ALIBABA CLOUD

阿里云

专有云企业版

云数据库 OceanBase

用户指南

产品版本：V3.13.0

文档版本：20210105

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

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

1.什么是云数据库OceanBase 26
2.快速入门 27
2.1. 概述 27
2.2. 登录OceanBase运维控制台 27
2.3. 登录OceanBase用户控制台 28
2.4. 安装 OBProxy 29
2.5. 新建集群 31
2.6. 新建租户 34
3.OB 云平台 38
3.1. 集群管理 38
3.1.1. 概述 38
3.1.2. 集群概览 38
3.1.3. 管理单个集群 39
3.1.3.1. 查看集群总览 39
3.1.3.2. 管理集群 40
3.1.3.2.1. 新建备集群 41
3.1.3.2.2. 升级版本 42
3.1.3.2.3. 修改密码 43
3.1.3.2.4. 重启集群 44
3.1.3.2.5. 停止集群 45
3.1.3.2.6. 删除集群 45
3.1.3.3. 管理 Zone 46
3.1.3.3.1. 新建 Zone 46
3.1.3.3.2. 添加 OBServer 48
3.1.3.3.3. 重启 Zone 49
3.1.3.3.4. 停止 Zone 49

3.1.3.3.5. 删除 Zone 50
3.1.3.4. 管理 OBServer 50
3.1.3.4.1. 重启 OBServer 50
3.1.3.4.2. 启动 OBServer 50
3.1.3.4.3. 停止 OBServer 51
3.1.3.4.4. 替换 OBServer 51
3.1.3.4.5. 删除 OBServer 51
3.1.3.5. 查看集群拓扑图 52
3.1.3.6. 租户管理 53
3.1.3.6.1. 查看租户资源分布 54
3.1.3.6.2. 新建租户 54
3.1.3.6.3. 查看租户列表 57
3.1.3.6.4. 查看租户监控 57
3.1.3.6.5. 管理租户 59
3.1.3.7. 性能监控 60
3.1.3.8. 合并管理 61
3.1.3.8.1. 合并详情 61
3.1.3.8.2. 合并统计 62
3.1.3.8.3. 执行合并 63
3.1.3.8.4. 修改合并配置 64
3.1.3.9. 参数管理 65
3.1.3.9.1. 查看参数列表 65
3.1.3.9.2. 修改参数 66
3.1.3.9.3. 集群参数类型说明 67
3.1.3.9.4. 查看参数修改历史 68
3.2. 租户管理 69
3.2.1. 概述 69
3.2.2. 新建租户 69

3.2.3. 查看租户列表 72
3.2.4. 管理租户 74
3.2.4.1. 管理租户副本 74
3.2.4.2. 修改 Zone 优先级 76
3.2.4.3. 修改租户白名单 76
3.2.4.4. 修改密码 78
3.2.4.5. 查看租户拓扑图 79
3.2.4.6. 查看性能监控 83
3.2.4.7. 会话管理 86
3.2.4.7.1. 查看租户会话 86
3.2.4.7.2. 查看会话统计 87
3.2.4.8. 参数管理 88
3.2.4.8.1. 查看参数列表 88
3.2.4.8.2. 修改参数 89
3.2.4.8.3. 查看参数修改历史 90
3.3. 主机管理 91
3.3.1. 查看主机信息 91
3.3.2. 删除主机 92
3.4. 告警管理 93
3.4.1. 概述 93
3.4.2. 告警相关概念 94
3.4.3. 告警项配置 97
3.4.3.1. 新建告警项 97
3.4.3.2. 查看告警项 99
3.4.3.3. 告警分组 99
3.4.3.4. 编辑告警项 102
3.4.3.5. 删除告警项 104
3.4.4. 告警通道配置 105

3.4.4.1. 通道配置说明 105
3.4.4.2. 新建告警通道 105
3.4.4.3. 查看告警通道 110
3.4.4.4. 编辑告警通道 111
3.4.4.5. 删除告警通道 115
3.4.4.6. 配置样例 115
3.4.5. 告警订阅配置 121
3.4.6. 查看告警事件 124
3.4.7. 查看告警通知记录 127
3.4.8. 屏蔽告警 128
3.4.9. OB 日志过滤 129
3.5. OBProxy 运维 130
3.5.1. 安装 OBProxy 130
3.5.2. 升级 OBProxy 132
3.5.3. 重启 OBProxy 133
3.5.4. 下线 OBProxy 133
3.5.5. 启动参数说明 134
3.6. 备份与恢复 142
3.6.1. 概述 142
3.6.2. 进入备份恢复界面 142
3.6.3. 准备工作 143
3.6.4. 管理备份恢复配置 144
3.6.4.1. 新增备份配置 144
3.6.4.2. 新增恢复配置 146
3.6.4.3. 管理配置 148
3.6.5. 安装备份恢复组件 149
3.6.5.1. 安装备份恢复组件 149
3.6.5.2. 运维备份恢复组件 150

3.6.6. 执行备份 151
3.6.6.1. 集群备份调度 151
3.6.6.2. 租户备份调度 153
3.6.6.3. 立即备份 155
3.6.7. 恢复备份数据 156
3.6.8. 监控和告警配置 157
3.7. 用户与权限管理 159
3.7.1. 访问控制概述 159
3.7.2. 用户管理 159
3.7.2.1. 新建用户 159
3.7.2.2. 查看用户列表 160
3.7.2.3. 查看用户详情 161
3.7.2.4. 复制用户 161
3.7.2.5. 编辑用户 162
3.7.2.6. 删除用户 163
3.7.3. 角色管理 164
3.7.3.1. 概述 164
3.7.3.2. 新建角色 166
3.7.3.3. 查看角色信息 167
3.7.3.4. 复制角色 168
3.7.3.5. 编辑角色 169
3.7.3.6. 删除角色 170
3.8. 任务管理 171
3.8.1. 查看任务列表 171
3.8.2. 查看任务详情 171
3.9. 系统参数管理 173
3.9.1. 查看系统参数 173
3.9.2. 修改系统参数 174

3.10. 个人中心 175
3.10.1. 设置个人信息 175
3.10.2. 修改登录密码 176
3.10.3. 密码箱 177
3.11. 附录 177
3.11.1. 附录1. OCP 配置参数列表 177
3.11.2. 附录2. OCP 资源单元规格表 186
3.11.3. 附录3. OCP 错误信息表 187
3.11.4. 附录4. OCP 告警模板变量 200
3.11.5. 附录5. 后台任务列表 206
3.11.6. 附录6. 监控指标 216
4.OB 开发者中心 232
4.1. 创建连接 232
4.2. 管理连接 234
4.3. 管理会话 234
4.4. 使用工作台 237
4.4.1. 概述 237
4.4.2. SQL 窗口 237
4.4.3. 匿名块窗口 241
4.4.4. 已保存的脚本 244
4.5. 使用回收站 245
4.6. 数据导出和导入 246
4.6.1. 概述 246
4.6.2. 导出导入格式 247
4.6.3. 批量导出与导入 248
4.6.4. 单表导出与导入 250
4.6.5. 任务管理 253
4.7. 数据库对象 254

4.7.1. 表对象 255
4.7.1.1. 概述 255
4.7.1.2. 新建表 255
4.7.1.3. 表列表 260
4.7.1.4. 表属性管理 261
4.7.1.5. 表数据管理 264
4.7.2. 视图对象 264
4.7.2.1. 概述 265
4.7.2.2. 新建视图 265
4.7.2.3. 管理视图 266
4.7.3. 函数对象 267
4.7.3.1. 概述 268
4.7.3.2. 新建函数 268
4.7.3.3. 管理函数 269
4.7.4. 存储过程对象 271
4.7.4.1. 概述 271
4.7.4.2. 新建存储过程 272
4.7.4.3. 管理存储过程 273
4.7.5. 序列对象 275
4.7.5.1. 概述 276
4.7.5.2. 新建序列 276
4.7.5.3. 管理序列 277
4.7.6. 程序包对象 278
4.7.6.1. 概述 279
4.7.6.2. 新建程序包 279
4.7.6.3. 管理程序包 280
4.8. 帮助中心 283
5.SQL 参考（MySQL 模式） 285

5.1. 基本元素 285
5.1.1. 数据类型 285
5.1.2. 表达式 289
5.1.3. 类型转换 289
5.1.4. 字符集 289
5.1.5. 排序规则 289
5.1.6. 数据比较规则 290
5.1.7. 字面量 290
5.1.8. 注释 291
5.2. 运算符 292
5.2.1. 算术运算符 292
5.2.2. 位运算符 293
5.2.3. 比较运算符 293
5.2.4. 逻辑运算符 295
5.2.5. 日期时间运算符 295
5.2.6. 字符连接运算符 296
5.2.7. 层次查询运算符 296
5.2.8. 排序规则运算符 297
5.3. 函数 297
5.3.1. 函数 297
5.3.2. 聚集函数 345
5.3.3. 分析函数 351
5.3.4. 信息函数 364
5.3.5. 其它函数 365
5.4. 查询和子查询 368
5.4.1. 概述 368
5.4.2. 连接 370
5.4.3. 集合 372

5.5. SQL语句 376
5.5.1. 通用语法 376
5.5.2. ALTER DATABASE 381
5.5.3. ALTER OUTLINE 382
5.5.4. ALTER RESOURCE POOL 383
5.5.5. ALTER RESOURCE UNIT 384
5.5.6. ALTER SYSTEM 386
5.5.7. ALTER TABLE 409
5.5.8. ALTER TABLEGROUP 414
5.5.9. ALTER TENANT 416
5.5.10. ALTER USER 418
5.5.11. CREATE DATABASE 420
5.5.12. CREATE INDEX 421
5.5.13. CREATE OUTLINE 423
5.5.14. CREATE RESOURCE POOL 425
5.5.15. CREATE RESOURCE UNIT 425
5.5.16. CREATE SYNONYM 427
5.5.17. CREATE TABLE 429
5.5.18. CREATE TABLEGROUP 435
5.5.19. CREATE TENANT 438
5.5.20. CREATE USER 440
5.5.21. CREATE VIEW 443
5.5.22. DELETE 444
5.5.23. DROP DATABASE 449
5.5.24. DROP INDEX 450
5.5.25. DROP OUTLINE 451
5.5.26. DROP RESOURCE POOL 451
5.5.27. DROP RESOURCE UNIT 452

5.5.28. DROP TABLE 452
5.5.29. DROP TABLEGROUP 453
5.5.30. DROP TENANT 454
5.5.31. DROP SYNONYM 454
5.5.32. DROP USER 455
5.5.33. DROP VIEW 456
5.5.34. EXPLAIN 457
5.5.35. FLASHBACK DATABASE 462
5.5.36. FLASHBACK TABLE 463
5.5.37. GRANT 465
5.5.38. INSERT 470
5.5.39. KILL 475
5.5.40. PURGE DATABASE 476
5.5.41. PURGE INDEX 477
5.5.42. PURGE RECYCLEBIN 478
5.5.43. PURGE TABLE 479
5.5.44. RENAME TABLE 479
5.5.45. RENAME USER 481
5.5.46. REPLACE 482
5.5.47. REVOKE 484
5.5.48. SAVEPOINT 487
5.5.49. SCHEMA 489
5.5.50. SELECT 489
5.5.51. SESSION 496
5.5.52. SET PASSWORD 497
5.5.53. SHOW GRANTS 498
5.5.54. SHOW RECYCLEBIN 499
5.5.55. TRANSACTION 500

5.5.56. TRUNCATE TABLE 502
5.5.57. UPDATE 503
6.SQL 参考（Oracle 模式） 511
6.1. 与 Oracle 兼容性对比 511
6.2. SQL 概述 515
6.3. 伪列 517
6.4. 基本元素 520
6.4.1. 内建数据类型 520
6.4.1.1. 内建数据类型概述 520
6.4.1.2. 字符数据类型 520
6.4.1.2.1. 字符数据类型概述 520
6.4.1.2.2. CHAR 数据类型 521
6.4.1.2.3. NCHAR 数据类型 522
6.4.1.2.4. NVARCHAR2 数据类型 524
6.4.1.2.5. VARCHAR2 数据类型 524
6.4.1.2.6. VARCHAR 数据类型 525
6.4.1.3. 数值数据类型 526
6.4.1.3.1. 数值数据类型概述 526
6.4.1.3.2. NUMBER 数据类型 527
6.4.1.3.3. FLOAT 数据类型 529
6.4.1.3.4. 浮点数字 530
6.4.1.3.5. 数值类型的优先级 532
6.4.1.4. 日期时间和间隔数据类型 532
6.4.1.4.1. 日期时间和间隔数据类型概述 532
6.4.1.4.2. DATE 数据类型 533
6.4.1.4.3. TIMESTAMP 数据类型 535
6.4.1.4.4. TIMESTAMP WITH TIME ZONE 数据类型 536
6.4.1.4.5. TIMESTAMP WITH LOCAL TIME ZONE 数据类型 538

6.4.1.4.6. INTERVAL YEAR TO MONTH 数据类型 540
6.4.1.4.7. INTERVAL DAY TO SECOND 数据类型 542
6.4.1.4.8. 日期时间 DATE 和间隔 INTERVAL 的计算 544
6.4.1.5. RAW 数据类型 546
6.4.1.6. 大对象数据类型 548
6.4.1.6.1. 大对象数据类型概述 548
6.4.1.6.2. BLOB 数据类型 549
6.4.1.6.3. CLOB 数据类型 549
6.4.2. 数据类型比较规则 550
6.4.2.1. 数据类型比较规则概述 550
6.4.2.2. 数值 550
6.4.2.3. 日期值 550
6.4.2.4. 字符值 550
6.4.2.5. 数据类型优先级 552
6.4.2.6. 数据类型转换 552
6.4.2.7. 数据转换的安全注意事项 557
6.4.3. 字面量 557
6.4.3.1. 字面量概述 557
6.4.3.2. 文本字面量 557
6.4.3.3. 数值字面量 558
6.4.3.4. 日期时间字面量 559
6.4.3.4.1. 日期字面量 559
6.4.3.4.2. 时间戳字面量 561
6.4.3.5. 间隔字面量 562
6.4.4. 格式化 565
6.4.4.1. 格式化概述 565
6.4.4.2. 数值格式化 565
6.4.4.3. 日期时间格式化 568

6.4.4.4. RR 日期时间格式化元素 573
6.4.4.5. 字符串到日期的转换规则 575
6.4.5. 空值 575
6.4.5.1. 空值概述 575
6.4.5.2. SQL 函数中的空值 575
6.4.5.3. 比较条件中的空值 577
6.4.5.4. 条件判断表达式中的空值 578
6.4.6. 注释 578
6.4.6.1. 注释概述 578
6.4.6.2. SQL 语句的注释 578
6.4.6.3. Schema 与非 Schema 对象的注释 580
6.4.6.4. Hint 580
6.4.6.4.1. Hint 概述 580
6.4.6.4.2. Hint 清单 582
6.4.6.4.2.1. 与访问路径相关的 Hint 582
6.4.6.4.2.2. 与联接顺序相关的 Hint 583
6.4.6.4.2.3. 与联接操作相关的 Hint 584
6.4.6.4.2.4. 与并行执行相关的 Hint 586
6.4.6.4.2.5. 与查询策略相关的 Hint 591
6.4.6.4.2.6. 与查询转换相关的 Hint 594
6.4.6.4.2.7. 其他 Hint 598
6.4.7. 数据库对象 602
6.4.7.1. Schema 对象 602
6.4.8. 数据库命名规范 602
6.4.8.1. 数据库对象命名规范概述 602
6.4.8.2. 数据库对象命名规则 603
6.4.8.3. Schema 对象命名示例 605
6.4.8.4. Schema 对象命名准则 605

6.4.9. 数据库对象引用方式 605
6.4.9.1. 数据库对象引用概述 605
6.4.9.2. 引用 Schema 对象 606
6.4.9.3. 引用远程数据库中的对象 607
6.4.9.3.1. 创建数据库链接 607
6.4.9.3.2. 引用数据库链接 607
6.4.9.4. 引用分区表和索引 608
6.4.9.5. 引用对象类型的属性和方法 609
6.5. 运算符 611
6.5.1. 运算符概述 611
6.5.2. 算术运算符 612
6.5.3. 串联运算符 613
6.5.4. 层次查询运算符 613
6.5.5. 集合运算符 614
6.5.6. 排序规则 614
6.6. 函数 615
6.6.1. 函数概述 615
6.6.2. 单行函数 634
6.6.2.1. 数字函数 634
6.6.2.1.1. ABS 634
6.6.2.1.2. ACOS 635
6.6.2.1.3. BITAND 636
6.6.2.1.4. CEIL 636
6.6.2.1.5. EXP 637
6.6.2.1.6. FLOOR 638
6.6.2.1.7. LN 639
6.6.2.1.8. LOG 640
6.6.2.1.9. MOD 641

6.6.2.1.10. POWER 642
6.6.2.1.11. REMAINDER 643
6.6.2.1.12. ROUND 644
6.6.2.1.13. SIGN 645
6.6.2.1.14. SQRT 645
6.6.2.1.15. TRUNC 646
6.6.2.2. 返回字符串的字符串函数 647
6.6.2.2.1. CHR 647
6.6.2.2.2. CONCAT 649
6.6.2.2.3. INITCAP 650
6.6.2.2.4. LOWER 650
6.6.2.2.5. LPAD 651
6.6.2.2.6. LTRIM 652
6.6.2.2.7. REGEXP_REPLACE 653
6.6.2.2.8. REPLACE 655
6.6.2.2.9. RPAD 656
6.6.2.2.10. RTRIM 657
6.6.2.2.11. SUBSTR 658
6.6.2.2.12. TRANSLATE 659
6.6.2.2.13. TRIM 660
6.6.2.2.14. UPPER 661
6.6.2.3. 返回数字的字符串函数 662
6.6.2.3.1. ASCII 662
6.6.2.3.2. INSTR 663
6.6.2.3.3. LENGTH 665
6.6.2.4. 时间日期函数 666
6.6.2.4.1. ADD_MONTHS 666
6.6.2.4.2. CURRENT_DATE 667

6.6.2.4.3. CURRENT_TIMESTAMP 668
6.6.2.4.4. DBTIMEZONE 669
6.6.2.4.5. EXTRACT (datetime) 670
6.6.2.4.6. FROM_TZ 671
6.6.2.4.7. LAST_DAY 672
6.6.2.4.8. LOCALTIMESTAMP 673
6.6.2.4.9. MONTHS_BETWEEN 674
6.6.2.4.10. NEXT_DAY 675
6.6.2.4.11. NUMTODSINTERVAL 676
6.6.2.4.12. NUMTOYMINTERVAL 677
6.6.2.4.13. ROUND (date) 678
6.6.2.4.14. SESSIONTIMEZONE 680
6.6.2.4.15. SYS_EXTRACT_UTC 681
6.6.2.4.16. SYSDATE 682
6.6.2.4.17. SYSTIMESTAMP 682
6.6.2.4.18. TO_CHAR (datetime) 683
6.6.2.4.19. TO_DSINTERVAL 685
6.6.2.4.20. TO_TIMESTAMP 685
6.6.2.4.21. TO_TIMESTAMP_TZ 686
6.6.2.4.22. TO_YMINTERVAL 687
6.6.2.4.23. TRUNC (date) 688
6.6.2.4.24. TZ_OFFSET 690
6.6.2.5. 通用比较函数 691
6.6.2.5.1. GREATEST 691
6.6.2.5.2. LEAST 692
6.6.2.6. 转换函数 693
6.6.2.6.1. CAST 693
6.6.2.6.2. HEXTORAW 695

6.6.2.6.3. RAWTOHEX 696
6.6.2.6.4. TO_BINARY_DOUBLE 697
6.6.2.6.5. TO_BINARY_FLOAT 697
6.6.2.6.6. TO_CHAR (character) 698
6.6.2.6.7. TO_CHAR (datetime) 699
6.6.2.6.8. TO_CHAR (number) 701
6.6.2.6.9. TO_DATE 702
6.6.2.6.10. TO_DSINTERVAL 703
6.6.2.6.11. TO_NUMBER 704
6.6.2.6.12. TO_TIMESTAMP 705
6.6.2.6.13. TO_TIMESTAMP_TZ 706
6.6.2.6.14. TO_YMINTERVAL 707
6.6.2.7. 编码解码函数 707
6.6.2.7.1. DECODE 707
6.6.2.7.2. ORA_HASH 710
6.6.2.7.3. VSIZE 712
6.6.2.8. 空值相关函数 713
6.6.2.8.1. COALESCE 713
6.6.2.8.2. LNNVL 714
6.6.2.8.3. NVL 716
6.6.2.8.4. NVL2 718
6.6.3. 聚合函数 719
6.6.3.1. AVG 719
6.6.3.2. COUNT 722
6.6.3.3. SUM 725
6.6.3.4. MAX 727
6.6.3.5. MIN 729
6.6.3.6. LISTAGG 731

6.6.3.7. ROLLUP 734
6.6.3.8. STDDEV 737
6.6.3.9. STDDEV_POP 739
6.6.3.10. STDDEV_SAMP 741
6.6.3.11. VARIANCE 743
6.6.3.12. APPROX_COUNT_DISTINCT 746
6.6.4. 分析函数 747
6.6.4.1. 窗口函数说明 747
6.6.4.2. AVG 751
6.6.4.3. COUNT 754
6.6.4.4. SUM 757
6.6.4.5. MAX 759
6.6.4.6. MIN 761
6.6.4.7. LISTAGG 763
6.6.4.8. STDDEV 766
6.6.4.9. STDDEV_POP 768
6.6.4.10. STDDEV_SAMP 770
6.6.4.11. VARIANCE 772
6.6.4.12. RANK 774
6.6.4.13. LEAD 776
6.6.4.14. LAG 778
6.6.4.15. FIRST_VALUE 780
6.6.4.16. LAST_VALUE 782
6.6.4.17. NTH_VALUE 784
6.6.4.18. CUME_DIST 786
6.6.4.19. DENSE_RANK 788
6.6.4.20. NTILE 789
6.6.4.21. PERCENT_RANK 791

