;
Query OK, 0 rows affected
```

15 ORM 规约

ORM 是对象关系映射（Object Relational Mapping，简称 ORM），是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换。从效果上说，它其实是创建了一个可在编程语言里使用的 "虚拟对象数据库"。

- 不用 resultClass 当返回参数，即使所有类属性名与数据库字段一一对应，也需要定义；反过来，每一个表也必然有一个与之对应。

15.0.0.1 说明

配置映射关系，使字段与 DO 类解耦，方便维护。

- sql.xml 配置中参数注意：`#{}` 和 `#param#` 中不要使用 `${}`，此种方式容易出现 SQL 注入。
- iBATIS 自带的 `queryForList(String statementName,int start,int size)` 不推荐使用。

15.0.0.1 说明

该方法的实现方式是在数据库取到 statementName 对应的 SQL 语句的所有记录，再通过 subList 取 start，size 的子集合，线上因为这个原因曾经出现过 OOM。解决该问题的方法，可以在 sqlmap.xml 中引入 `#start#`， `#size#` 。示例如下：

```
Map


<string,> map = newHashMap


<string,>();
map.put("start", start);
map.put("size", size);
```

不允许直接拿 HashMap 与 HashTable 作为查询结果集的输出。

反例：某开发人员为避免写一个 <resultMap> ，直接使用 HashTable 来接收数据库返回结果把 bigint 转成 Long 值，而线上由于数据库版本不一样，解析成 BigInteger，从而导致出现生

更新数据表记录时，必须同时更新记录对应的 gmt_modified 字段值为当前时间。

不写一个大而全的数据更新接口，传入为 POJO 类，不管是不是自己的目标更新字段，都进行 c1=value1,c2=value2,c3=value3; 这是不对的。执行 SQL 时，不要更新无改动的字段，一是低；三是binlog增加存储。

异常处理

异常处理注意事项

OceanBase 推荐只在下列情形里使用异常处理程序：

- 您预计可能有异常，并期望处理它。

  比如说，您预计最终一个 SELECT INTO 语句可能会返回空行，导致 OceanBase 上报一个 NO_DATA_FOUND 。您期望通过子程序或者 STORED PROCEDURE 块处理这个异常（后让程序能继续执行下去。示例如下：

```
create or replace procedure select_dept(
num_deptno in number,--定义in模式变量，要求输入部门编号
var_dname out dept.dname%type,--定义out模式变量，可以存储部门名称并输出
var_loc out dept.loc%type) is
begin
select dname,loc
into var_dname,var_loc
from dept
where deptno = num_deptno;--检索某个部门编号的部门信息
exception
when no_data_found then --若select语句无返回记录
```

```
dbms_output.put_line('该部门编号的不存在');--输出信息
end select_dept;
```

- 您必须放弃或关闭一个资源。示例如下：

```
...
file := UTL_FILE.OPEN ...
BEGIN
statement statement]...
EXCEPTION
WHEN OTHERS THEN
UTL_FILE.FCLOSE(file); -- then you want to close the file.
RAISE; -- 继续上报异常.
END;
UTL_FILE.FCLOSE(file);
...
```

- 在子程序代码的顶层，您需要记录错误日志。示例如下：

```
BEGIN
proc(...); -- 调用其他子程序
EXCEPTION
WHEN OTHERS THEN
log_error_using_autonomous_transaction(...); -- 使用自治事务记录日志
RAISE; -- 继续上抛异常.
END;
/
```

异常处理示例

OceanBase 提供了两种异常处理机制。一种是在 SQL 语句编写时，通过 PL 中的 exception 进
种是应用层代码开发时，使用不同的驱动会有对应的异常处理方式。

PL 异常处理

PL 异常错误包含系统内部异常、预定义异常和用户定义异常三种类型。针对这 3 种类型，如下

- 系统内部异常。

  示例：通过命名捕获系统内部异常。

```
obclient>CREATE TABLE dept(
dept_id NUMBER(10,0),
dname VARCHAR(15),
loc VARCHAR(20),
CONSTRAINT pk_dept PRIMARY KEY(dept_id)
);
Query OK, 0 rows affected (0.07 sec)

obclient>INSERT INTO dept VALUES (100,'ACCOUNTING','Los Angeles'),
(110,'OPERATIONS','CHICAGO'),
(111,'SALES','NEW YORK');

obclient>DECLARE
DUPLICATED_DEPT_ID EXCEPTION;
PRAGMA EXCEPTION_INIT(DUPLICATED_DEPT_ID, -5024);
BEGIN
UPDATE dept SET dept_id=110
where dept_id=100;
EXCEPTION
WHEN DUPLICATED_DEPT_ID THEN
DBMS_OUTPUT.PUT_LINE('Duplicated Department ID!');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(SQLCODE||'---'||SQLERRM);
END;
```

```
/
Query OK, 0 rows affected (0.03 sec)

Duplicated department id!
```

- 预定义异常。

```
obclient>CREATE TABLE employees(
empno NUMBER(4,0),
empname VARCHAR(10),
job VARCHAR(10),
deptno NUMBER(2,0),
salary NUMBER(7,2),