6.6.4.22. RATIO_TO_REPORT 792
6.6.4.23. ROW_NUMBER 793
6.7. 表达式 795
6.7.1. SQL 表达式概述 795
6.7.2. 简单表达式 796
6.7.3. 复合表达式 796
6.7.4. 条件表达式 796
6.7.5. 列表达式 798
6.7.6. 日期时间表达式 798
6.7.7. 函数表达式 799
6.7.8. 间隔表达式 800
6.7.9. 标量子查询表达式 800
6.7.10. 表达式列表 800
6.8. 条件 802
6.8.1. SQL 条件概述 802
6.8.2. 比较条件 804
6.8.3. 逻辑条件 804
6.8.4. 模式匹配条件 805
6.8.5. 空条件 807
6.8.6. 复合条件 808
6.8.7. BETWEEN 条件 808
6.8.8. 存在条件 808
6.8.9. IN 条件 809
6.9. 查询和子查询 809
6.9.1. 查询和子查询概述 810
6.9.2. 简单查询 811
6.9.3. 层次查询 814
6.9.4. 集合 817

6.9.5. 连接 823
6.9.6. 子查询 829
6.10. SQL语句 832
6.10.1. DDL 832
6.10.1.1. ALTER KEYSTORE 833
6.10.1.2. ALTER OUTLINE 833
6.10.1.3. ALTER SEQUENCE 834
6.10.1.4. ALTER SESSION 836
6.10.1.5. ALTER TABLE 837
6.10.1.6. ALTER TABLEGROUP 842
6.10.1.7. ALTER USER 844
6.10.1.8. CREATE INDEX 845
6.10.1.9. CREATE KEYSTORE 847
6.10.1.10. CREATE OUTLINE 848
6.10.1.11. CREATE SEQUENCE 850
6.10.1.12. CREATE SYNONYM 851
6.10.1.13. CREATE TABLE 854
6.10.1.14. CREATE TABLEGROUP 860
6.10.1.15. CREATE TABLESPACE 862
6.10.1.16. CREATE USER 863
6.10.1.17. CREATE VIEW 864
6.10.1.18. DROP INDEX 865
6.10.1.19. DROP OUTLINE 866
6.10.1.20. DROP SEQUENCE 867
6.10.1.21. DROP SYNONYM 867
6.10.1.22. DROP TABLE 868
6.10.1.23. DROP TABLEGROUP 869
6.10.1.24. DROP TABLESPACE 869

6.10.1.25. DROP USER 870
6.10.1.26. DROP VIEW 870
6.10.1.27. RENAME 871
6.10.1.28. TRUNCATE TABLE 872
6.10.2. DML 872
6.10.2.1. DELETE 872
6.10.2.2. INSERT 875
6.10.2.3. MERGE 878
6.10.2.4. PURGE DATABASE 880
6.10.2.5. PURGE INDEX 881
6.10.2.6. PURGE RECYCLEBIN 882
6.10.2.7. PURGE TABLE 883
6.10.2.8. SELECT 883
6.10.2.9. UPDATE 892
6.10.3. DCL 895
6.10.3.1. EXPLAIN 895
6.10.3.2. FLASHBACK TABLE BEFORE DROP 901
6.10.3.3. GRANT 903
6.10.3.4. KILL 913
6.10.3.5. REVOKE 914
6.10.3.6. SAVEPOINT 924
6.10.3.7. SET NAMES 927
6.10.3.8. SET PASSWORD 928
6.10.3.9. SET VARIABLE 929
6.10.3.10. SHOW 930
6.10.3.11. SHOW RECYCLEBIN 930
6.10.3.12. SHRINK 931
6.10.3.13. TRANSACTION 932

1.什么是云数据库OceanBase

云数据库OceanBase是一款高性能、分布式的金融级关系型数据库，具备多地多活、异地容灾的高可用性，支持自由扩展以应对日益增长的业务需求。

云数据库OceanBase是一款高性能、分布式的金融级关系型数据库，当用户的业务飞速发展的时候，OceanBase总是可以自由扩展，提供低延迟、高吞吐的数据库服务，让用户有更好的体验。例如2017年的双十一，OceanBase承担了100%的交易流和支付流量，支付峰值25.6万笔／秒，数据库处理峰值4200万次／秒，为互联网金融的蓬勃发展做出了巨大贡献。

OceanBase通过在底层分布式引擎实现的Paxos多数派协议和多副本特性，拥有令人称道的高可用和容灾能力，不负永不停机的数据库系统的盛名 ，可以完美支持多地多活、异地容灾等高可用部署，实现跨机房、跨地域、甚至跨洲际部署，实现金融级可用性和事务的强一致性。

OceanBase是一个准内存数据库系统，独有的读写分离架构和面向SSD 固态盘的高效存储引擎，为用户带来了超高性能的体验 。OceanBase定位为云数据库，通过在数据库内部实现多租户隔离，实现一个集群服务多个租户，且租户之间完全隔离，不会相互影响。

OceanBase目前兼容MySQL5.6版本大部分功能，基于MySQL的业务只需零修改或少量修改即可迁移到OceanBase，提高了应用开发和迁移的效率。同时，OceanBase在数据库内部实现了分区表和二级功能，可以完全取代 MySQL常用的分库分表方案。同时，OceanBase控制台还提供实例升降级、性能数据查看、优化建议等功能，让用户可以轻松管理复杂的数据库。

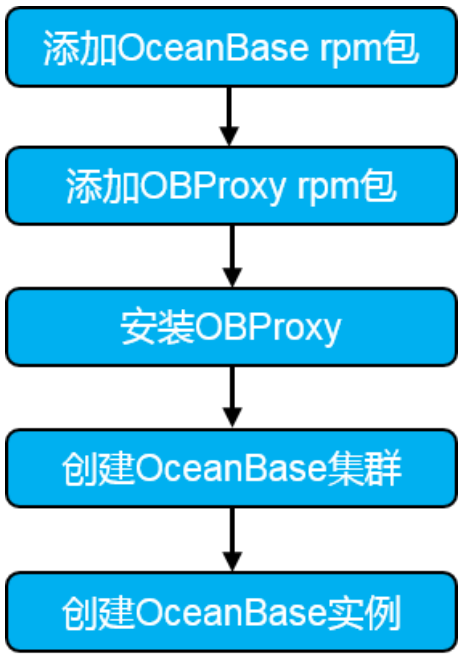
2.快速入门

2.1. 概述

本章节介绍如何快速登录OceanBase控制台、添加OceanBase和OBProxy rpm包、安装OBProxy、创建集群及创建实例，旨在引导您一站式完成实例使用前的准备工作。

具体操作步骤请您参见OceanBase使用流程图。

OceanBase使用流程图



各步骤说明如下：

1. 添加OceanBase rpm包

   OceanBase支持通过rpm包和源码方式安装，OceanBase控制台提供rpm包管理的功能。

2. 添加OBProxy rpm包

   添加当前OceanBase对应的OBProxy rpm包。

3. 安装OBProxy

   OBProxy作为OceanBase分布式关系数据库专用的反向代理服务器，具有防连接闪断、屏蔽后端异常(宕机、升级、网络抖动)、MySQL协议兼容、强校验、支持热升级和多集群等功能，为前端用户请求提供高性能、高准确率的路由转发服务，为后端Server服务提供高可用易扩展的容灾保障。

4. 创建集群

   创建OceanBase集群，并配置相关参数。

5. 创建实例

   创建OceanBase实例，并配置相关参数。

2.2. 登录OceanBase运维控制台

本节以Chrome浏览器为例，介绍如何通过Apsara Uni-manager运维系统进入OceanBase运维控制台。

前提条件

- 已从部署人员或管理员处获取Apsara Uni-manager运维控制台的访问地址、用户名和密码。

  Apsara Uni-manager运维控制台访问地址格式为*region-id*.*ops*.asconsole.*intranet-domain-id*.com。
- 推荐使用Chrome浏览器。

操作步骤

1. 在地址栏中，输入ASO的访问地址*region-id*.aso.*intranet-domain-id*.com，按回车键。

   > **说明** 您可以单击页面右上角的下拉按钮来进行语言切换。

2. 输入正确的用户名及密码。

   > **说明** 登录ASO的用户名和密码请从部署人员或管理员处获取。

   首次登录ASO时，需要修改登录用户名的密码，请按照提示完成密码修改。

   为提高安全性，密码必须满足以下要求：

   - 英文大小写字母
   - 阿拉伯数字（0~9）
   - 特殊符号，包括感叹号（！）、at（@）、井号（#）、美元符号（$）、百分号（%）等
   - 密码长度10~20个字符
3. 单击**登录**，进入**ASO**页面。
4. 在页面右侧的导航栏中，单击**产品运维管理**，选择**产品列表 > 数据库服务**，然后单击**OB云平台**跳转至OceanBase运维控制台。

2.3. 登录OceanBase用户控制台

本节以Chrome浏览器为例，介绍如何通过Apsara Uni-manager运营控制台并进入OceanBase用户控制台。

前提条件

- 登录Apsara Uni-manager运营控制台前，确认您已从部署人员处获取Apsara Uni-manager运营控制台的服务域名地址。
- 推荐使用Chrome浏览器。

背景信息

通过Apsara Uni-manager运营控制台进入OceanBase用户控制台的用户只有查看监控和管理实例的权限，集群运维操作请登录OceanBase运维控制台进行操作。

操作步骤

1. 在浏览器地址栏中，输入Apsara Uni-manager运营控制台的访问地址，按回车键。
2. 输入正确的用户名及密码。

   请向运营管理员获取登录控制台的用户名和密码。

   > **说明** 首次登录Apsara Uni-manager运营控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码长度必须为 8~20 位，且至少包含以下两种类型：
   > - 英文大写或小写字母（A~Z、a~z）
   > - 阿拉伯数字（0~9）
   > - 特殊符号（感叹号（！）、at（@）、井号（#）、美元符号（$）、百分号（%）等）

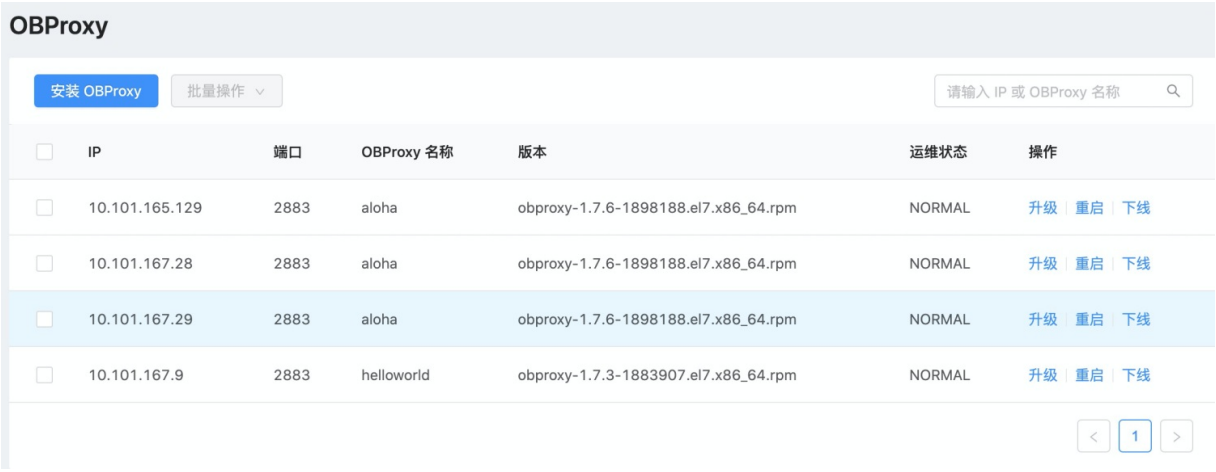
3. 单击**登录**。
4. 在页面上方的导航栏中，单击**产品**，选择**数据库 > 云数据库OceanBase**。
5. 选择**组织**和**地域**，然后单击**OB云平台**跳转至OceanBase用户控制台。

2.4. 安装 OBProxy

在界面右上角单机返回旧版，即可进入 OBProxy 界面进行安装。

操作步骤

1. 单击导航栏的**运维**，进入 **OBProxy** 界面。
2. 在 OBProxy 页面查看 IP、端口、OBProxy 名称、版本、运维状态、操作等信息：

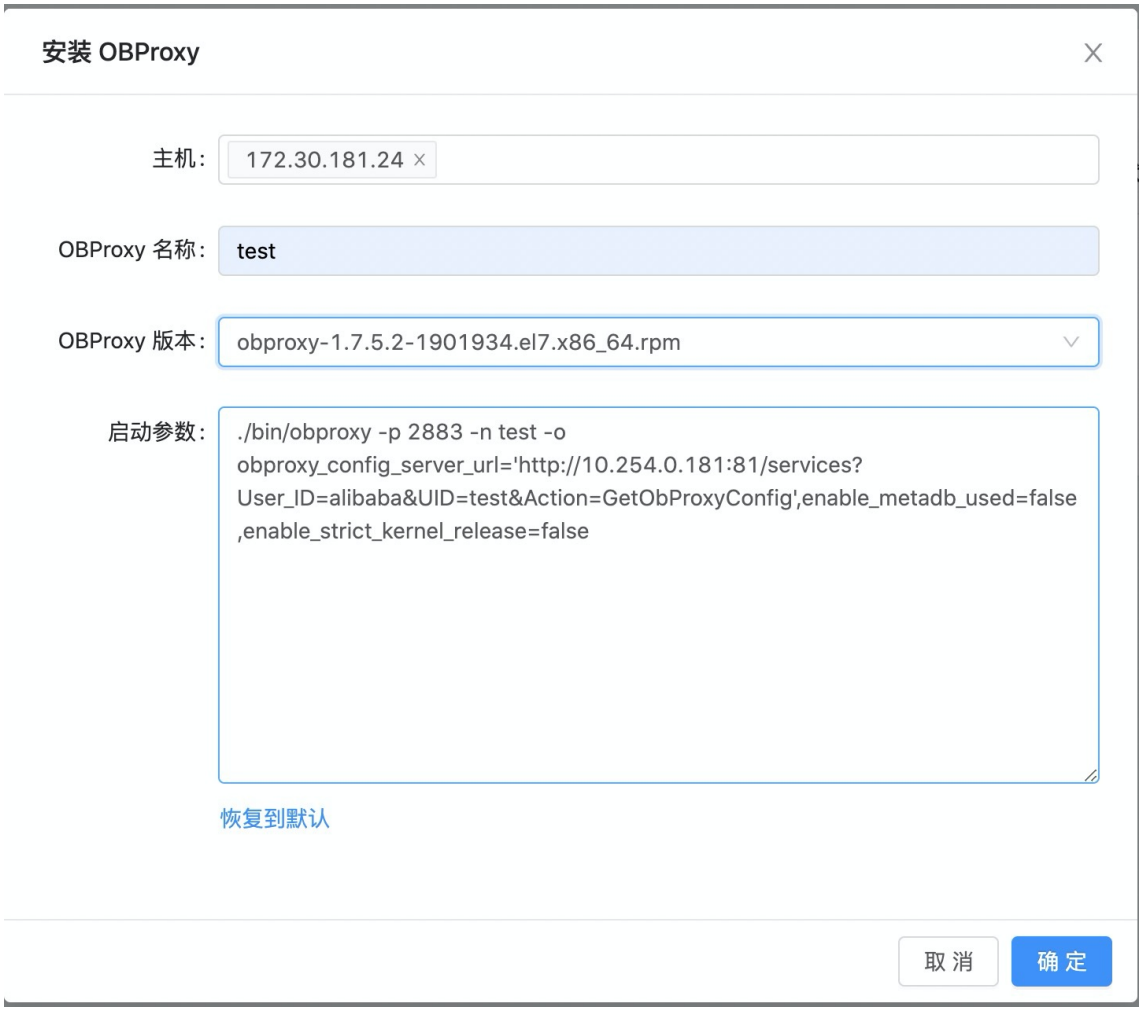


字段	说明
运维状态	• OPERATING 状态是正在执行运维操作； • NORMAL 状态时可以发起运维操作。
操作	• 单击操作列表下**升级**后详细操作参见：升级 OBProxy； • 单击操作列表下**重启**后详细操作参见：重启 OBProxy； • 单击操作列表下**下线**后详细操作参见：下线 OBProxy。

3. 单击**安装 OBProxy**。
4. 在弹出的对话框中，填写如下信息：
   - **主机**：可在下拉框中选择主机，或者单击**添加主机**添加新的机器。



   - **OBProxy 版本**：在下拉框中选择已有的 OBProxy 版本，或者单击**添加版本**在弹出框中**点击上传** 1.7.3 及以上版本的 OBProxy 文件。



- **启动参数**：参考启动参数说明，对启动参数进行设置，参数分为如下两种：
  a. 第一类是更新后可以直接生效；
  b. 第二类是更新后需要重启OBProxy才能生效。

5. 单击**确定**，会生成一个 OBProxy 运维任务，可在**系统管理>任务**中，查看安装进展。

2.5. 新建集群

您可以根据业务需要新建待管理的集群。

前提条件

当前登录用户必须具备创建集群的权限。

操作步骤

1. 登录 OCP 后，根据实际业务场景，找到新建集群的入口。
   - 如果您没有可管理的集群，系统会在**集群概览**页面提示您新建集群，直接在提示信息中单击**创建集群**。
   - 如果您已经有可管理的集群，则在**集群概览**页面右上角，单击**新建集群**。
2. 在**新建集群**页面，设置集群的基础信息。



基础信息相关说明如下表所示。

配置	描述
集群类型	可选择**主集群**或**备集群**。如果选择**备集群**，则需要确认已创建主集群。
集群名称	自定义待管理的集群的名称。集群名称必须以英文字母开头，可支持大小写字母、数字和下划线，长度为2~48字符。
root@sys密码	支持自定义或随机生成。 密码需要满足以下复杂度条件： • 长度：8位 ~ 32位 • 至少包含 2 个数字、2 个大写字母、2 个小写字母和 2 个特殊字符。 支持的特殊字符如下： ._+@#$%)
OB 版本	可以从列表中选择已有的 OB 版本，也可以在列表下方单击**添加版本**，上传一个OB的版本。

3. 设置集群的部署模式信息。

默认是添加 3 个 Zone 的信息，如果您希望部署的集群 Zone 的个数大于 3 个，您可以在下方单击**新增**按钮，增加 Zone 信息。

如果部署的集群 Zone 的个数小于 3 个，您可以单击 Zone 后面的删除图标。



每个 Zone 需要设置的信息及其说明如下表所示。

配置	描述
Zone 名称	一般会有一个默认名称，您可以根据需要自定义名称。 zone 名称必须以英文字母开头，可支持大小写字母、数字和下划线，长度为 2~字符。
机房	Zone 所在的机房，每个 Zone 只能部署在同一个机房。
机型	可选项。 如果选择了机型，后面主机列表会根据机型进行过滤。
机器选择方式	可以选择**自动分配** 或者**手动选择**。
IP	您可以选择多个 IP。 如果**机器选择方式**是**自动分配**，则只需要输入机器的数量，OCP 会自动选择相应数量的可用机器；如果 **机器选择方式** 是 **手动选择**，则需要您手动从列表中选择若干个 IP。
Root Server位置	您可以选择一个 IP 作为 root Server 所在的机器。

配置	描述
优先级排序	Zone的优先级排序。该优先级顺序影响 sys 租户的 primary Zone 的优先级顺序。 左边的列表框中显示了当前集群的所有 Zone。 您可以在左侧列表框中选择一个或多个 Zone 添加到右侧的列表框中，默认先选择的 Zone 的优先级高于后选择的 Zone；一次选中的多个 Zone 的优先级相同。 移动到右侧的列表框中后，您也可以在右侧的列表框中通过拖拽调整顺序，列表框上方的 Zone 的优先级高于下方的 Zone。

4. 完成后，单击**提交**。
5. 在弹出的**确认提交信息**对话框中，确认信息无误后，单击**确定**。

2.6. 新建租户

租户（Tenant）既是各类数据库对象的容器，又是资源（CPU、Memory、IO等）的容器。您可以根据业务需求在集群中创建租户。

新建租户有以下两种方式：

- 在租户概览页面创建租户。
- 在指定集群的租户管理页面创建租户。

本文档为您展示指定集群的租户管理页面创建租户的操作步骤。

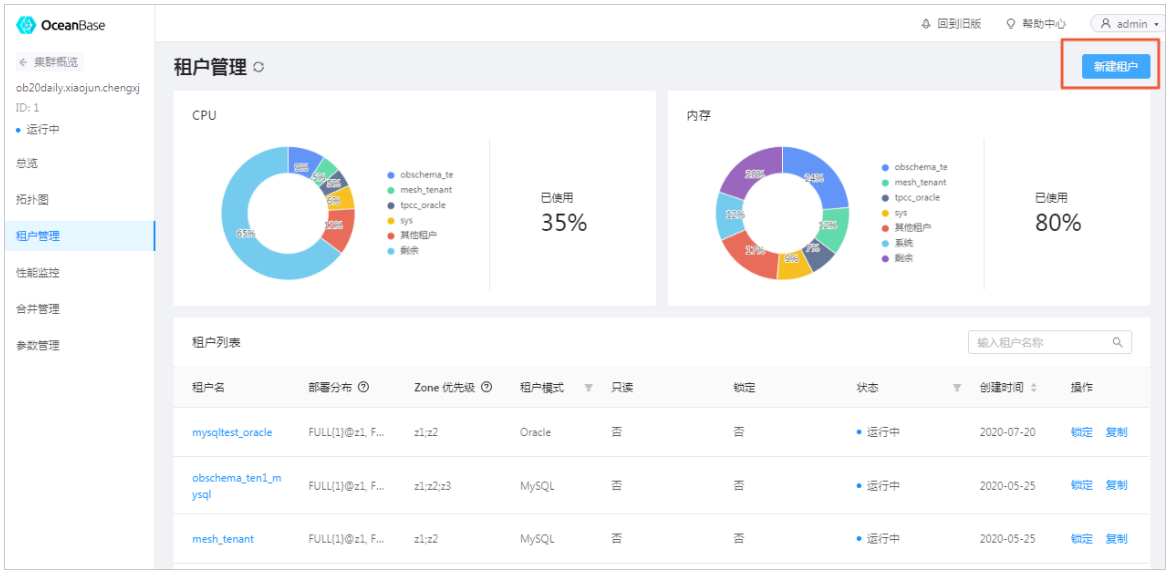
前提条件

- 需要创建租户的集群为主集群，且状态为运行中。
- 当前登录用户具备系统管理员角色、OCP 租户管理员角色或其他拥有管理该集群权限的角色。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**租户管理**。

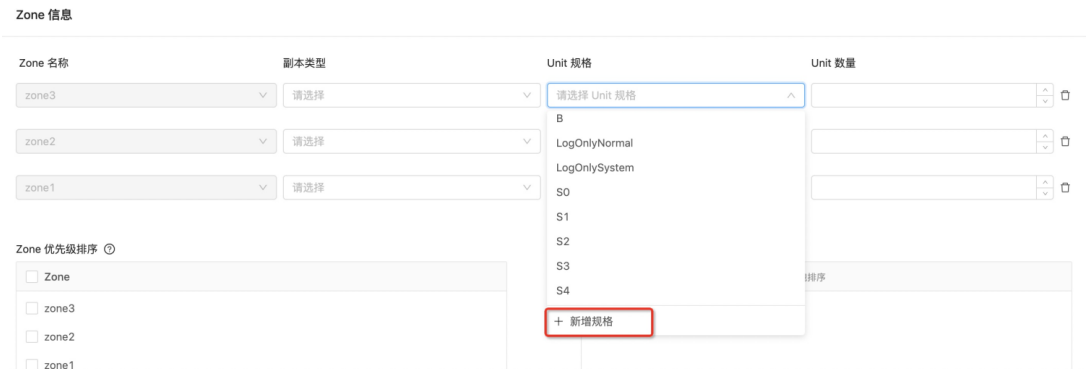
4. 在页面右上角单击**新建租户**。



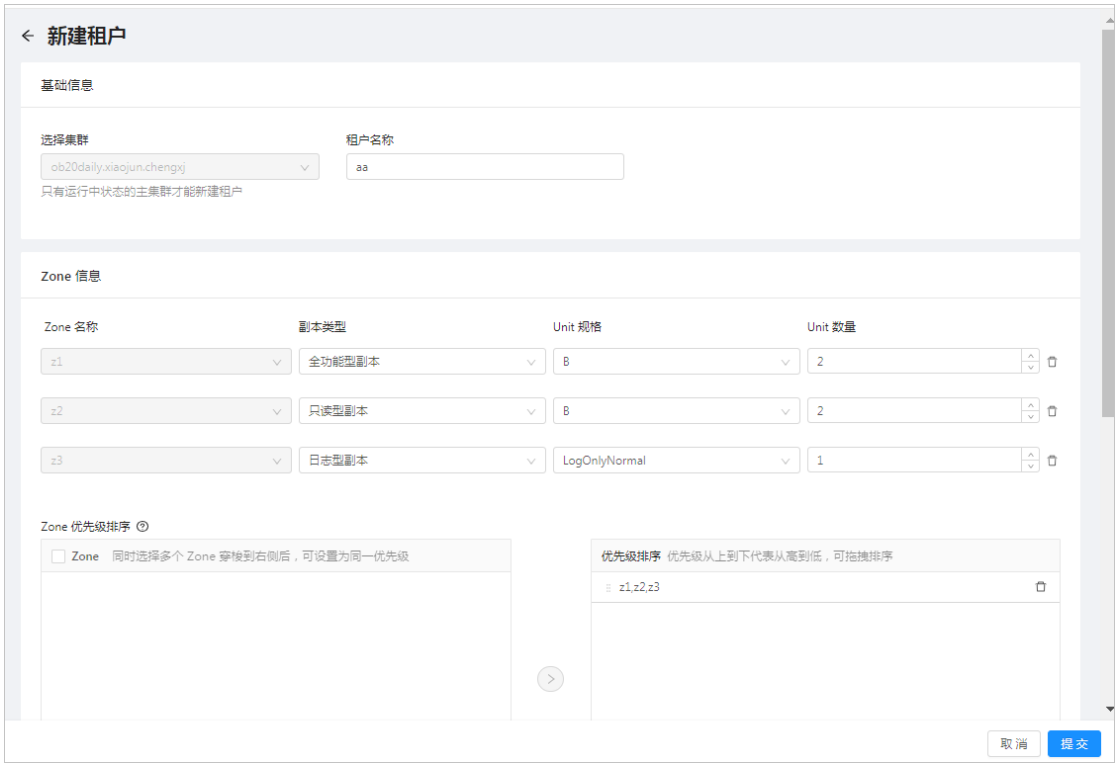
5. 填写**基础信息**。
   i. 集群默认为当前集群。
   ii. 请输入**租户名称**。租户名称格式为英文大小写字母、数字和下划线的组合，长度为 2~64字符。
6. 填写 **Zone 信息**。
   i. 为 z1、z2 和 z3 设置副本类型、Unit 规格和 Unit 数量。

   集群选定后，页面会根据所选集群的 zone 信息给出可配置的 zone 列表；对于无需做副本分布的 zone，可以通过最右侧的按钮删除该zone条目。

   - 副本类型支持全功能型副本、只读型副本和日志型副本。
   - OCP 内置了一套 Unit 规格；另外也可以在下拉列表的最下方点击按钮新增自定义规格。
   - 指定该 zone下的 Unit 数量。需要注意的是，unit 数量不能超过该 zone下 server个数。



ii. 对 **Zone 优先级** 进行排序。同时选择多个 Zone 穿梭到右侧后，可设置为同一优先级。



7. 填写**基本设置**。

i. 设置

- 对于 MySQL 模式，其管理员账户为"root"。
- 对于 Oracle 模式，其管理员账户为“SYS”。密码格式：长度为8 ~ 32，且至少包含 2 个数字、2 个大写字母、2 个小写字母和 2 个特殊字符，支持的特殊字符为 ._+@#$%。

ii. 设置**租户模式**。

- 支持Oracle 和 MySQL 租户模式。
- 所选集群的 OB 版本在 2.1 及以上时才支持 Oracle 租户模式。

iii. 设置**字符集与编码**。

- 对于 MySQL 模式，可选字符集有：utf8mb4, binary, gbk, gb18030。缺省为utf8mb4。
- 对于 Oracle 模式，可选字符集有：utf8mb4, gbk, gb18030。缺省为utf8mb4。

iv. （可选）设置**备注**。

v. 设置 **IP 地址白名单**。

- 此处可以指定该租户允许登录的客户端列表。如果不指定，默认配置为"%"，允许所有的客户端。自定义白名单列表时需要注意，OCP 机器地址以及所依赖的 OBProxy 地址必须在此名单中，否则 OCP 将无法管理此租户。
- 默认配置：所有 IP 都可访问。
- 自定义：设置 IP 白名单，白名单中的IP才能访问。

- 白名单格式说明：
  - IP地址，示例：10.10.10.10,10.10.10.11

- 子网/掩码，示例：10.10.10.0/24
- 模糊匹配，示例：10.10.10.% 或 10.10.10._
- 多种格式混合，示例：10.10.10.10,10.10.10.11,10.10.10.%,10.10.10._,10.10.10.0/24

> **说明**
>
> % 表示所有客户端都可以连接



8. 单击**提交**。

3.OB 云平台

3.1. 集群管理

3.1.1. 概述

OCP 提供了 OceanBase 集群的全生命周期管理功能，包括集群的安装、运维、性能监控、配置、升级和删除等功能。

3.1.2. 集群概览

本页面为您介绍集群概览页面可查看到的信息。

登录 OCP 后即进入**集群概览**页面。当前登录用户需要具备集群的管理权限或只读权限，才能查看集群概览数据。



集群概览页面展示了当前登录用户下的集群概览信息，包括**性能指标 Top5**、**集群合并时间 Top5**、**机器资源**、**资源水位**及**集群列表**等信息。

性能指标 Top5

性能指标 Top5区域展示了某个时间段某个性能指标的平均值从大到小排序最大的5个集群。

您可以根据业务需要，选择待查看的性能指标和对应的时间范围。

可选择的性能指标如下：

- QPS（次/s）
- 查询响应时间（us）
- 活跃会话数（个）
- CPU 使用率（%）

可选择的时间范围如下：

- 最近一小时
- 最近一天
- 最近一周

集群合并时间 Top5

集群合并时间 Top5 区域以柱状图的形式展示了最近三天内合并时间平均值从大到小排序最大的5个集群。合并操作（Major freeze）是将动静态数据做归并，会比较费时。当转储产生的增量数据积累到一定程度时，通过 Major freeze 实现大版本的合并。

如果最近三天内没有集群合并，则此处为空。

机器资源

机器资源区域按照 IDC 展示了机房里的机器总数和已经部署了 OBServer 的机器数量，可以为集群扩容提供参考。

资源水位

资源水位区域展示了集群的名称、主备角色、租户数量、每个集群总的资源单位（Resource Unit）和已分配的资源单位情况，包括已分配的CPU的百分比、已分配内存的百分比、已使用磁盘空间和总磁盘空间大小。

集群列表

集群列表区域展示了各集群的集群名、集群ID、版本、部署模式（包括每个 Zone 所在的 Region，有多少台 OBServer 等）、集群状态、创建时间和告警统计信息等。

对于主备库集群，默认只显示主集群的信息，单击左侧的 图标可以查看备集群的信息。

单击各集群的集群名称，可以跳转到集群详情页面。

3.1.3. 管理单个集群

3.1.3.1. 查看集群总览

您可以在集群概览中选择一个集群，查看该集群的总览信息，包括该集群的概要信息、资源水位信息、告警概览信息、最近7天系统事件以及 Zone 列表和 OBServer 列表等。

在**集群概览**页面的**集群列表**区域，找到待查看的集群，单击集群名称，进入集群的**总览**页面。



集群概要信息

主要展示当前集群的集群类型、 OB 版本号、Region 数量、Zone 数量、机器数量、分区数、租户数量、集群的创建者以及 ConfigUrl 等信息。

资源水位

主要展示当前集群中可用系统资源的百分比，即 CPU 剩余、内存剩余和磁盘剩余的百分比。

告警概览

主要展示与当前集群相关的所有告警信息，并且这些告警信息按照告警的严重程度进行了分类展示。

目前主要有停服、严重、警告、注意、提醒等 5 类告警。

系统默认显示当前存在的各种告警，您可以根据业务需要，查看最近 24 小时、最近一周或最近 30 天的告警信息。

最近7天的系统事件

主要展示当前集群与总控服务相关的事件详情，并且所有事件均按照事件发生的时间进行了排序。

每一条系统事件展示的信息主要包括该事件的事件名称、所属模块、开始时间、事件详情和事件说明。

对于常见的事件，系统会显示事件详情和事件说明；对于不常见的事件，事件详情和说明会无任何内容。

Zone 列表

主要展示当前集群中的各个 Zone 的基本信息，包括 Zone 名称、所属 Region、所在机房以及 RootServer 地址和状态信息，同时还提供了Zone 级别的运维操作入口，包括添加 OBServer、重启某个 Zone下所有节点的 OBServer 进程、停止某个 Zone下所有节点的 OBServer 进程以及删除某个 Zone 下的所有节点。

OBServer 列表

主要展示当前集群中的各个 OBServer 的基本信息，包括 IP地址、OBServer 使用的端口、所在机房、所属 Zone、机型以及剩余的 CPU、内存和磁盘资源信息等。

此外，OCP 还为当前集群中的每个 OBServer 提供了常见的运维操作入口，包括重启、启动、停止、替换以及删除 OBServer 等，方便您进行基本运维操作。

3.1.3.2. 管理集群

3.1.3.2.1. 新建备集群

如果您有一个主集群，您可以为当前集群创建一个备用集群。

前提条件

请确保当前登录账号具备集群的管理权限。

操作步骤

1. 登录 OCP 。

默认进入**集群概览**页面。

2. 您可以通过以下2种方式进入创建备集群的页面。
   - 在**集群概览**页面的右上角，单击 **新建集群**，然后在**新建集群**页面，选择**集群类型**为**备集群**，并在**主集群**列表中选择对应的主集群。
   - 在**集群概览**页面的**集群列表**区域，选择待添加备集群的集群并单击其集群名，单击**总览**页面的右上角，选择**新建备集群**。
3. 在**新建集群**页面，设置备集群的部署模式。

   默认是添加 3 个 Zone 的信息，如果您希望部署的集群 Zone 的个数大于 3 个，您可以在下方单击**新增**按钮，增加 Zone 信息。

   如果部署的集群 Zone 的个数小于 3 个，您可以单击 Zone 后面的删除图标。



每个 Zone 需要配置的信息及说明如下表所示。

配置	描述
Zone 名称	一般会有一个默认名称，您可以根据需要自定义名称。
机房	Zone 所在的机房，每个 Zone 只能部署在同一个机房。
机型	可选项。 如果选择了机型，后面主机列表会根据机型进行过滤。
机器选择方式	可以选择**自动分配**或者**手动选择**。
IP	您可以选择多个 IP。 如果**机器选择方式** 是**自动分配**，则只需要输入机器的数量，OCP 会自动选择相应数量的可用机器；如果**机器选择方式**是**手动选择**，则需要您手动从列表中选择若干个 IP。
Root Server位置	您可以选择一个 IP 作为 Root Server 所在的机器。

v. 完成后，单击**提交**。

vi. 在弹出的**确认提交信息**对话框中，确认信息无误后，单击**确定**。

3.1.3.2.2. 升级版本

当有新的集群版本时，您可以为您管理的集群升级版本。

前提条件

请确保当前登录账号具备集群的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 单击在**总览**页面的右上角图标，选择**升级版本**。
4. 在**升级版本**对话框中，选择需要升级到的目标版本。



如果需要升级的目标版本还未上传到 OCP 上，您可以在 **升级版本** 列表下方，单击 **添加版本** 按钮，上传对应的软件包。

5. 单击**升级**。
6. 在弹出的**升级路径确认**窗口，单击**确定**。

3.1.3.2.3. 修改密码

您可以根据业务需要，修改您管理的集群的 sys 租户的 root 用户密码。

前提条件

请确保当前登录账号具备集群的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 单击**总览**页面的右上角图标，选择**修改密码**。
4. 在**修改密码**对话框中，输入新密码并确认新密码。

> **说明**
>
> 系统默认已选择 **设为默认连接信息，并保存到密码箱**，方便您下次登录 OCP 时，直接使用新的密码登录到对应的集群。



5. 单击**确定**。

3.1.3.2.4. 重启集群

您可以根据业务需要，重启集群中所有节点的 OBServer 进程或者重启某些 Zone 的 OBServer 进程。

前提条件

请确保当前登录账号具备集群的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 单击**总览**页面的右上角图标，选择**重启集群**。
4. 选择**重启范围**为**全部**、某个 Zone或某几个 Zone，单击**重启**。

如果选择了 **全部**，则系统会重启集群中所有节点的 OBServer 进程；如果选择的是某些 Zone，则系统会重启这些 Zone 的 OBServer 进程。



3.1.3.2.5. 停止集群

您可以根据业务需要停止集群，停止集群将会停止集群中所有节点上运行的 OBServer 进程。

前提条件

请确保当前登录账号具备集群的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的右上角，单击…图标，选择**停止集群**。

   > 注意
   >
   > 停止集群将会停止集群中所有节点上运行的 OBServer 进程，请谨慎操作。

4. 在弹出的**确认**框中，单击**停止**。

3.1.3.2.6. 删除集群

您可以根据业务需要，删除您不需要的OB 集群。

前提条件

请确保当前登录账号具备集群的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待删除的集群并单击其集群名。
3. 在**总览**页面的右上角，单击…图标，选择**删除集群**。

4. 在弹出的**删除集群**对话框中，输入 delete，单击**删除**。



3.1.3.3. 管理 Zone

3.1.3.3.1. 新建 Zone

您可以向您管理的集群中添加新的 Zone。

操作步骤

1. 登录 OCP。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面右上角，单击**新增 Zone**。
4. 在弹出的对话框中，设置 Zone 信息。

默认是添加一个 Zone 信息，如果您需要添加多个 Zone ，可以在下方单击 **新增** 按钮，添加个Zone信息。



需要设置的 Zone 信息如下表所示。

配置	描述
Zone 名称	自定义 Zone 名称。
机房	Zone 所在的机房。
机型	可选项。 如果选择了机型，后面主机列表会根据机型进行过滤。
机器选择方式	可以选择 **自动分配** 或者 **手动选择**。
IP	您可以选择多个 IP。 如果 **机器选择方式** 是 **自动分配**，则只需要输入机器的数量，OCP 会自动选择相应数量的可用机器；如果 **机器选择方式** 是 **手动选择**，则需要您手动从列表中选择若干个 IP。

5. 完成后，单击**确定**。

3.1.3.3.2. 添加 OBServer

您可以向您管理的集群中添加 OBServer 节点。

操作步骤

- 向集群中添加 OBServer节点
  i. 登录 OCP 。默认进入**集群概览**页面。
  ii. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
  iii. 在**总览**页面右上角，单击**添加 OBServer**。
  iv. 在 **OBServer 信息** 区域，分别为每个 Zone 选择机型、机器选择方式及待添加的 OBServer所在的主机 IP 址。



  v. 完成后，单击**确定**。
- 向指定的Zone当中添加 OBServer节点
  i. 登录 OCP 。默认进入**集群概览**页面。
  ii. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
  iii. 在**总览**页面的 **Zone 列表**区域，选择一个需要添加 OBServer 的 Zone，在对应的**操作**列中，单击**添加 OBServer**。

iv. 在 默认添加一个 OBServer，如果需要添加多个 OBServer，您可以在下方单击 **新增** 按钮。



v. 完成后，单击**确定**。

3.1.3.3.3. 重启 Zone

您可以对集群中的 Zone 执行重启操作，重启 Zone 会重启当前 Zone 下所有节点的 OBServer 进程。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的 **Zone 列表**区域，选择一个需要重启的 Zone，在对应的**操作**列中，单击**重启**。

> 注意
>
> 重启Zone会导致当前Zone下所有节点的 OBServer 进程重启，请谨慎操作。

4. 在弹出的确认框中，单击**重启**。

3.1.3.3.4. 停止 Zone

您可以对集群中的 Zone 执行停止 Zone 的操作，停止 Zone 会停止当前 Zone 下所有节点的 OB Server 进程。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的 **Zone 列表**区域，选择待停止的 Zone，在对应的**操作**列中，单击...图标，选择**停止**。
4. 在弹出的确认框中，单击**停止**。

3.1.3.3.5. 删除 Zone

您可以对集群中的 Zone 执行删除 Zone 的操作，删除 Zone 会删除当前 Zone 下的所有节点，同时会删除节点上的数据。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的 **Zone 列表**区域，选择待删除的 Zone，在对应的**操作**列中，单击...图标，选择**删除**。
4. 在弹出的确认框中，单击**删除**。

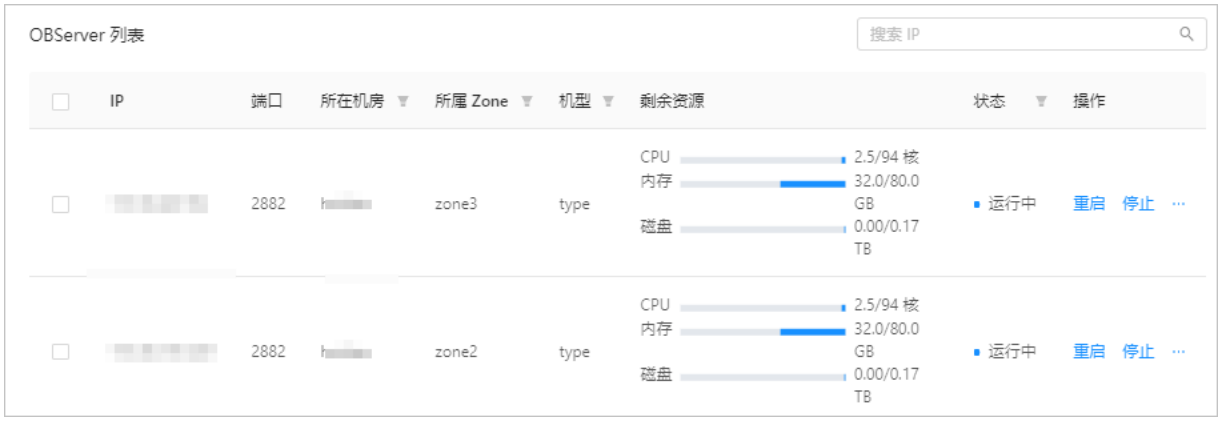
3.1.3.4. 管理 OBServer

3.1.3.4.1. 重启 OBServer

当某个 OBServer 出现故障时，您可以重启该 OBServer。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的**OBServer 列表**区域，找到需要重启 OBServer 的主机，在对应的**操作**列中，单击**重启**。



4. 在弹出的**重启 OBServer** 对话框中，单击**重启**。

3.1.3.4.2. 启动 OBServer

当某个主机上的 OBServer 进程停止运行后，您可以启动该主机上的 OBServer 进程。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的 **OBServer 列表**区域，找到需要启动 OBServer 的主机，在对应的**操作**列中，单击**启动**。
4. 在弹出的**启动 OBServer** 对话框中，单击**启动**。