CONSTRAINT PK_emp PRIMARY KEY (empno)

);
Query OK, 0 rows affected (0.07 sec)

obclient>INSERT INTO employees VALUES (200,'Jennifer','AD_ASST',1,15000),
(202,'Pat','MK_REP',3,12000),(113,'Karen','PU_CLERK', 4,null),(201,'Michael','MK_M
Query OK, 4 rows affected (0.06 sec)
Records: 4 Duplicates: 0 Warnings: 0

obclient>DECLARE
v_empid employees.empno%TYPE;
v_sal employees.salary%TYPE;
BEGIN
v_empid := 100;
SELECT salary INTO v_sal FROM employees
```

```
WHERE empno=v_empid;
IF v_sal<=10000 THEN
UPDATE employees SET salary=salary+100 WHERE empno=v_empid;
DBMS_OUTPUT.PUT_LINE('Employee '||v_empid||' updated');
ELSE
DBMS_OUTPUT.PUT_LINE('Employee '||v_empid||' ignored');
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN

-- 如果员工号 v_empid 不存在，触发 NO_DATA_FOUND 异常
DBMS_OUTPUT.PUT_LINE('Employee id '||v_empid||' not found');
WHEN TOO_MANY_ROWS THEN

-- 如果员工号 v_empid 不止一条，触发 TOO_MANY_ROWS 异常
DBMS_OUTPUT.PUT_LINE('Duplicated id: '||v_empid);
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(SQLCODE||'---'||SQLERRM);
END;
/
Query OK, 0 rows affected (0.06 sec)

Employee id 100 not found
```

- 用户自定义异常，分为自定义异常和自定义错误码。
  - 自定义异常。

    ```
    obclient>DECLARE
    v_empid employees.empno%TYPE;
    v_sal employees.salary%TYPE;
    ```

```
-- 1.定义异常名称 SALARY_NOT_SET
SALARY_NOT_SET EXCEPTION;
BEGIN
v_empid := 113;
SELECT salary INTO v_sal FROM employees
WHERE empno=v_empid;
IF v_sal<=10000 THEN
UPDATE employees SET salary=salary+100 WHERE empno=v_empid;
DBMS_OUTPUT.PUT_LINE('Employee '||v_empid||' updated');
ELSIF v_sal is NULL THEN

-- 2. 当 v_sal 为空时，触发这个异常
RAISE SALARY_NOT_SET;
ELSE
DBMS_OUTPUT.PUT_LINE('Employee '||v_empid||' ignored');
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Employee id '||v_empid||' not found');

-- 3. 对 SALARY_NOT_SET 这个异常进行处理
WHEN SALARY_NOT_SET THEN
DBMS_OUTPUT.PUT_LINE('Salary not set: '||v_empid);
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(SQLCODE||'---'||SQLERRM);
END;
/
Query OK, 0 rows affected (0.07 sec)
```

```
Salary not set: 113
```

- 自定义错误码。

```
obclient>DECLARE
v_empid employees.empno%TYPE;
v_sal employees.salary%TYPE;
BEGIN
v_empid := 103;
SELECT salary INTO v_sal FROM employees
WHERE empno=v_empid;
IF v_sal is NULL THEN

-- 抛出错误 20999
RAISE_APPLICATION_ERROR(-20999, 'The salary of employee is not found');
ELSIF v_sal<=1500 THEN
UPDATE employees SET salary=salary+100 WHERE empno=v_empid;
DBMS_OUTPUT.PUT_LINE('Employee '||v_empid||' updated');
ELSE
DBMS_OUTPUT.PUT_LINE('Employee '||v_empid||' ignored');
END IF;
END;
/
OBE-20999: The salary of employee is not found
```

更多关于 PL 异常处理的内容，请参见官网《 **PL 参考** 》中的 **PL 异常处理** 章节。

OceanBase JDBC 驱动使用示例

对于有直接依赖底层驱动错误类型的 SQL 语句，建议 ORM 框架统一使用标准的 JDBCTempla
SqlMapClientTemplate(SqlMapClientDaoSupport) 等标准模板类，这些实现会将底层的错

架的标准异常类型。如使用了标准模板类，则可以不直接感知底层驱动类型 com.alipay.ocean
mysql.exceptions.jdbc4.MySQLIntegrityConstraintViolationException ，参考如下示例。

```
try {
// 业务逻辑
} catch (Exception e) {
if (e instanceof DataIntegrityViolationException
&& ((DataIntegrityViolationException) e).contains(DuplicateKeyException.class)) {
// 唯一性约束冲突异常处理
}
}
```

另外需要检查下其他驱动异常 com.alipay.oceanbase.obproxy.mysql.exceptions .*，如果
并调整为对应的 Spring 标准异常。

说明

MySql 模式推荐使用 MySQL 数据库原生 jdbc driver。

这里仅以 OceanBase JDBC 驱动为示例。其他更多的不同的驱动对异常的处理方式请从对应的
习。