3.1.3.4.3. 停止 OBServer

您可以根据业务需要，停止某个主机上的 OBServer 进程。

操作步骤

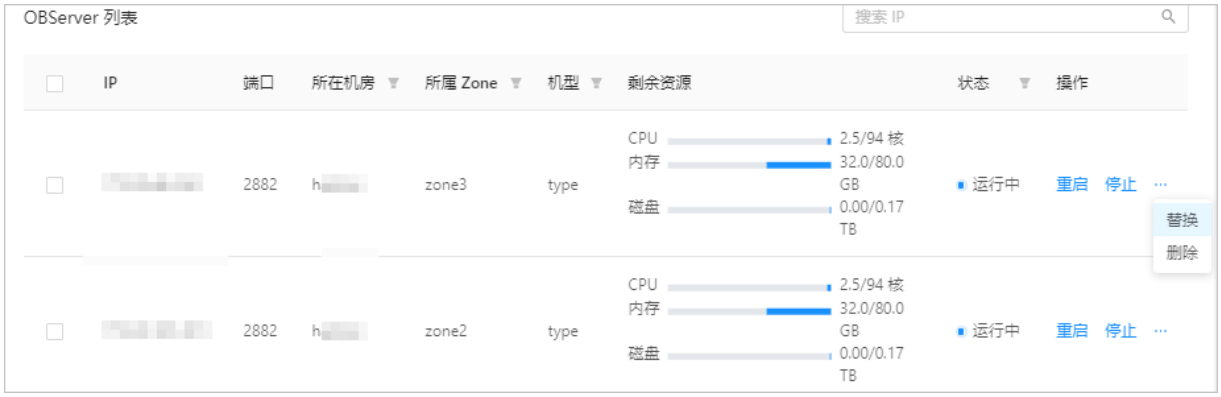
1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的 **OBServer 列表**区域，找到需要停止 OBServer 的主机，在对应的**操作**列中，单击**停止**。
4. 在弹出的**停止 OBServer** 对话框中，单击**停止**。

3.1.3.4.4. 替换 OBServer

您可以根据业务需要，替换某个主机上的 OBServer。执行替换操作后，系统会将当前主机上的 OBServer 从集群中删除，然后在目标主机上安装 OBServer 软件，并将其添加到集群。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的 **OBServer 列表**区域，选择一个需要替换 OBServer 的主机，在对应的**操作** 列中，单击 ··· 图标，选择**替换**。



4. 在弹出的**替换 OBServer** 对话框中，选择待替换的目标主机，单击**替换**。

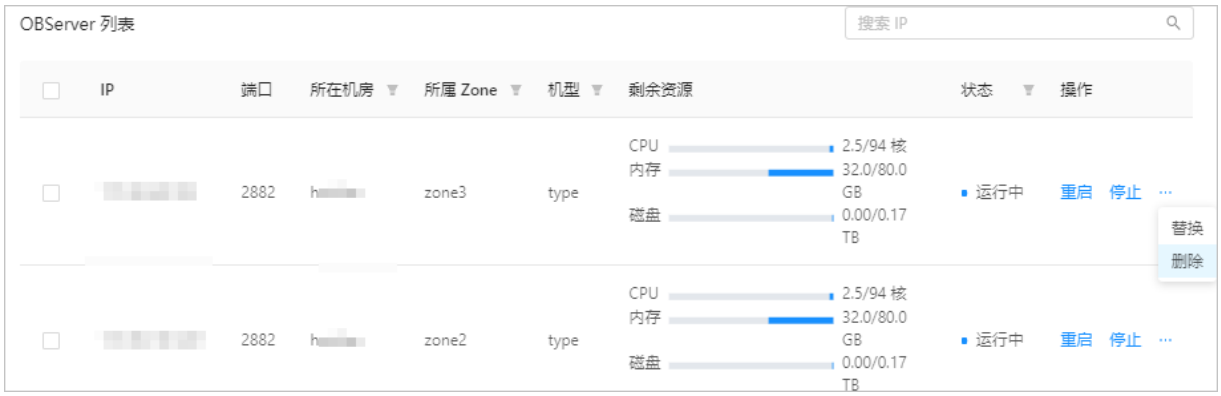
3.1.3.4.5. 删除 OBServer

您可以根据业务需要，删除某个主机上的 OBServer 进程。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在**总览**页面的 **OBServer 列表**区域，找到待删除 OBServer 的主机，在对应的**操作**列中，单击…图标，选择 **删除**。

> **注意**
>
> 删除 OBServer 将会删除 OBServer 的所有数据，且无法恢复，请谨慎操作。

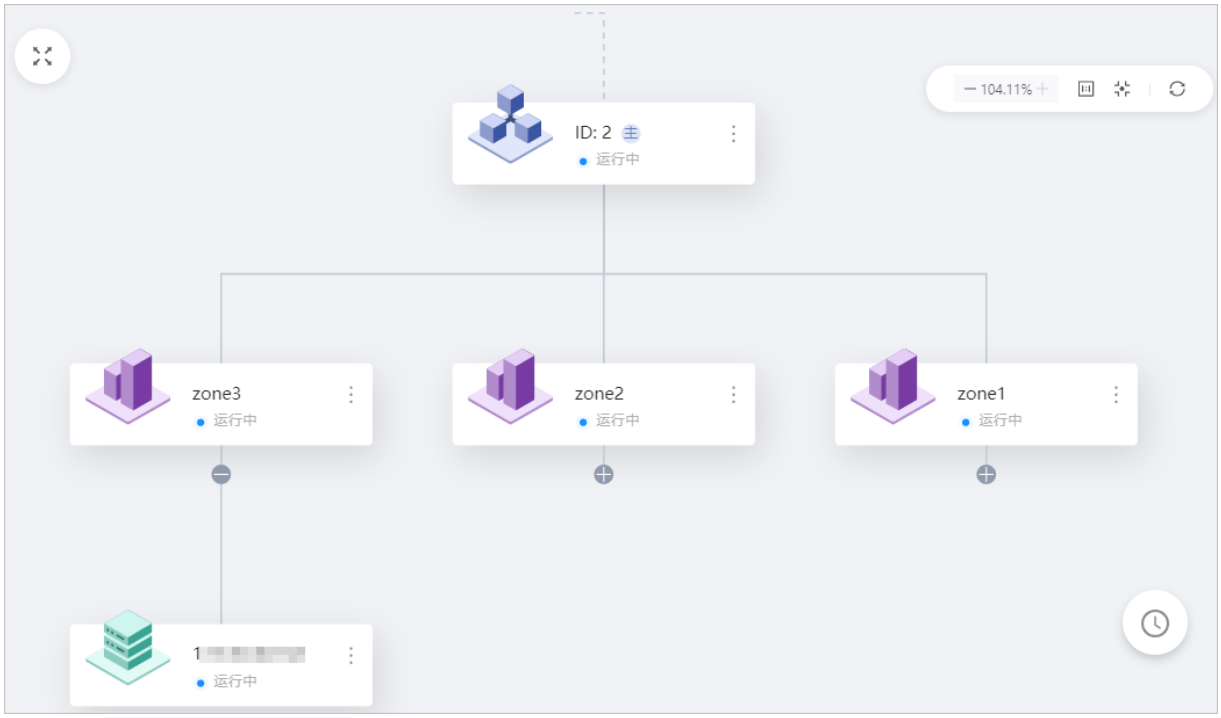


4. 在弹出的**删除 OBServer** 对话框中，选择待替换的目标主机，单击**删除**。

3.1.3.5. 查看集群拓扑图

OCP 提供了全新的拓扑图功能，您可以查看查看集群和租户中各 Zone 与 OBServer 节点之间的逻辑关系。

在集群概览页面的集群列表区域，选择待操作的集群并单击其集群名，再在显示的页面的左侧导航栏上，单击拓扑图，进入集群拓扑图页面。



页面右上角提供了一些功能按钮，您可以通过这些按钮来调整拓扑图的大小或刷新当前拓扑图。

拓扑图从上到下主要分为集群、Zone、Server 三层。

集群

集群图标上展示了 OB 集群的ID、集群类型（主或备）以及集群当前的状态。

备集群图标上方还会展示当前的备库延迟时间。

同时，拓扑图上还提供了集群的管理入口。

对于主集群，您可以对其执行新建备集群、新增 Zone、添加 OBServer、升级版本、修改密码、重启集群、停止集群以及删除集群等操作；对于备集群，您可以对其执行切为主集群、查看集群等操作。

Zone

Zone 图标上展示了 Zone 名称、以及 Zone 当前的状态。

将鼠标悬停在 Zone 图标上，还可以查看当前 Zone下的运行信息，包括QPS、连接数和Unit数。

同时，拓扑图上还提供了 Zone 的管理入口，您可以对 Zone 执行添加 OBServer、重启、停止及删除 Zone 等操作。

Server

Server 图标上展示了 Server 的 IP 地址以及 Server 当前的状态。

将鼠标悬停在 Server 图标上，还可以查看当前 Server 下的运行信息，包括QPS、连接数和SQL端口。

同时，拓扑图上还提供了 Server 的管理入口，您可以对 Server 执行重启、启动、停止、替换、删除等操作。

3.1.3.6. 租户管理

3.1.3.6.1. 查看租户资源分布

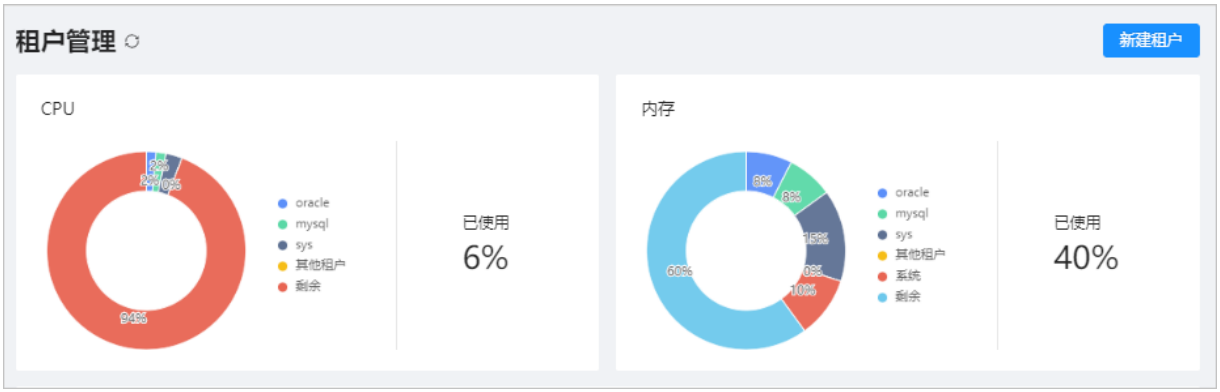
您可以查看您管理的集群中所有租户的资源分布信息。

前提条件

已具备指定集群的任意权限和其名下租户的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**租户管理**。



4. 在**租户管理**页面，查看资源分布信息。

   资源分布信息主要包括各租户占用集群中CPU和内存资源的百分比以及系统剩余CPU和内存资源的百分比。

   将鼠标悬停在某个租户在饼图中的位置时，会显示该租户占用资源的具体数值。

3.1.3.6.2. 新建租户

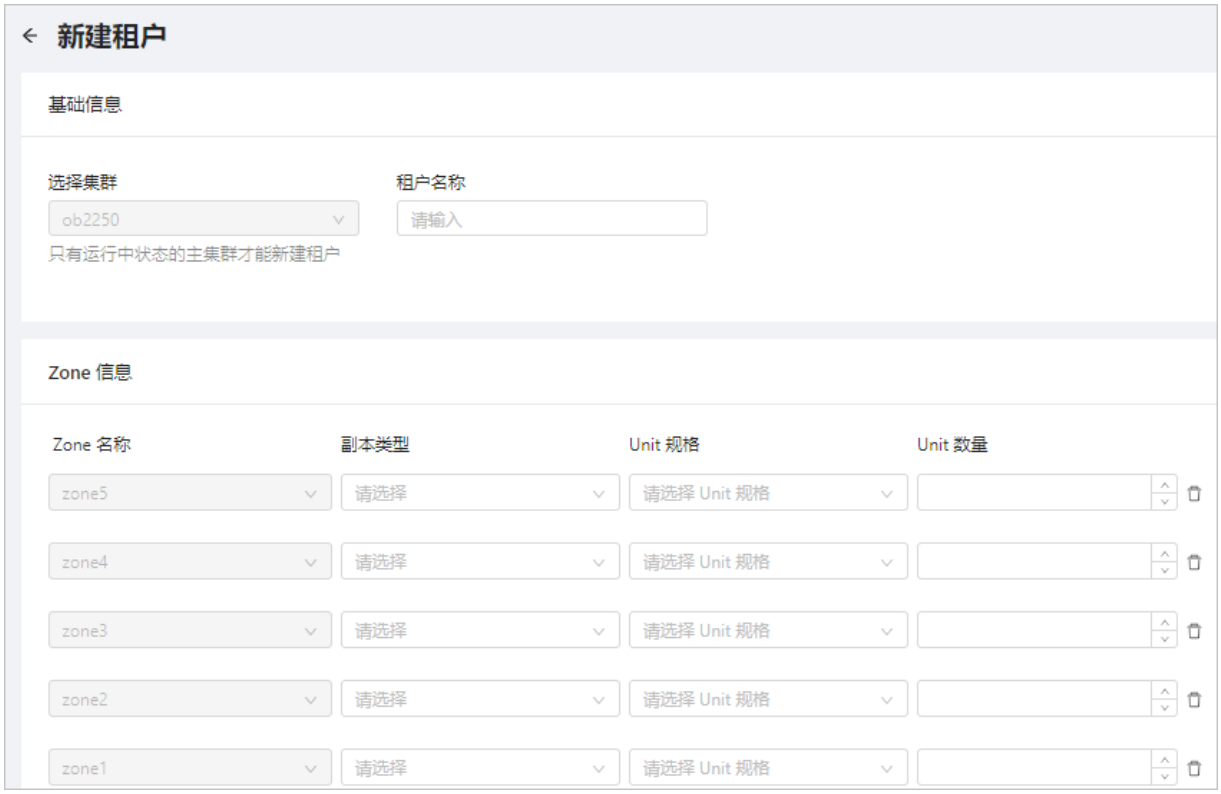
您可以向您管理的集群中新建租户。

前提条件

已具备指定集群的任意权限和其名下租户的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 根据显示的页面，您可以通过以下两种方式进入**新建租户**页面。
   - 在**总览**页面右上角，单击**新建租户**。
   - 在左侧导航栏上，单击**租户管理**，然后在页面右上角单击**新建租户**。



4. 输入新租户的名称。

   租户名称格式需要满足以下条件：

   - 英文大小写字母、数字和下划线组合
   - 长度：2个~64个字符

5. 配置所选集群新租户的 Zone 信息。

   默认系统会根据所选集群的 Zone 信息显示可配置的 Zone 列表，对于无需做副本分布的 Zone，可以单击后面的删除按钮删除该 Zone 条目。需要配置的 Zone 信息及其相关说明如下表所示。

配置	描述
副本类型	有以下选项： - 全功能型副本 - 只读型副本 - 日志型副本
Unit 规格	OCP内置了一套Unit规格，请参考本文档附录中 **OCP资源单元规格列表** 进行选择，或者您也可以在列表下方单击 **新增规格** 按钮新增自定义规格。

配置	描述
Unit 数量	指定该 Zone下的 Unit 数量。 **注意：Unit 数量不能超过该 Zone下 Server 个数。**
Zone 优先级排序	Zone的优先级排序。该优先级顺序影响 sys 租户的 primary Zone 的优先级顺序。 左边的列表框中显示了当前集群的所有 Zone。 您可以在左侧列表框中选择一个或多个 Zone 添加到右侧的列表框中，默认先选择的 Zone 的优先级高于后选择的Zone；一次选中的多个 Zone 的优先级相同。 移动到右侧的列表框中后，您也可以在右侧的列表框中通过拖拽调整顺序，列表框上方的 Zone 的优先级高于下方的Zone。

6. 进行新租户的基本设置。需要完成的基本设置及其相关说明如下表所示。

配置	描述
管理员初始密码	租户管理员账号的密码。 对于 MySQL 模式，其管理员账号为 root；对于 Oracle 模式，其管理员账号为 SYS。 密码格式要求如下： ○ 长度： 8 位~32 位 ○ 至少包含 2 个数字、2 个大写字母、2 个小写字母和 2 个特殊字符。 ○ 支持的特殊字符如下： ._+@#$%
租户模式	当前支持 MySQL 和 Oracle 两种租户模式。需要注意的是，当所选集群的 OB 版本在 2.1 及以上时才支持 Oracle 租户模式。
字符集与编码	对于 MySQL 模式，可选字符集有utf8mb4、binary、gbk 和 gb18030；默认为 utf8mb4。 对于 Oracle 模式，可选字符集有utf8mb4、gbk 和 gb18030，默认为 utf8mb4。
备注	可选，备注说明信息。

配置	描述
IP 地址白名单	用于指定该租户允许登录的客户端列表。如果不指定，则默认配置为 %，表示允许所有客户端链接。 自定义白名单列表时需要注意，OCP 机器地址以及所依赖的 OBProxy 地址必须在此名单中，否则 OCP 将无法管理此租户。 白名单格式说明如下： ○ IP地址示例：10.10.10.10，10.10.10.11。 ○ 子网/掩码示例：10.10.10.0/24。 ○ 模糊匹配示例：10.10.10.% 或 10.10.10._。 ○ 多种格式混合示例：10.10.10.10，10.10.10.11，10.10.10.%，10.10.10._，10.10.10.0/24。

7. 单击**提交**。

3.1.3.6.3. 查看租户列表

您可以查看您管理的集群中所有租户的基本信息。

前提条件

已具备指定集群的任意权限和其名下租户的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**租户管理**。
4. 在**租户管理**页面的**租户列表**区域，查看所有租户的基本信息，包括租户名、副本类型、对应 Zone 的优先级、租户模式、状态等信息。



3.1.3.6.4. 查看租户监控

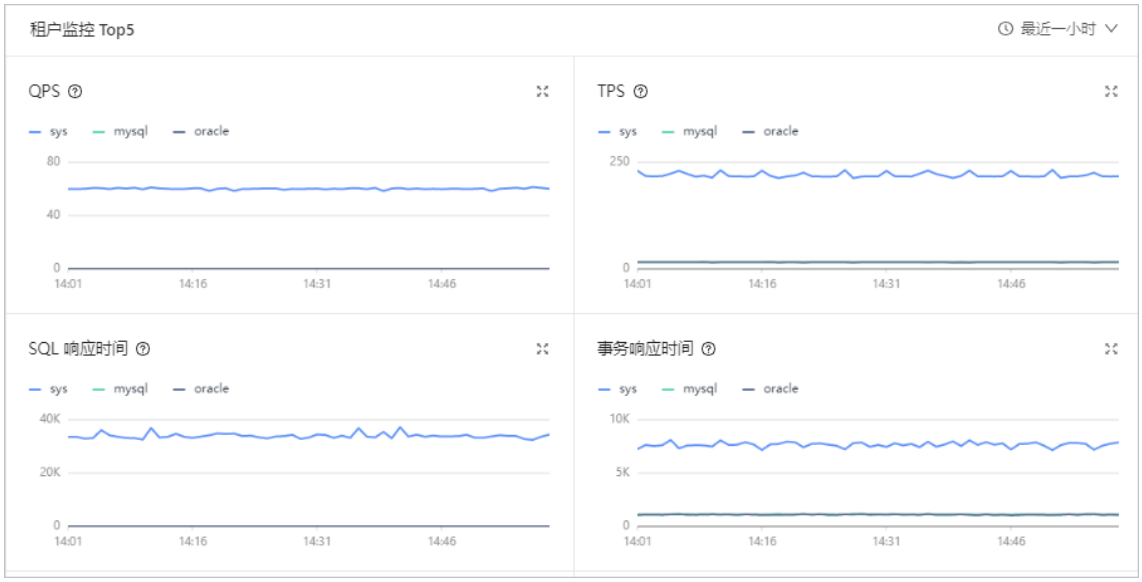
您可以查看您管理的集群中工作负载最高的 5 个租户的性能信息。

前提条件

已具备指定集群的任意权限和其名下租户的管理权限。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**租户管理**。
4. 在**租户管理**的**租户性能 TOP5** 区域，可以查看当前集群下工作负载最高的 5 个租户的性能信息，包含 TPS、QPS、SQL 响应时间、事务响应时间、活跃会话数、事件等待_次数、事件等待_时间、容量_表数量、容量_分区数量等性能监控指标。
   - 同时，您还可以根据业务需要，查看最近一小时、最近一天或最近一周的监控信息。



   - 性能监控指标相关说明如下表所示。

指标名	说明	数据来源
QPS	平均每秒处理sql语句数 (次/s)	v$sysstat
TPS	平均每秒处理事务数 (次/s)	v$sysstat
SQL 响应时间	SQL响应时间 (us)	v$sysstat
事务响应时间	服务端每个事务平均处理时间 (us)	v$sysstat
活跃会话数	活跃会话数 (个)	__all_virtual_processlist

事件等待_次数	平均每秒等待事件数 (次/s)	v$system_event
事件等待_时间	平均每个等待事件等待的时间 (us)	v$system_event
容量_表数量	表数量（个）	gv$table
容量_分区数量	分区数量（个	v$partition

3.1.3.6.5. 管理租户

您可以对租户执行锁定和复制操作。

前提条件

已具备指定集群的任意权限和其名下租户的管理权限。

操作步骤

- 锁定租户
  i. 登录 OCP 。默认进入**集群概览**页面。
  ii. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
  iii. 在显示的页面的左侧导航栏上，单击**租户管理**。
  iv. 在**租户管理**页面的**租户列表**区域，找到待锁定的租户，在对应的**操作**列中单击**锁定**。

> **注意**
> 锁定租户操作会禁止新用户连接到该租户，请谨慎操作。

  v. 在弹出的确认框中，单击**锁定**。



- 复制租户
  i. 登录 OCP 。默认进入**集群概览**页面。
  ii. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
  iii. 在显示的页面的左侧导航栏上，单击**租户管理**。

iv. 在**租户管理**页面的**租户列表**区域，找到待复制的租户，在对应的**操作**列中单击**复制**。

v. 在**复制租户**页面参考新建租户中 Zone 信息设置和基本设置。

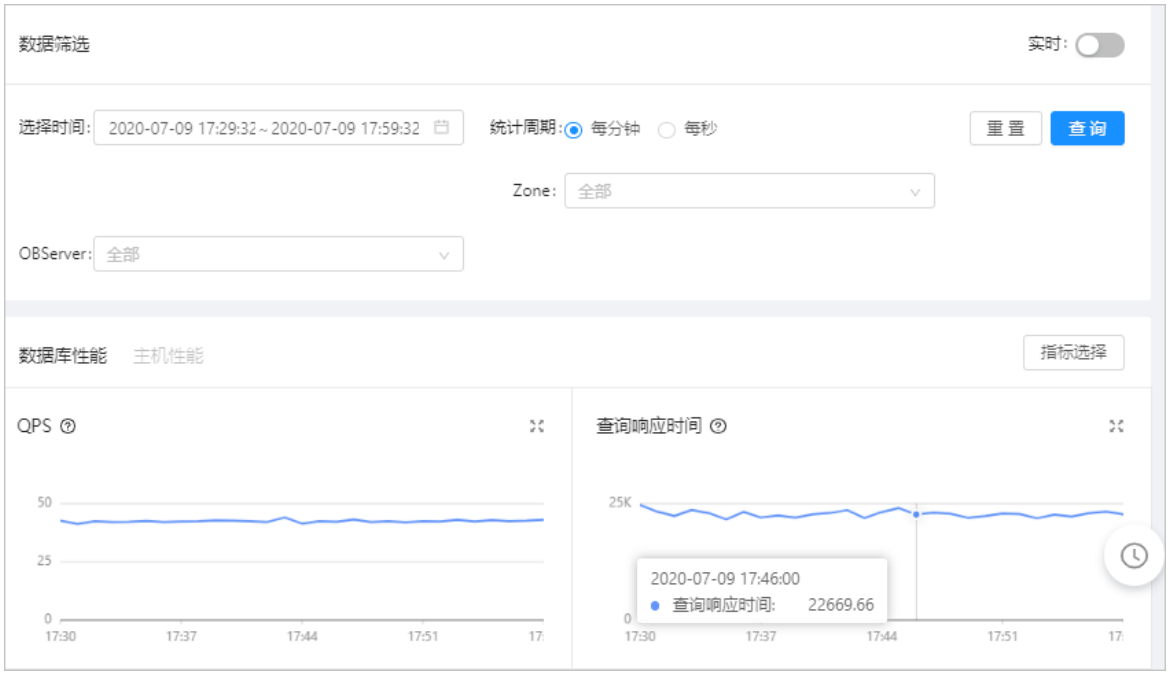
vi. 单击**提交**。提交后，系统会根据当前租户的定义复制一个新的租户，但当前租户中的数据不会被复制。

3.1.3.7. 性能监控

您可以从不同维度查看集群中的各种性能监控数据。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**性能监控**。
4. 在**数据筛选**区域，选择过滤条件，单击**查询**。



部分过滤条件相关说明如下表所示。

过滤条件	描述
选择时间	选择某一段时间范围，用于返回这段时间范围内的数据。

过滤条件	描述
统计周期	主要用于计算每个点数据的统计周期，支持按**每分钟**和**每秒**统计，分别对应了每分钟一个点或每秒钟一个点，OCP 同时也会根据选择的时间区间再去计算一个统计周期，计算的规则是使返回的数据接近 1440 个点，如果选择的时间区间比较长，可能会出现统计周期大于 1 分钟的情况。
Zone	选择待查看的 Zone 。
OBServer	选择待查看的 OBServer 。

5. 切换页签，分别查看集群的数据库性能和主机性能数据。

   如果需要展示其他指标，您可以单击区域右侧的**指标选择**按钮，在指标选择页面选择需要展示的指标，系统最多支持展示 10 个指标。

   数据库性能和主机性能中具体指标的含义请参考本文档附录6. 监控指标章节的内容。

3.1.3.8. 合并管理

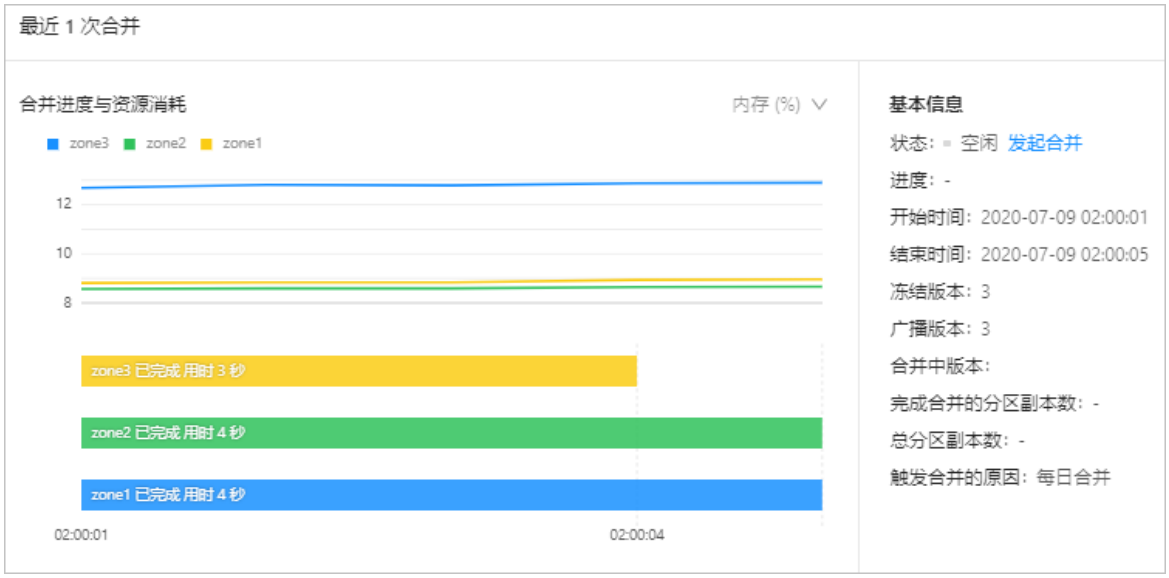
3.1.3.8.1. 合并详情

合并操作（Major freeze）是将动静态数据做归并，会比较费时。当转储产生的增量数据积累到一定程度时，通过Major freeze实现大版本的合并。您可以查看集群的合并详情。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**合并管理**。

4. 在**合并详情**页签中，查看合并详情。



合并详情一般显示的是集群最近的一次合并。如果当前正在合并，则展示合并中的版本，如果当前是空闲状态即未处于合并状态，则显示上一次合并版本详情。

> **说明**
>
> 对于合并中的版本，合并的 **开始时间** 和 **结束时间** 后显示的是预计完成时间。预计完成时间是基于最近三次合并完成的平均值来估算的，可能与实际合并完成时间有较大偏差

一般而言，对于稳定运行的环境，预计完成时间会比较准确，新部署的环境因为估算的数据缺失，估算的准确度会较低。

此外，您还可以通过 **总分区副本数** 和 **完成合并的分区副本数** 来观察分区级别的合并完成情况，推测实际的合并完成进度。

> **注意**
>
> 基于完成合并的分区副本数来直接计算合并完成进度仍然不准确，因为不同分区的大小差异可能很大，每个分区合并的耗时也可能差异很大。

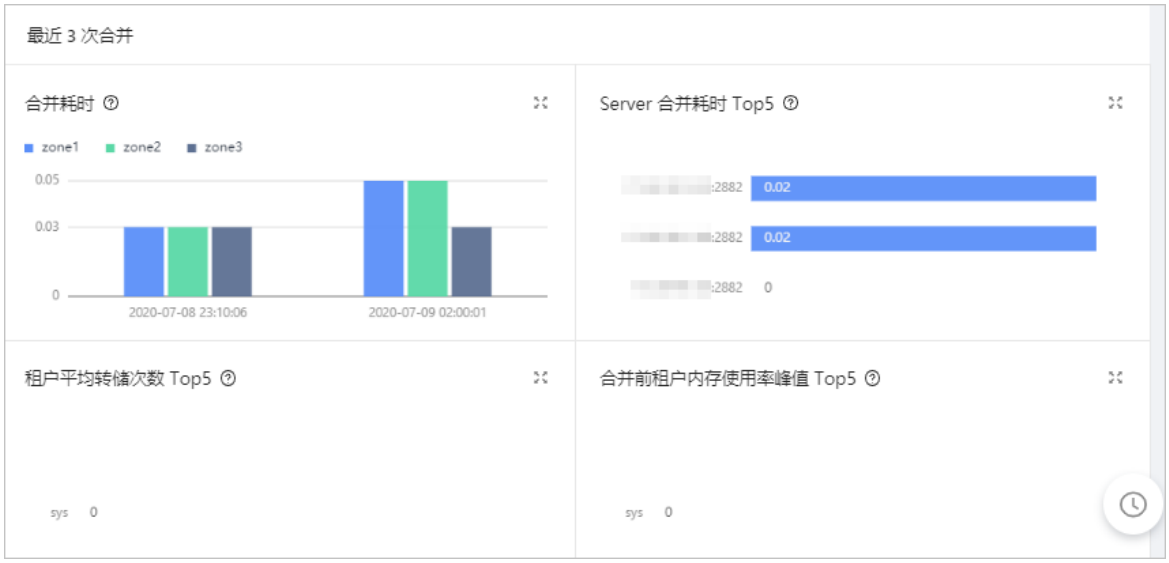
3.1.3.8.2. 合并统计

您可以查看集群的合并统计信息。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**合并管理**。

4. 在**合并详情**页签的**最近 3 次合并**区域，查看合并统计信息。



合并统计提供了最近 3 次合并的 4 个统计图形，包括：

- 合并耗时
- Server 合并耗时 Top 5
- 租户平均转储次数 Top 5
- 合并前租户内存使用率峰值 Top 5

由于合并对集群整体性能会有影响，合并的耗时和性能数据用于辅助观察 OB 集群的健康度。

一般来说，我们希望每日合并只在配置的每日合并时间触发。

租户转储次数和内存使用率超限会触发集群合并，通过观察租户的对应指标，可以判断给租户分配的资源是否充分，可以通过提升租户的资源规格避免因为单个租户资源不足而触发集群合并。

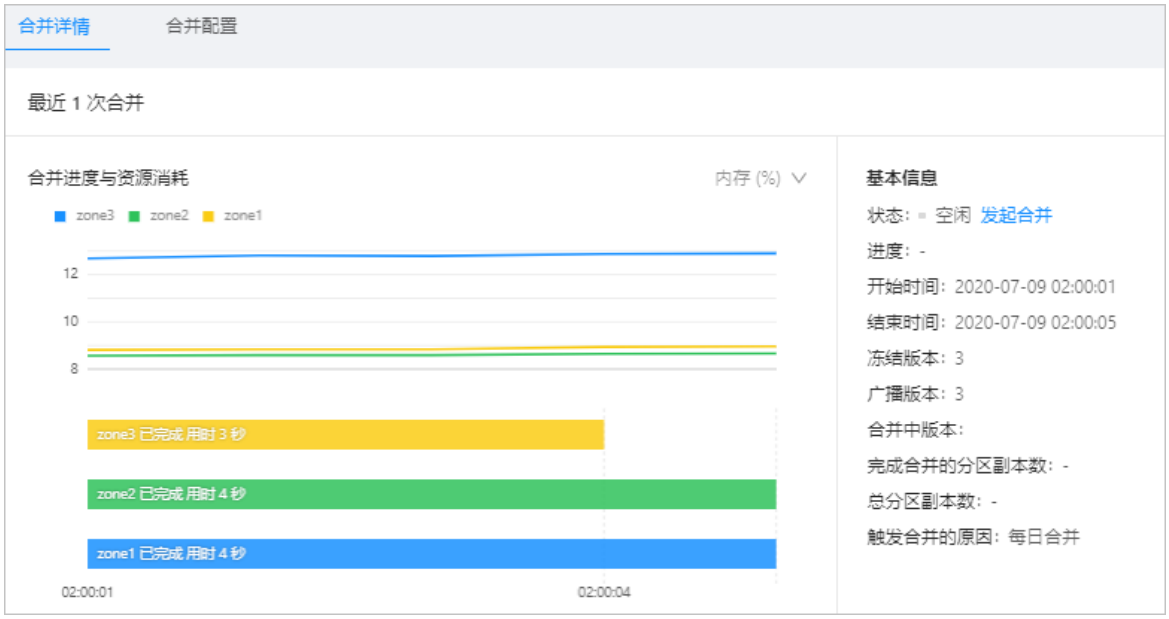
3.1.3.8.3. 执行合并

当集群处于 空闲 状态时，您可以立即触发一次合并。当集群出现合并错误，手动分析排查完合并错误之后，可以单击 继续合并，用于清除合并错误标记以便继续合并。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**合并管理**。

4. 在**合并详情**页签的**最近 1 次合并**区域，找**基本信息**的展示区域。



5. 确认**状态**为**空闲**后，单击**发起合并**。
6. **当集群出现合并错误，手动分析排查完合并错误之后，可以单击 继续合并，用于清除合并错误标记以便继续**

   合并。

> **注意**
>
> 一旦出现合并错误，建议首先联系专业人员确认问题，未排查错误的情况下直接清除合并错误操作可能导致错误范围扩大。

3.1.3.8.4. 修改合并配置

您可以根据业务需要配置合并策略和转储策略，用于每日合并。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**合并管理**。

4. 单击**合并配置**页签。



5. 配置合并策略。
   i. 在**合并策略**区域，单击**修改**。
   ii. 将鼠标悬停在各配置项的图标上，根据系统提示配置合并策略。
   iii. 完成后，单击**保存**。
7. 配置转储策略。
   i. 在**转储策略**区域，单击**修改**。
   ii. 将鼠标悬停在各配置项的图标上，根据系统提示配置转储策略。
   iii. 完成后，单击**保存**。完成后，单击**保存**。

3.1.3.9. 参数管理

3.1.3.9.1. 查看参数列表

通过参数列表，您可以查看当前集群所有参数的名称、类别、取值类型、取值范围、默认值、当前值、说明及是否重启生效等。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**参数管理**。

4. 在**参数列表**页面，可以查看当前集群所有参数的信息，包括各参数的参数名称、取值类型、取值范围、默认值、当前值、参数说明和是否重启生效等信息。



参数名称	类别	取值类型	取值范围	默认值	当前值	说明	是否重启生效	操作
auto_delete_expired_...	OBSERVER	ENUM	[TRUE/FA...	FALSE	False (集...	自动删除...	否	修改值
backup_region	OBSERVER	STRING			(集群)	用户显式...	否	修改值
backup_recovery_wi...	OBSERVER	INT	[0,]		0 (集群)	备份数据...	否	修改值
log_archive_checkpo...	OBSERVER	TIME	[5s, 1h]		30s (集群)	为冷数据...	否	修改值
log_archive_concurre...	OBSERVER	INT	[0,]		0 (集群)	日志归档...	否	修改值

> **说明**
>
> - 如果**当前值**是集群范围内统一，则如上图 backup_region 的参数值所示，表示 0（集群）。
> - 如果参数在 Zone 或者 Server 级别不一样，则会显示所有值的组合，例如，12；10 自定义）

3.1.3.9.2. 修改参数

您可以根据需要修改集群参数的值及生效范围。

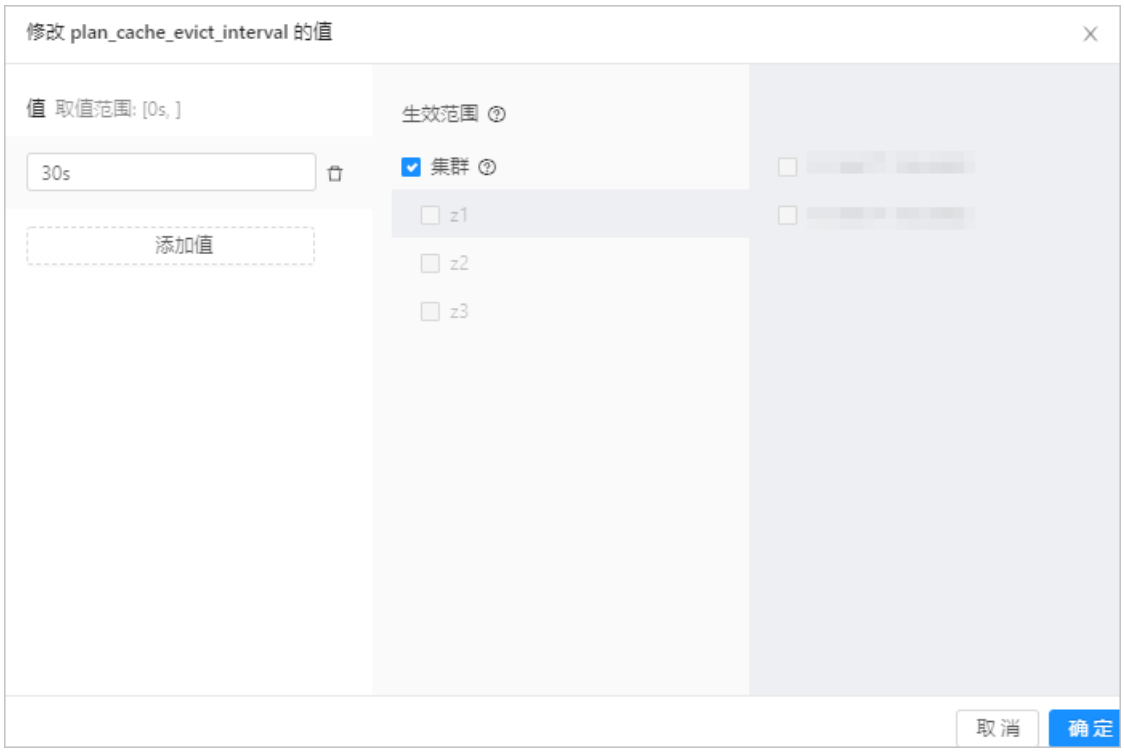
操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**参数管理**。
4. （可选）在**参数列表**页面上方的搜索框中，输入参数名相关信息进行模糊搜索。



5. 找到待修改的参数，在对应的**操作**列中，单击**修改值**。

6. 在弹出的对话框中，修改参数的值及生效范围，单击**确定**。



由于 OceanBase 集群参数可以有全局（即**集群**）、Zone 和 Server 三种生效范围，故在修改值时请根据业务需要选择生效范围。

默认生效范围是**集群**。如果需要调整到 Zone 或 Server 的生效范围，则可以在**生效范围** 列取消选中**集群**，此时系统会显示集群下的 Zone 列表。根据提示选择 Zone并选中，则选中了 Zone 的生效范围；如果选择 Zone 后，继续选择该 Zone 下的 Server 列表，则在 Server 范围生效。

如果需要修改参数的值同时在不同的范围生效，例如，修改 **backup_concurrency** 在 **Zone 1** 为 **10**；在 **Zone 2**为 **12**；在 **Server 1** 上为 20，则可以在**值**列点单击**添加值**后出现 3 行记录。每行记录可以选中对应的生效范围。

> **说明**
>
> 如果单击**添加值**后出现多行修改值时，对集群的参数修改顺序是从第一行往下依次执行，并且每次执行成功后，会在 **修改历史** 页签中产生一条历史记录。

3.1.3.9.3. 集群参数类型说明

当前 OceanBase 支持 BOOL、CAPACITY、DOUBLE、INT、MOMENT、STRING、STRING_LIST 和 TIME 多种参数类型。

主要类型及默认的取值单位如下：

参数	说明
BOOL	boolean 类型（布尔），支持 true/false。

参数	说明
CAPACITY	容量单位，支持 b （字节）、k （KB，千字节）、m （MB，百万字节）、 g （GB，10亿字节）、t （TB，万亿字节）、p（PB，千万亿字节）。不区分单位大小写字母，默认 m。
DOUBLE	双精度浮点数，占用 64bit 存储空间，精确到小数点后 15 位，有效位数为 16位 。
INT	int64整型，支持正负整数和0。
MOMENT	时刻。格式为 hh:mm（比如'02:00'）；或者特殊值 disable，表示不指定时间。目前仅用在 major_freeze_duty_time 参数。
STRING	字符串。用户输入的字符串的值，依据 OceanBase 文档中对应参数的要求。
STRING_LIST	字符串列表，即以;分隔的多个字符串。
TIME	时间类型。支持us （微妙）、ms （毫秒）、s （秒），m （分钟）、h （小时）、d （天）等单位。如果不加后缀，默认为秒（s）。不区分单位大小写字母。

3.1.3.9.4. 查看参数修改历史

您可以在修改历史页面查看用户通过 OCP 对当前集群发起的所有参数修改的历史记录，包括每次参数修改的参数名称、变更前后的参数值、修改时间和操作用户。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**参数管理**。
4. 单击**修改历史**页签。
5. 通过**修改日期**、**参数名称**或**操作用户**进行过滤查询。

其中，**参数名称** 支持模糊查询；**操作用户** 必须要精确匹配。



6. 在查询结果中，查看参数的历史修改信息。

3.2. 租户管理

3.2.1. 概述

租户是一个逻辑概念，在 OceanBase 里是资源分配的单位，是数据库对象管理和资源管理的基础。对于系统运维，尤其是对于云数据库的运维有着重要的影响。登录 OCP 后，在左侧导航栏中单击租户可以进入租户概览页面。页面默认展示登录用户有权限查看的租户列表、租户监控 Top5 信息。

3.2.2. 新建租户

租户（Tenant）既是各类数据库对象的容器，又是资源（CPU、Memory、IO等）的容器。您可以根据业务需求在集群中创建租户。

新建租户有以下两种方式：

- 在租户概览页面创建租户。
- 在指定集群的租户管理页面创建租户。

本文档为您展示指定集群的租户管理页面创建租户的操作步骤。

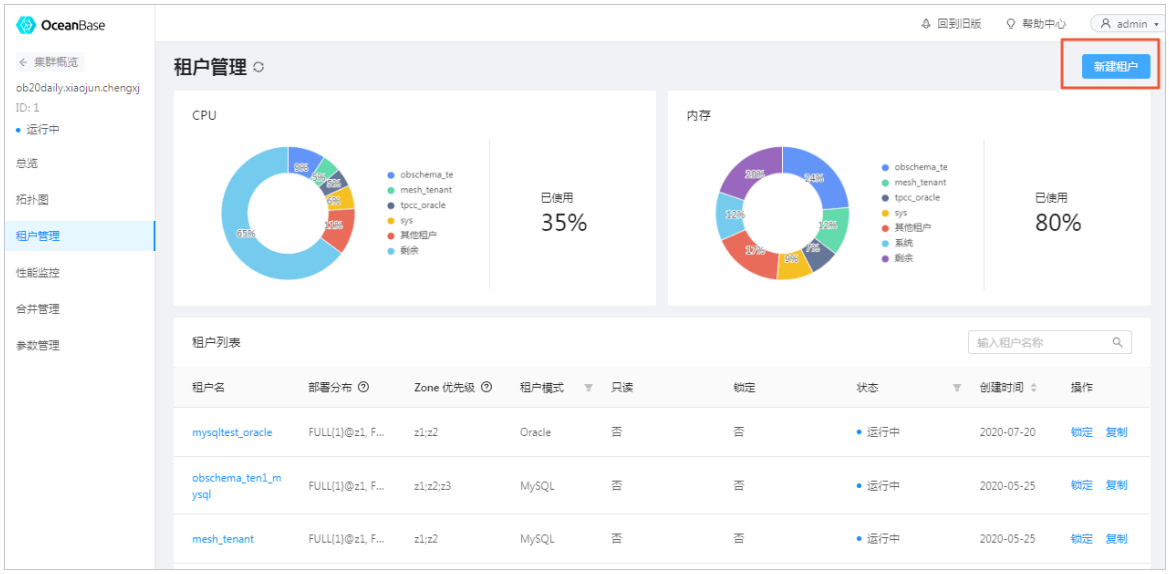
前提条件

- 需要创建租户的集群为主集群，且状态为运行中。
- 当前登录用户具备系统管理员角色、OCP 租户管理员角色或其他拥有管理该集群权限的角色。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在**集群概览**页面的**集群列表**区域，选择待操作的集群并单击其集群名。
3. 在显示的页面的左侧导航栏上，单击**租户管理**。

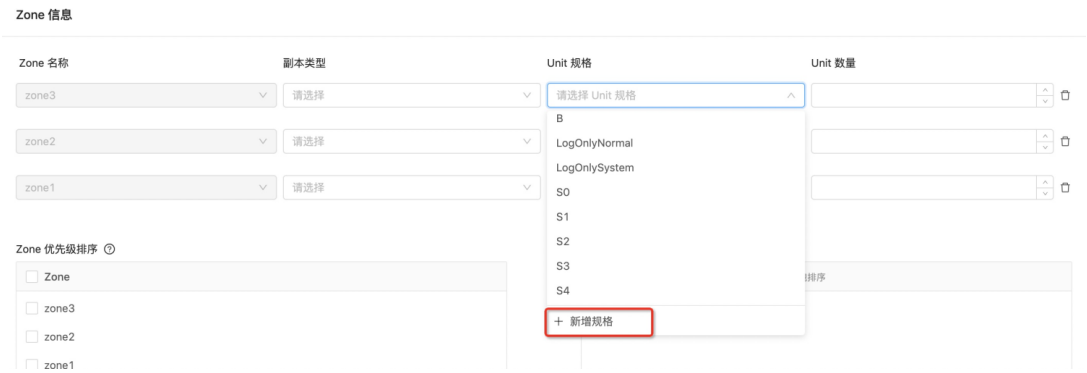
4. 在页面右上角单击**新建租户**。



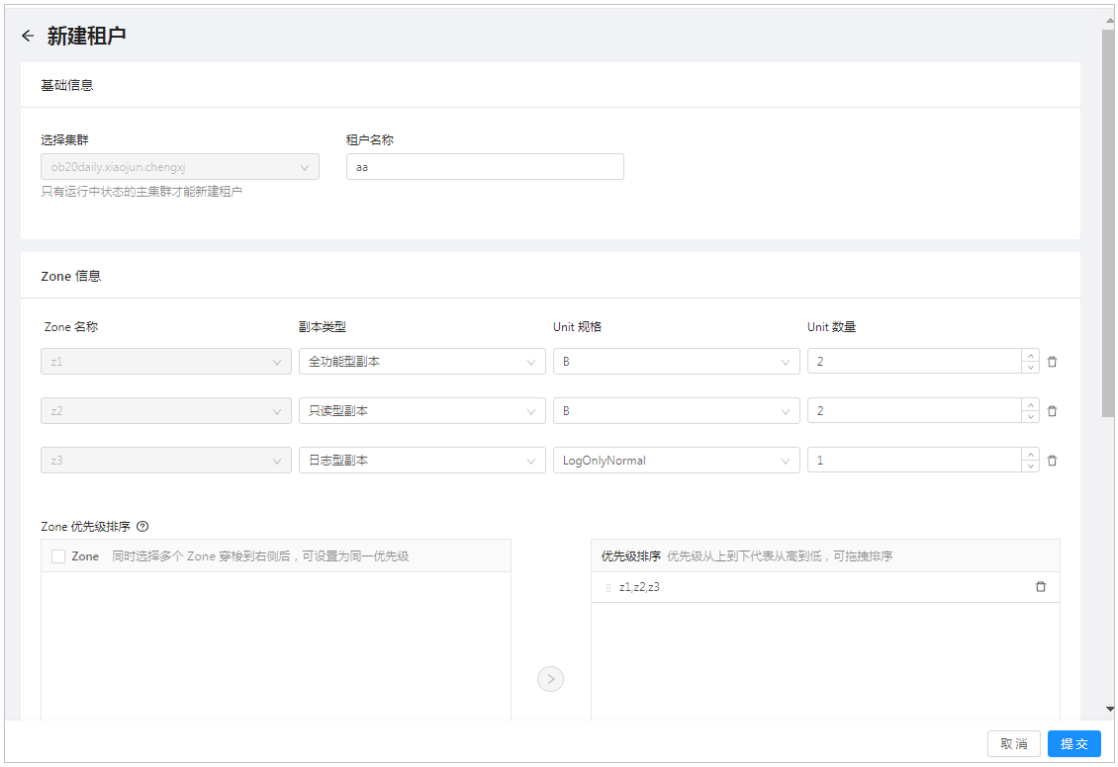
5. 填写**基础信息**。
   i. 集群默认为当前集群。
   ii. 请输入**租户名称**。租户名称格式为英文大小写字母、数字和下划线的组合，长度为 2~64字符。
6. 填写 **Zone 信息**。
   i. 为 z1、z2 和 z3 设置副本类型、Unit 规格和 Unit 数量。

   集群选定后，页面会根据所选集群的 zone 信息给出可配置的 zone 列表；对于无需做副本分布的 zone，可以通过最右侧的按钮删除该zone条目。

   - 副本类型支持全功能型副本、只读型副本和日志型副本。
   - OCP 内置了一套 Unit 规格；另外也可以在下拉列表的最下方点击按钮新增自定义规格。
   - 指定该 zone下的 Unit 数量。需要注意的是，unit 数量不能超过该 zone下 server个数。



ii. 对 **Zone 优先级** 进行排序。同时选择多个 Zone 穿梭到右侧后，可设置为同一优先级。



7. 填写**基本设置**。

i. 设置

- 对于 MySQL 模式，其管理员账户为"root"。
- 对于 Oracle 模式，其管理员账户为“SYS”。密码格式：长度为8 ~ 32，且至少包含 2 个数字、2 个大写字母、2 个小写字母和 2 个特殊字符，支持的特殊字符为 ._+@#$%。

ii. 设置**租户模式**。

- 支持Oracle 和 MySQL 租户模式。
- 所选集群的 OB 版本在 2.1 及以上时才支持 Oracle 租户模式。

iii. 设置**字符集与编码**。

- 对于 MySQL 模式，可选字符集有：utf8mb4, binary, gbk, gb18030。缺省为utf8mb4。
- 对于 Oracle 模式，可选字符集有：utf8mb4, gbk, gb18030。缺省为utf8mb4。

iv. （可选）设置**备注**。

v. 设置 **IP 地址白名单**。

- 此处可以指定该租户允许登录的客户端列表。如果不指定，默认配置为"%"，允许所有的客户端。自定义白名单列表时需要注意，OCP 机器地址以及所依赖的 OBProxy 地址必须在此名单中，否则 OCP 将无法管理此租户。
- 默认配置：所有 IP 都可访问。
- 自定义：设置 IP 白名单，白名单中的IP才能访问。

- 白名单格式说明：
  - IP地址，示例：10.10.10.10,10.10.10.11

- 子网/掩码，示例：10.10.10.0/24
- 模糊匹配，示例：10.10.10.% 或 10.10.10._
- 多种格式混合，示例：10.10.10.10,10.10.10.11,10.10.10.%,10.10.10._,10.10.10.0/24

> 说明
>
> % 表示所有客户端都可以连接



8. 单击**提交**。

3.2.3. 查看租户列表

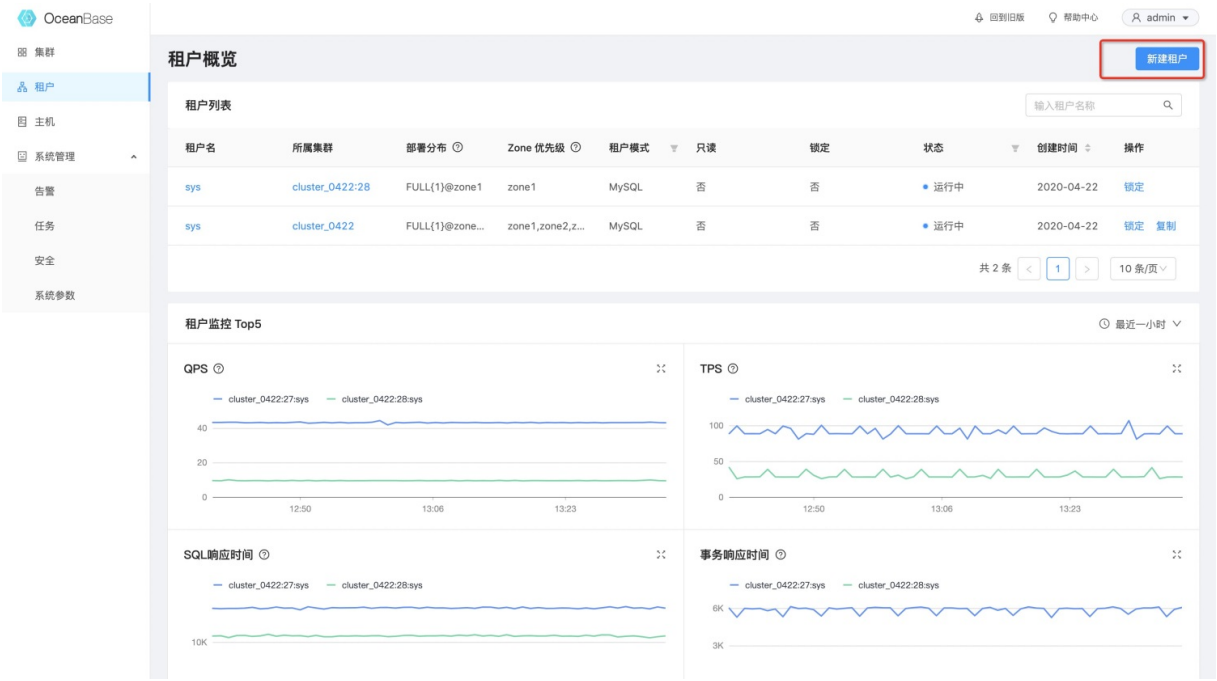
在租户列表页面，有指定租户的只读或全部权限的用户可以查看指定租户详细信息及监控数据。

租户列表

租户列表区域显示租户名、所属集群、部署分布、Zone 优先级、租户模式、是否只读、是否锁定、状态和创建时间。您也可以在租户列表中对指定租户进行锁定或复制操作。也支持分页查询、和租户名称进行模糊匹配。

部署分组指租户在各个 Zone 上的副本类型，Zone 优先级指租户中分区主副本分布的优先级。

用户对租户的“管理”权限，主要是租户的新建、删除、修改和只读。因此，不同的用户，有的用户“管理”的租户多，有的用户“管理”的租户少，在这里显示的结果记录数量也不一样。同理，对于某个租户，“用户A”具备只读操作，而“用户B”具备修改和删除操作，则在“操作”列表上也会区分。



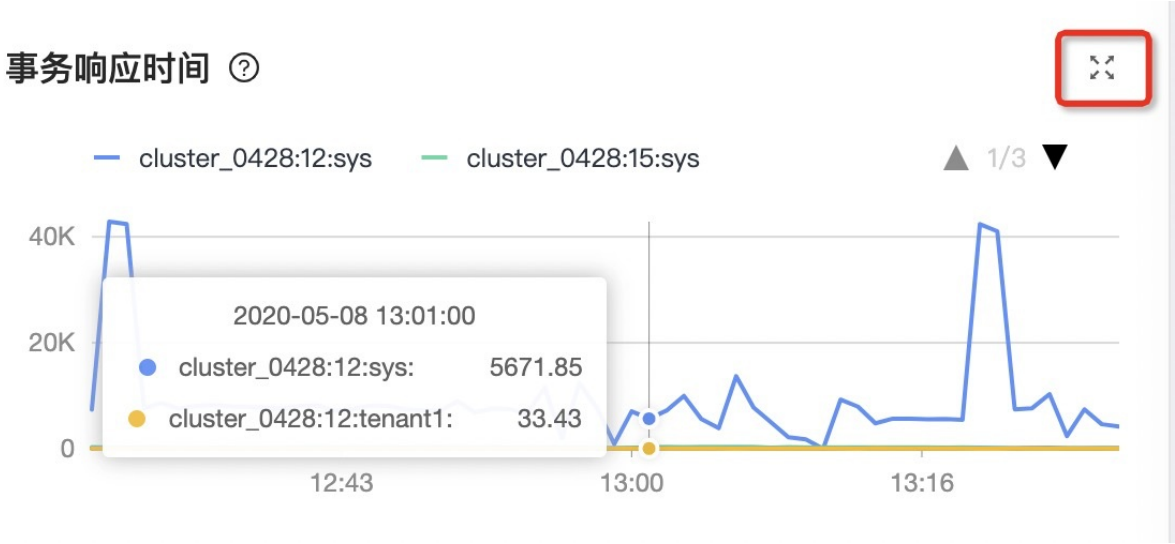
租户监控Top5

租户监控 Top5 区域主要从全局角度展示所有集群下所有租户在最近一段时间Top 5 租户的核心监控信息。包括：

- QPS：平均每秒处理sql语句数 (次/s)，数据来源：v$sysstat
- TPS：平均每秒处理事务数 (次/s)，数据来源：v$sysstat
- SQL 响应时间： SQL响应时间 (us)，数据来源：v$sysstat
- 事务响应时间：服务端每个事务平均处理时间 (us)，数据来源：v$sysstat
- 活跃会话数：活跃会话数 (个)，数据来源：__all_virtual_processlist
- 事件等待_次数：平均每秒等待事件数 (次/s)，数据来源：v$system_event
- 事件等待_时间：平均每个等待事件等待的时间 (us)，数据来源：v$system_event
- 容量_表数量：表数量（个），数据来源：gv$table
- 容量_分区数量：分区数量（个），数据来源：v$partition

默认显示最近一小时内的 Top 5租户的监控信息。 也可以从下拉框选择“最近一天”、或“最近一周”。

如下图所示，在每一个监控的小窗口，当鼠标停在某个具体时间点时，会弹出明细信息的窗口，显示当前时间、租户名、以及该租户对于的监控指标值。



如果需要放大显示某一个具体监控信息，也可以选择该监控的小窗口右上角的“X”按钮，进行放大。此时，会在当前页面弹出一个放大后的窗口，并支持鼠标停顿后显示指标值。结束后点击关闭窗口按钮即可，此时又重新回到租户概览页面。

3.2.4. 管理租户

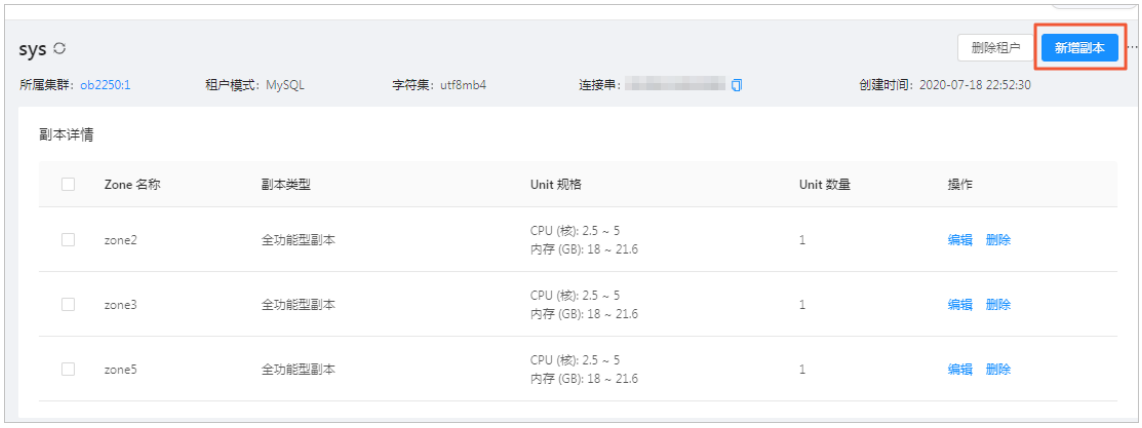
3.2.4.1. 管理租户副本

租户最多可以在所属集群的全部 Zone 上设置副本。每个Zone下仅允许创建一个副本。

操作步骤

- 新建副本
  i. 登录OCP。在左侧导航栏中单击**租户**。
  ii. 在**租户列表**中找到指定租户，并单击租户名称。
  iii. 在页面右上角单击**新建副本**。

> **说明**
> 如果租户已在所属集群的全部 Zone 上设置副本，不支持再新增副本。



iv. 设置**目标 Zone**，在目标 Zone 下新增对应副本。已经存在副本的 Zone，不会出现在目标 Zone 的可选列表下。

v. 设置**副本类型**和**资源池**。副本类型和资源池的相关说明请查看新建租户章节。



vi. 单击**确定**。

- 编辑副本

  租户的总览页面可以查看当前租户已创建的所有副本清单和副本详情，副本详情部分展示了租户在各 Zone 下的副本分布情况，包括 Zone 名称、副本类型、Unit 规格和 Unit 数量。如果需要修改这些信息，可以在对应的**操作**列单击**编辑**，重新设置租户副本相关的配置。



- 删除副本

  一般情况下，不需要的副本可以由管理员删除。在指定副本对应的操作列单击删除，并在弹出对话框中单击删除即可删除对应 Zone 下的副本。

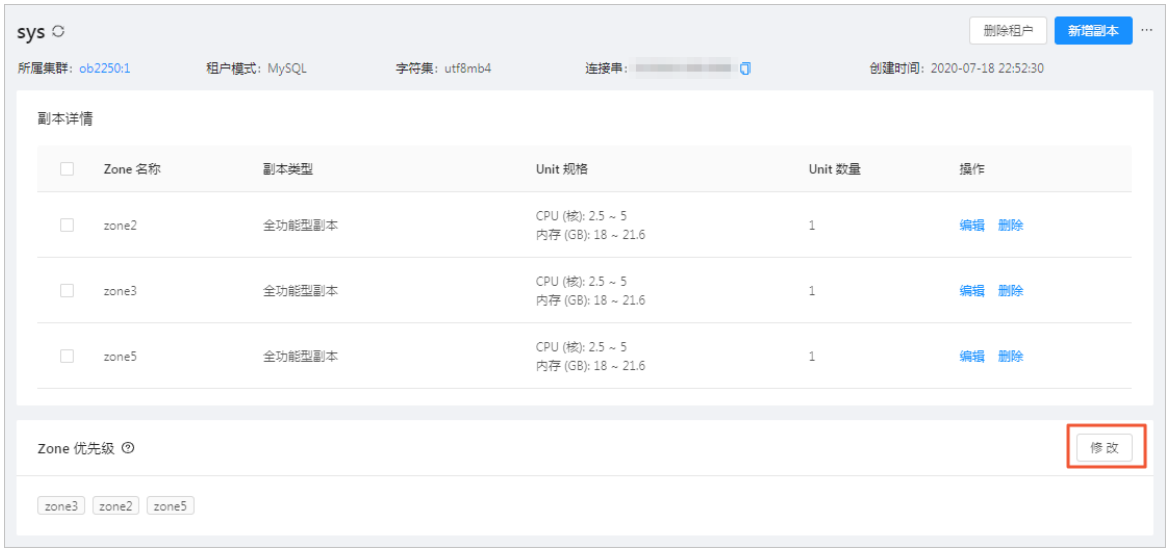
> **说明**
>
> 如果 OB 集群的 sys 租户有 5 个 Zone 副本，其中 2 个 Zone 异常停止，此时请勿删除任意一个副本。但可以批量删除宕机 Zone 上的两个副本。

3.2.4.2. 修改 Zone 优先级

Zone 优先级指租户中分区主副本分布的优先级。创建租户时已设置 Zone 优先级，后续可以根据业务需求更改。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**租户**。
3. 在**租户列表**中找到指定租户，并单击租户名称。
4. 在 **Zone 优先级**区域右上角单击**修改**。



5. 重新设置 **Zone 优先级排序**。
   - 同时选择多个 Zone 穿梭到右侧后，可设置为同一优先级。
   - 右侧优先级从上到下代表从高到低，可拖拽排序。



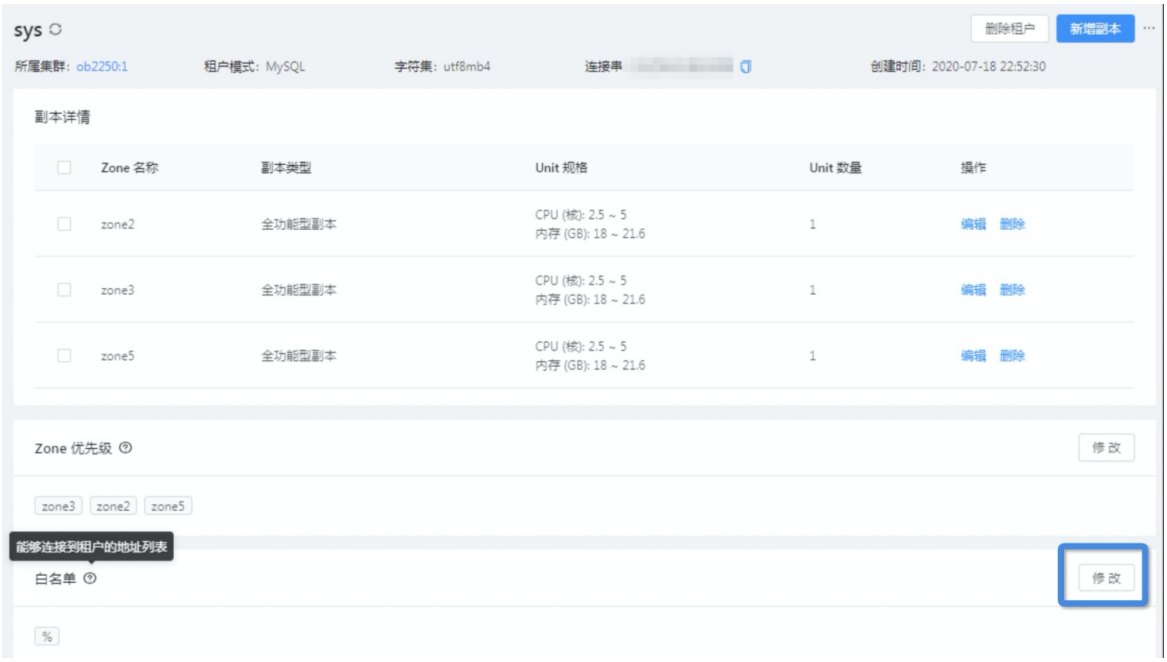
6. 单击**确定**。

3.2.4.3. 修改租户白名单

租户白名单指能够连接到租户的地址列表。创建租户时，如果未设置租户白名单，或后续需要修改租户白名单，可以在租户总览页面修改。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**租户**。
3. 在**租户列表**中找到指定租户，并单击租户名称。
4. 在**白名单**区域右上角单击**修改**。



5. 设置白名单。
   - **默认配置**：指所有IP都可以访问。
   - **白名单**：仅白名单中的IP可以访问。请根据页面说明和配置说明，重新设置白名单，白名单设置的字符数不超 128 个字符。



6. 单击**确定**。

3.2.4.4. 修改密码

对于 MySQL 兼容模式，管理员用户为 root 用户；对于 Oracle 兼容模式，管理员用户为 SYS 用户。在指定租户的总览页面中，系统管理员或租户管理员可以为租户重新设置管理员用户密码。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**租户**。
3. 在**租户列表**中找到指定租户，并单击租户名称。
4. 在页面右上角展开隐藏菜单，并单击**修改密码**。
5. 修改密码弹框中需要输入两次新密码。
6. 确认是否**设为默认连接信息，并保存到密码箱**。

建议选中，方便租户下次登录 OCP 时，直接使用新的密码登录到对应的集群。



7. 单击**确定**。

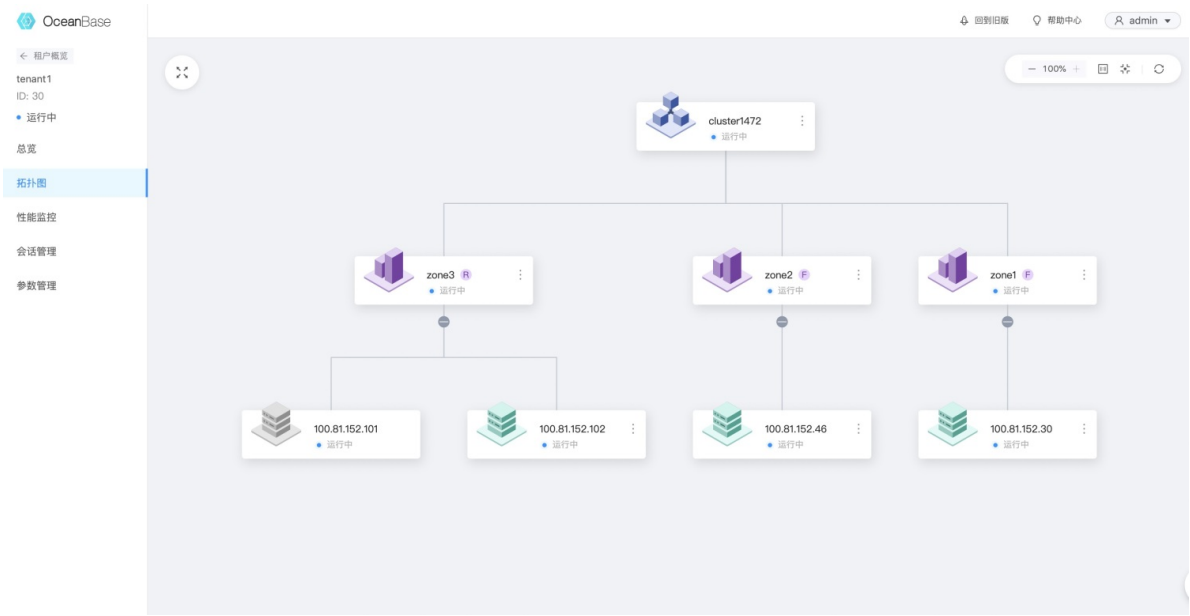
3.2.4.5. 查看租户拓扑图

登录 OCP 之后，在左侧导航栏单击租户，默认进入租户总览页面。在左侧单击拓扑图，可以查看当前租户的详细拓扑图。

这里展示租户在集群下副本分布情况。页面右上角，可以调整拓扑图的大小、刷新当前拓扑图。

拓扑图分三层：

- 第一层为集群
- 第二层为 Zone
- 第三层为 Server



集群

集群图标上展示了租户所属的集群名，以及该集群当前的状态。

单击图标右侧的操作按钮，可打开操作列表。

- 单击 **查看该集群**，会在新页面打开集群拓扑图。
- 单击 **修改租户的 Zone 优先级**，可以在弹窗中重新设置 Zone 优先级。



Zone

Zone 图标上展示了 Zone 名称以及该 Zone当前的状态。

如果租户在该 Zone 上存在副本分布，则该 Zone图标显示为紫色，且租户名右侧显示副本类型的简写，否则显示为灰色。

- 当鼠标悬停在Zone图标上时，会提示该 Zone上分布的副本类型以及 unit 数量。



- 单击 Zone 图标右侧的操作按钮，可打开操作列表。



- 单击**修改副本类型**，会弹出如下窗口，可对副本类型进行修改。



- 单击**修改资源池**，会弹出如下窗口，可对租户在该zone上的资源池进行修改。



- 单击**删除副本**，会弹出如下提示窗口，单击**删除**可对该zone下的副本进行删除操作。



Server

Server 图标上展示了 Server 的 IP 和端口以及该 Server 当前的状态。

如果租户在该 Server上存在 unit 分布，该 Server 图标显示为绿色，否则显示为灰色。

单击 Server 图标右侧的操作按钮，可打开操作列表。

目前支持删除 Unit：在弹出对话框中单击**删除**，可以立即删除指定 Unit。



3.2.4.6. 查看性能监控

本章节为您介绍性能监控页面可查看的数据筛选方式和租户监控指标。

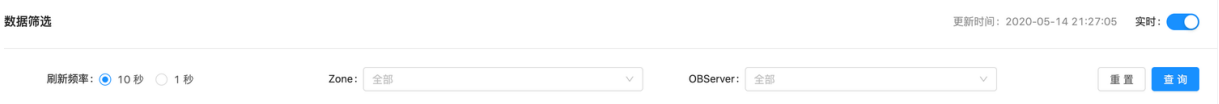
数据筛选

用户可以根据自己的需要按照多种维度对租户的性能信息进行筛选。

- **时间**：选择某一段时间范围，返回这段时间范围内的数据。
- **统计周期**：
  - 计算每个点数据的统计周期，支持每分钟和每秒，分别对应了每分钟一个点，每秒钟一个点。OCP同时也会根据选择的时间区间再去计算一个统计周期。
  - 计算的规则是使返回的数据接近1440个点，如果选择的时间区间比较长，可能会出现统计周期大于1分钟的情况。
- **Zone**：查看指定 Zone 的性能监控数据。
- **OBServer**：查看指定 OBServer 的性能监控数据。



OCP 租户性能监控支持查看实时数据，在页面右上角打开**实时**开关，可以查看实时数据。数据会根据选择的刷新频率定时刷新，默认展示最近两分钟的秒级数据， 支持的刷新频率为10秒和 1 秒。



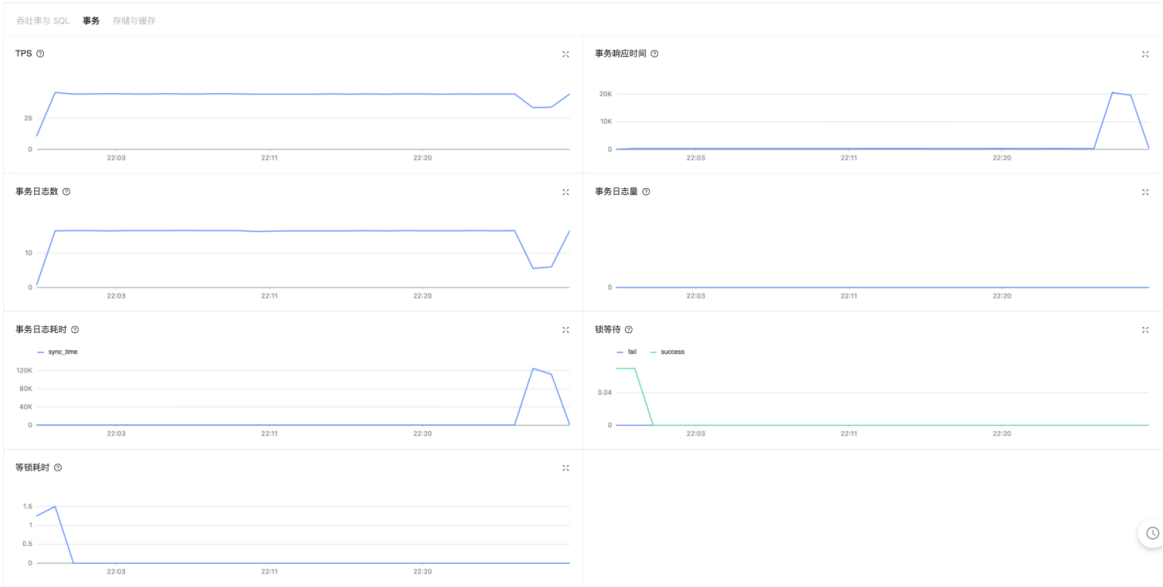
租户监控指标

- 吞吐率与SQL：吞吐率与 SQL 区域主要展示以下指标的监控数据：
  - QPS：平均每秒处理 SQL 语句数
  - 响应时间：响应时间（us）
  - 活跃会话数：活跃的会话数量（个）
  - SQL 执行计划类别：SQL 执行计划的类别
  - 等待事件：每秒等待事件次数 （次/s）
  - 等待事件耗时：等待事件平均耗时（us）
  - 请求等待队列：每秒 SQL 进入等待队列个数（次/s）

- 请求等待队列耗时：SQL 请求在等待队列中的等待耗时（us）



- 事务：事务页签主要展示以下指标的监控数据：
  - TPS：平均每秒处理事务数（次/s）
  - 事务响应时间：服务端每个事务平均处理时间（us）
  - 事务日志数：每秒提交的事务日志数（次/s）
  - 事务日志量：每秒提交的事务日志大小 （byte）
  - 事务日志耗时：服务端处理事务日志的平均耗时（us）
  - 锁等待：每秒事务锁等待次数
  - 等锁耗时：每个锁等待平均耗时（us）



- 存储与缓存：该区域主要展示以下指标的监控数据：
  - MEMStore：OB 可写入数据内存（Mb）
  - IOPS：平均每秒 IO 次数（次/s）

- IO 耗时：平均每次 IO 耗时（us）
- IO 吞吐率：平均每秒 IO 数据量（byte）
- 缓存大小：缓存大小（Mb）
- 缓存命中率：缓存命中率（%）



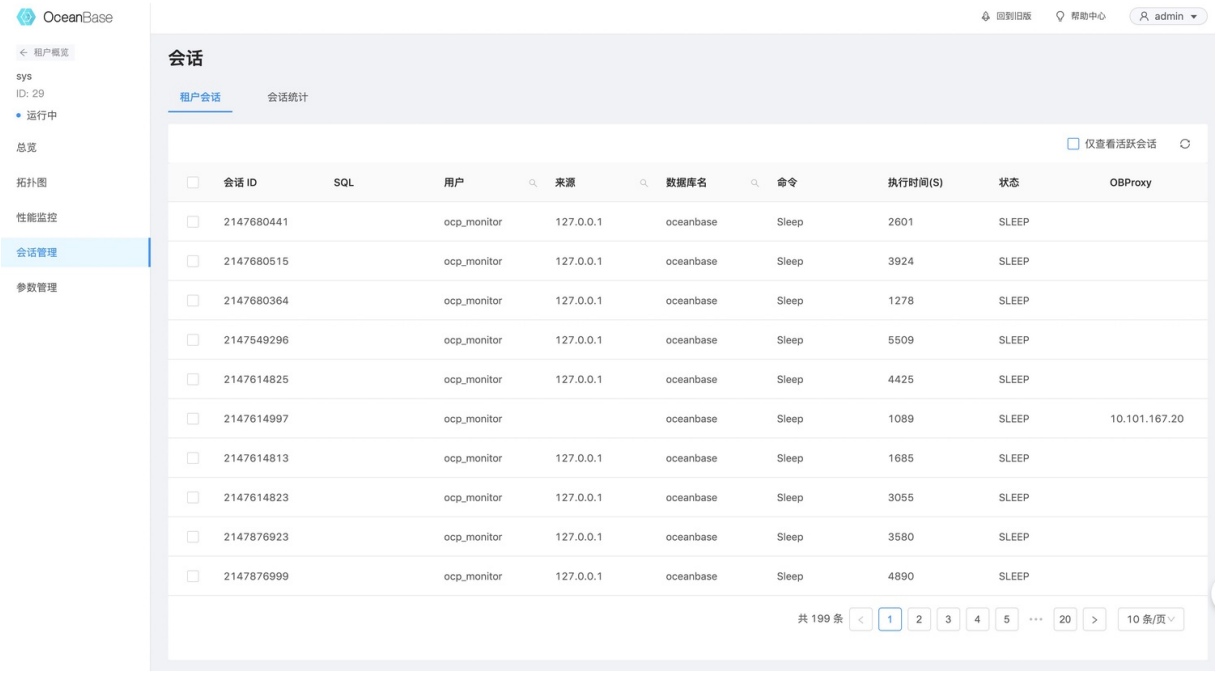
3.2.4.7. 会话管理

3.2.4.7.1. 查看租户会话

登录 OCP 之后，在左侧导航栏单击 租户，默认进入租户 总览 页面。在左侧单击 会话管理，默认显示当前租户的所有会话列表。右上角单击刷新按钮可以刷新页面。

查看会话列表

租户会话 页签展示租户当前的所有会话信息。包括会话 ID、SQL、用户、来源、数据库名、命令、执行时间(S)、状态和 OBProxy。



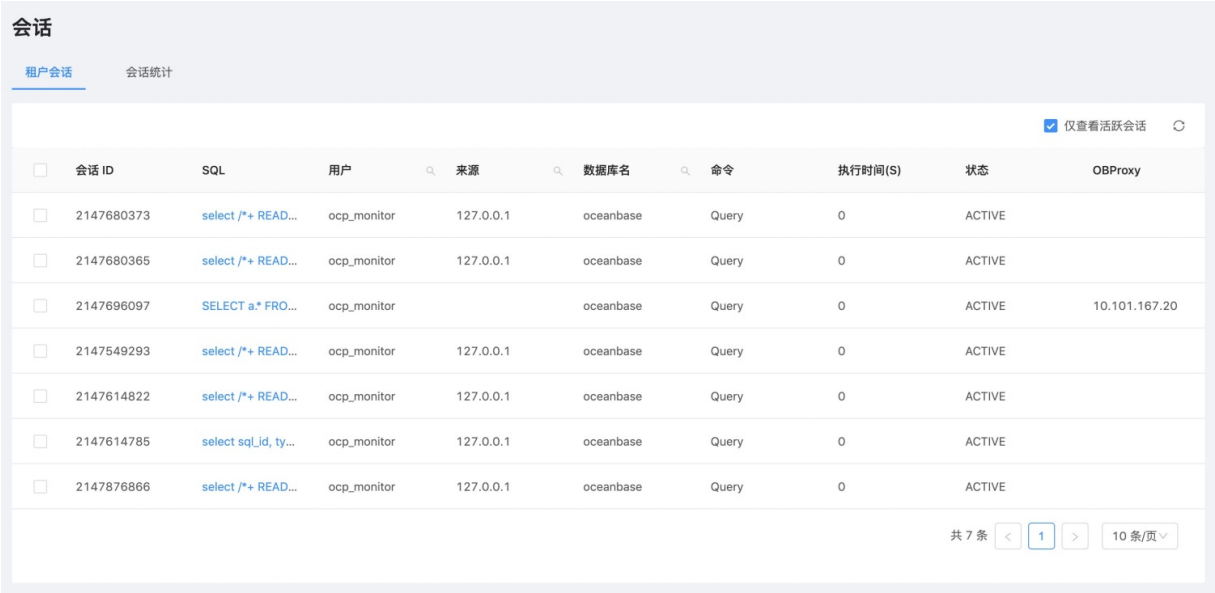
检索会话

用户、来源、数据库名这三列，支持检索功能。点击列名右侧的放大镜按钮，可打开检索输入框。



查看活跃会话

勾选会话列表右上角的 **仅查看活跃会话** 后，会话列表会按照会话状态进行过滤，仅展示状态为 **ACTIVE** 的会话。

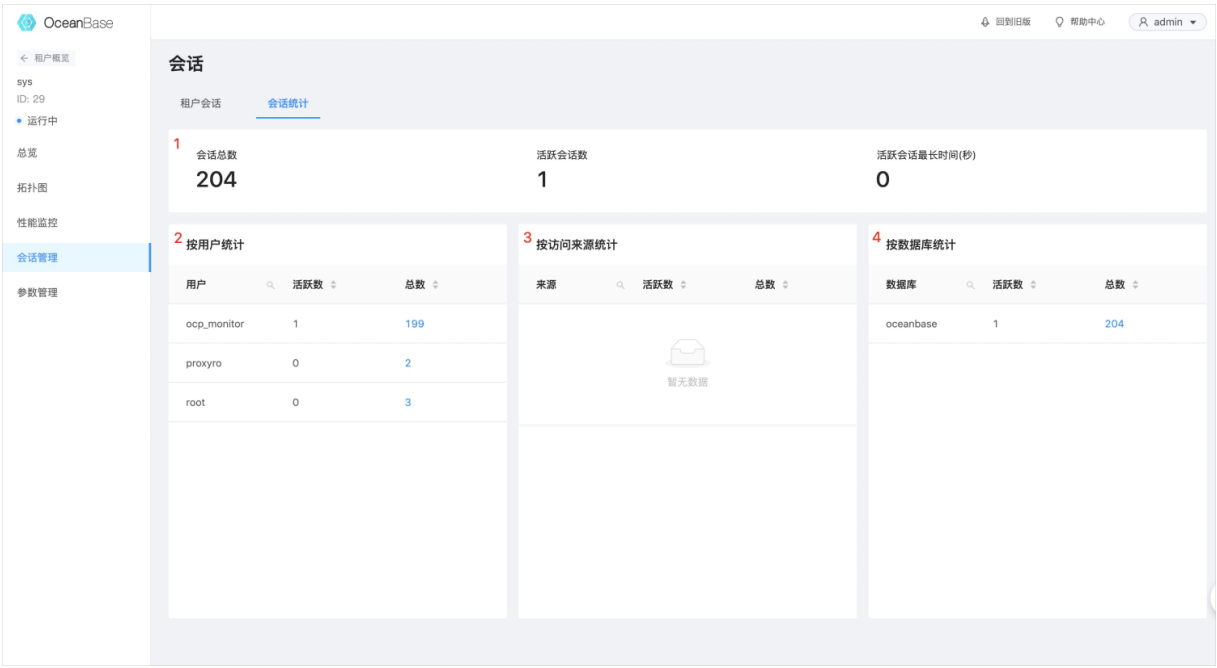


3.2.4.7.2. 查看会话统计

在租户 总览 页面左侧单击 会话管理，默认显示当前租户的所有会话列表。单击 会话统计 进入 会话统计 页签，页面中展示租户当前的所有会话的统计信息。

统计信息包括会话总体情况的统计，例如**会话总数**、**活跃会话数**、**活跃会话最长时间**等。支持按照**用户**、**访问来源**和**数据库**统计用户的**活跃会话数**和**会话总数**。

> **说明**
>
> 按访问来源统计功能仅支持 OceanBase 2.2.30 及以上版本。



3.2.4.8. 参数管理

3.2.4.8.1. 查看参数列表

通过租户参数列表，您可以查看当前租户所有参数的名称、类别、取值类型、取值范围、默认值、当前值、说明及是否重启生效等。列表上方的搜索框，支持以参数名进行模糊搜索。

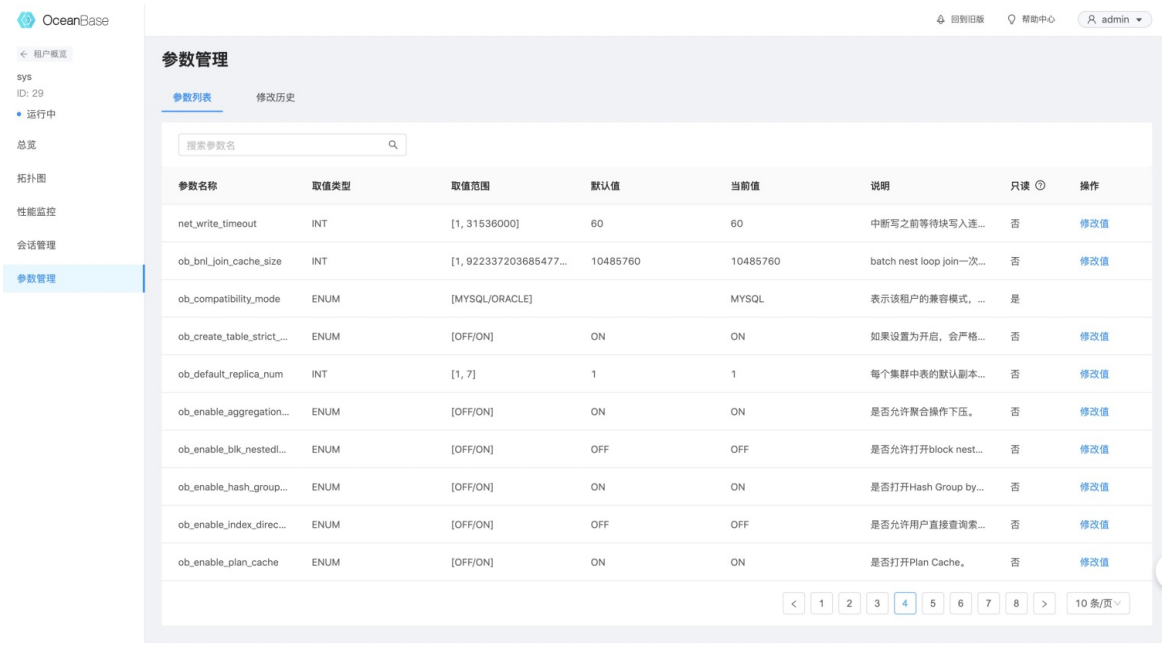
前提条件

检查密码箱中所存的该租户下root用户的密码是否正确，以及租户白名单设置是否正确。

操作步骤

1. 登录 OCP 。
2. 在左侧导航栏中单击**租户**，并在租户列表中单击租户名称。
3. 在显示的页面的左侧导航栏上，单击**参数管理**。
4. 在**参数列表**页面，可以查看当前集群所有参数的信息。
   - 展示信息包括各参数的参数名称、取值类型、取值范围、默认值、当前值、说明和是否只读等信息。

- **默认值**表示新建租户的缺省值。只读表示该参数是否可修改，如果是可修改的参数，在**操作**列中会展示修改值按钮。

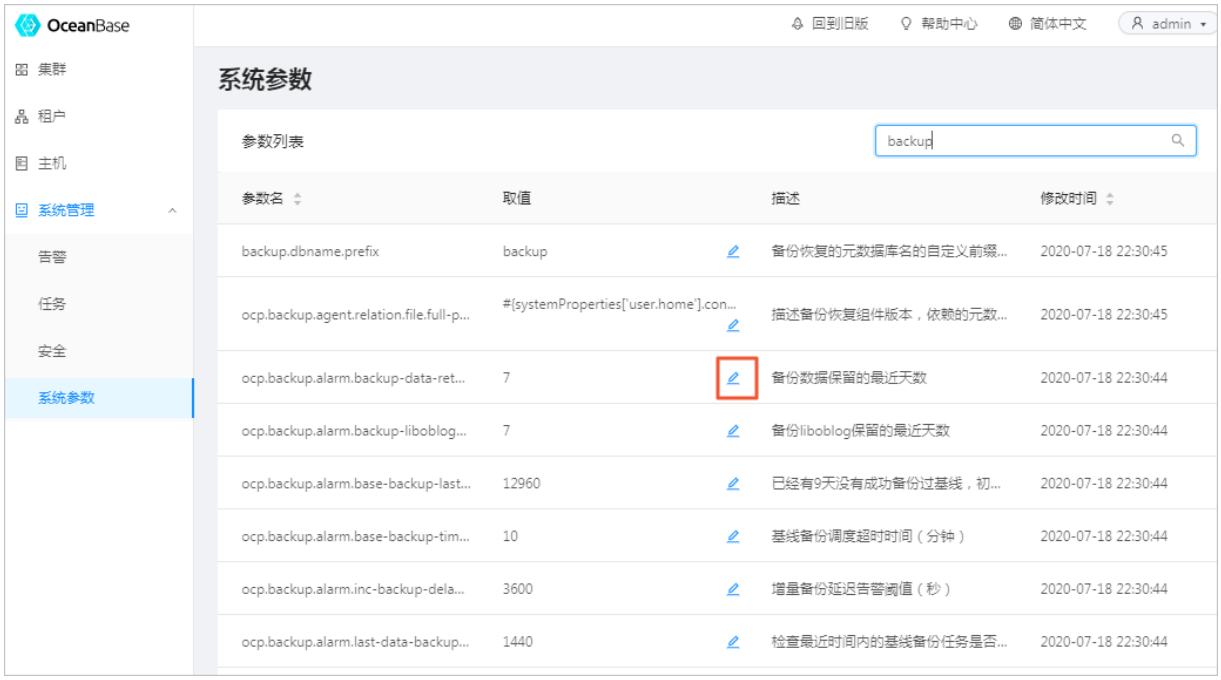


3.2.4.8.2. 修改参数

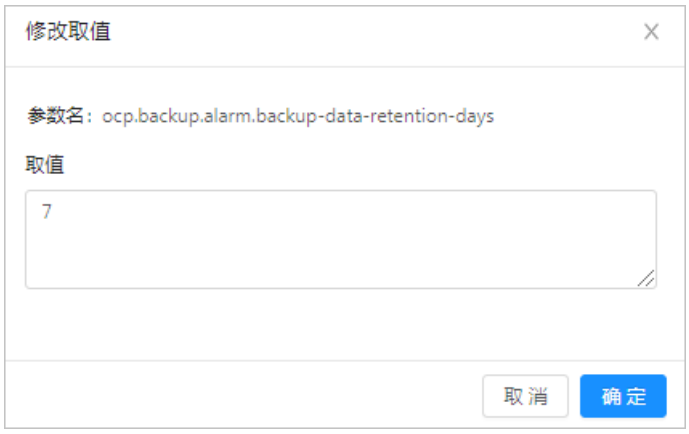
您可以根据需要修改租户参数的值，所有租户参数的值修改后均可实时生效。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在左侧导航栏上，单击**租户**。
3. 在**租户概览**页面的**租户列表**区域，选择待操作的租户并单击其租户名。
4. 在显示的页面的左侧导航栏上，单击**参数管理**。
5. （可选）在**参数列表**页面上方的搜索框中，输入参数名相关信息进行模糊搜索。
6. 找到待修改的参数，单击修改图标。



7. 在弹出的对话框中，修改参数的值。



8. 单击**确定**。

3.2.4.8.3. 查看参数修改历史

您可以在修改历史页面查看租户参数的历史修改信息，包括参数的名称、变更前的值、变更后的值、修改时间以及操作用户。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在左侧导航栏上，单击**租户**。
3. 在**租户概览**页面的**租户列表**区域，选择待操作的租户并单击其租户名。
4. 在显示的页面的左侧导航栏上，单击**参数管理**。
5. 单击**修改历史**页签。
6. 通过**修改日期**、**参数名称**或**操作用户**进行过滤查询。

其中，**参数名称** 支持模糊查询；**操作用户** 必须要精确匹配。



7. 在查询结果中，查看租户参数的历史修改信息。

3.3. 主机管理

3.3.1. 查看主机信息

主机概览

拥有系统管理员或主机管理员（HOST_MANAGER）角色的用户，可以单击页面左部导航中的**主机**进入主机管理页面。如下图所示，**主机列表**展示了当前所有主机信息，包括 IP 地址、SSH 端口、主机类型、物理机房、机型、所属 OB 集群、所属服务和状态。还可以进行分页查询，以及按照主机 IP、主机名和 OB 集群名进行搜索。

展开折叠页面，还可以查看主机 ID、主机名、操作系统、映射端口和主机说明。



主机状态

主机状态是主机当前监控和运行服务的重要标识。 当前版本 OCP 支持的主机状态包括：

- **新提交**：等待安装并启动 OCP-Agent
- **可用**：等待被其他服务使用
- **在线**：被一个或多个服务使用中
- **离线**：无法连接或者监控不到
- **删除中**：等待 OCP 清理主机上的配置和服务

主机类型

当前版本支持物理主机和容器两类主机：

- 单台物理主机混合部署多种服务，但是相同类型的服务只能部署一个，避免同类服务产生冲突。
- 单台容器主机，仅支持部署一种服务；根据 OceanBase 集群部署要求，不推荐部署 OceanBase 集群到容器类型主机。

容器端口映射

由于 OCP 服务端需要不定期向主机发起远程操作和监控，如果是容器类型的主机，需要基于 Docker 的“端口映射”方式：从 OCP 服务端访问宿主机的IP地址和端口，并映射到 Docker 容器的内部端口。

当添加容器类型的主机时，默认需要 4 组端口映射。即：容器内运行的 ocp-agent、node exporter 等进程的端口与宿主机端口的映射关系， 格式为： `<主机端口>:<docker端口>,<主机端口>:<docker端口>` 。

具体的映射关系如下：

运行服务	容器内端口号	是否跟宿主机端口一致	备注
OCP-Agent	62888	否	无
OBProxy监控	62881开始	否	从 62881 开始的第一个可用端口

3.3.2. 删除主机

在主机列表页面中，如果有不同状态的主机，部分状态的主机的 操作 列显示 删除 和 强制删除 的快捷操作，表示处于可用状态的主机，可以被正常删除；处于 新提交 和 离线 状态的主机，可以被强制删除。

> 注意
>
> - 如果强制删主机，可能会导致该主机相关的后台任务被打断。如果主机上有运行的服务和agent程序，可能会不能成功释放，因此一般建议将主机恢复到“可用”状态后，再进行删除。
> - 同时，强制删除后的主机，建议手工检查其运行的服务和进程，并进行必要的清理，如强行停止进程、移除安装包、移除文件等。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**主机**。
3. 在**主机列表**中找到需要强制删除的主机。
4. 在其对应的操作列单击**强制删除**。



5. 在弹出的确认框中单击**强制删除**。



3.4. 告警管理

3.4.1. 概述

OCP告警用于OCP监控的OceanBase和相关应用的故障告警和通知推送。

核心功能

- 基于告警规则生成告警
- 对大规模告警事件进行聚合以避免告警风暴
- 基于模板配置告警通知消息的格式和内容
- 配置化的通道集成，支持 HTTP 和 脚本方式

快速入门

告警和告警通知是OCP重要的状态监控功能，不同的角色在使用告警功能时需要采取不同的操作。

部署OCP之后，管理员需要对告警模块进行初始化配置，例如增加告警项、配置告警通道。管理员还需要根据业务需求，对已存在的告警项与告警通道进行管理。OCP用户在日常使用中，可以针对不同模块创建告警订阅、查看告警事件和通知，或者选择屏蔽告警等等。

OCP告警模块推荐配置流程为：

1. 管理员操作：
    i. 配置告警通道。

ii. 创建告警项。

2. 用户操作：

i. 订阅告警。

ii. 查看告警事件

iii. （可选）屏蔽告警。

3.4.2. 告警相关概念

本章节为您介绍告警对象、告警番位、告警项、告警、告警项分组、告警等级、模板、告警聚合和告警消除的概念。

告警对象

是告警任务所监控的主体，唯一标识一个告警对应的对象，可以是一个OB集群、一台机器或一个服务。

告警对象的格式取决于告警项的维度，可能是标签值或标签值的组合，如 `obregion=obocp:svr_ip=*.*.*.*` 标识 OceanBase OCP集群上的一台服务器。

告警范围

告警范围用于定义某个告警对应的范围，和指标范围保持一致。例如，CPU使用率超限可能是集群整体使用率超限、租户CPU使用率超限、机器使用率超限等不同范围的含义。

告警范围包括以下值：

- OB集群 ObCluster
- OB租户 ObTenant
- 应用集群 AppCluster
- 服务 Service
- 服务器主机 Host
- 进程 Process （预留类型）

告警项

告警项是告警的元数据，包括告警类型、名称、触发规则、告警概述和详情的模板等。

告警项按照生成方式可以分为2种类型：

- **表达式触发**

  ：控制台上创建的告警，告警告警规则引擎基于监控指标生成。

- **自定义触发**

  ：告警由其它组件自动触发。

对于表达式触发的告警，其告警规则表达式在告警项里配置；对于自定义触发的告警，告警规则表达式为空。

告警

告警是一个告警项在一个告警对象上发生的状态。

例如，告警对象 机器A 产生告警项 alarm_b，在故障期间每分钟触发一个告警信号，但是在OCP这属于一个告警。在告警事件页面只有一条记录。

告警项分组

每个告警项可以设置多个分组，分组的目的是为了方便配置告警订阅。

告警等级

每个告警项都有对应的告警级别。

级别	英文含义	中文含义	颜色	说明
1	Down	停服	紫	完全不可用，需要立即介入修复，如 `OB服务启动失败`
2	Critical	严重	红	系统可用性下降，需要紧急修复，避免完全不可用， 如 `机器内存使用率大于90%持续3分钟`
3	Alert	警告	橙	系统仍然可用，但是即将达到不可用状态，需要采取措施防止可用性下降， 如 `OB租户连接数大于上限的80%`

级别	英文含义	中文含义	颜色	说明
4	Caution	注意	蓝	根据趋势判断，系统关键性能指标正在下降，但是还没达到触发警告的程度，此时可以通过排查发现潜在的问题，避免警告产生（保留类型，目前尚未有匹配的告警项）
5	Info	提醒	绿	操作提醒，本质上不是告警，通常是管理员执行重要操作，如 `管理员执行集群下线操作`

模板

模板用于运行期根据变量生成动态内容，可以用于：

- 告警生成 （在告警项中配置 告警概述模板、告警详情模板）
- 通知内容（通道中的 消息模板 和 聚合消息模板）
- 通道参数 （如 HTTP通道配置 Header 和 Body的内容模板）

模板样例：

- alarm_summary:
  `${alarm_targe} CPU使用率超限`
- alarm_description: `${alarm_targe} CPU使用率持续 ${alarm_duration} 超过 ${alarm_threshold}`

支持的模板变量参见本文档附录4. OCP 告警模板变量。

告警聚合

为避免告警数量过多导致的告警风暴，告警通道支持配置聚合。

聚合规则为：

- OB 日志告警，按照告警类型、日志错误码、OB 集群进行聚合。
- 其它 OB 告警，按照告警类型、OB 集群进行聚合。

- 应用告警，按照告警类型、告警对象进行聚合。

告警消除

告警消除的概念就是故障恢复。故障恢复的感知可能通过监控模块确定故障消除了通知告警服务，也可能由告警服务基于告警消除超时自动实现。

告警超时自动消除逻辑：

- 每个告警项有检查周期，和忽略周期。
- 新的检查周期监控发现告警项已经满足消除条件，调用告警接口把告警项置为已消除。
- 到达忽略周期后，如果告警项没有继续报出来，认为告警项已消除。

3.4.3. 告警项配置

3.4.3.1. 新建告警项

本文档为您展示创建告警项的操作步骤。

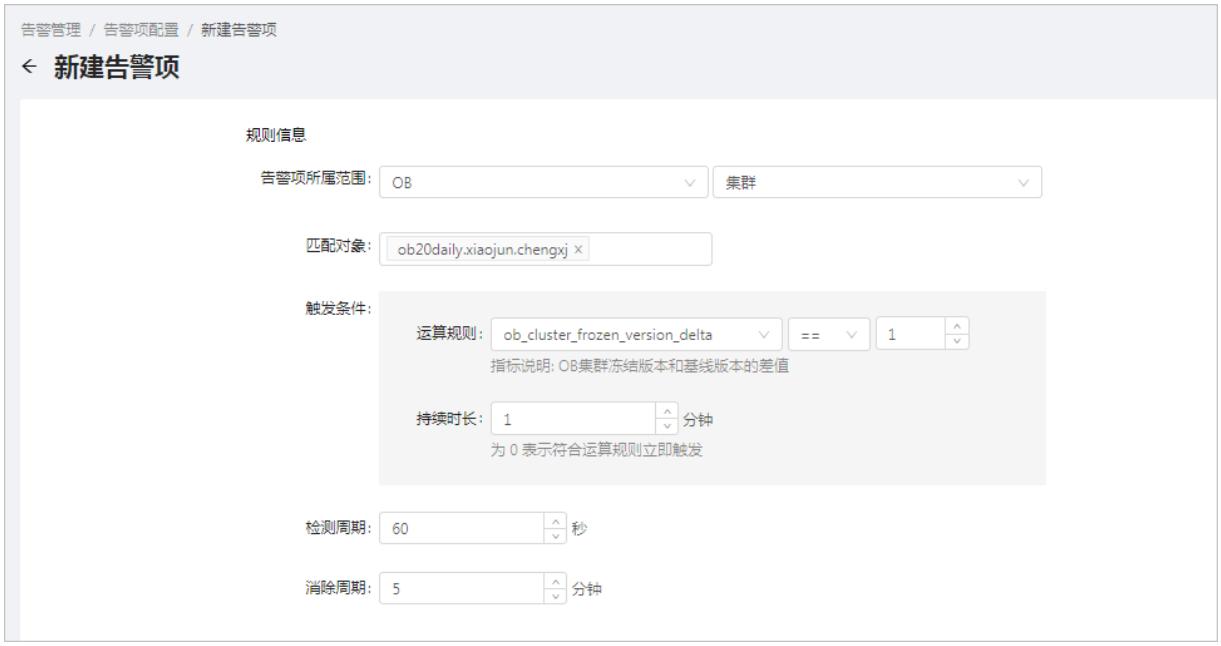
前提条件

当前登录用户具备系统管理员角色、OCP 告警管理角色。

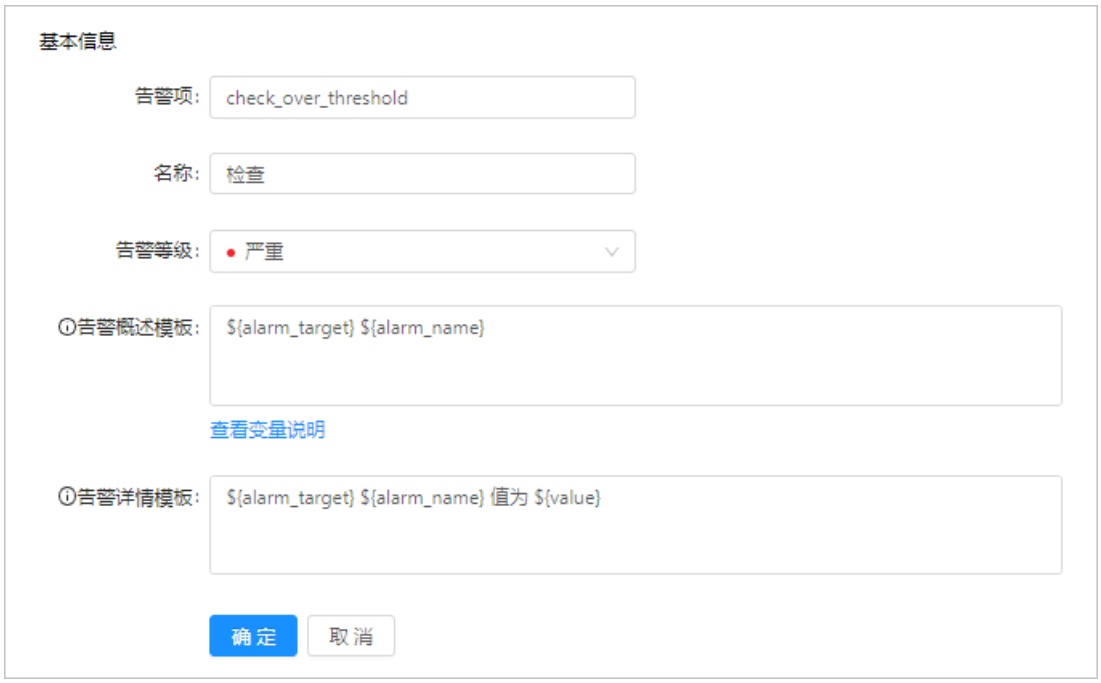
操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理>告警**。
3. 在告警项配置页签右上角单击**新建告警项**。
4. 配置告警项的规则信息。
   i. 设置告警项所属范围。
   ii. 设置匹配对象，确定告警规则适用的对象。
   iii. 设置触发条件。
   - 根据告警项所属范围的不同，可以在运算规则里选择不同的指标。
   - 持续时长为0表示立即触发，可以设定时长，用于避免因指标毛刺产生的告警误报。

   iv. 设置检测周期和消除周期。



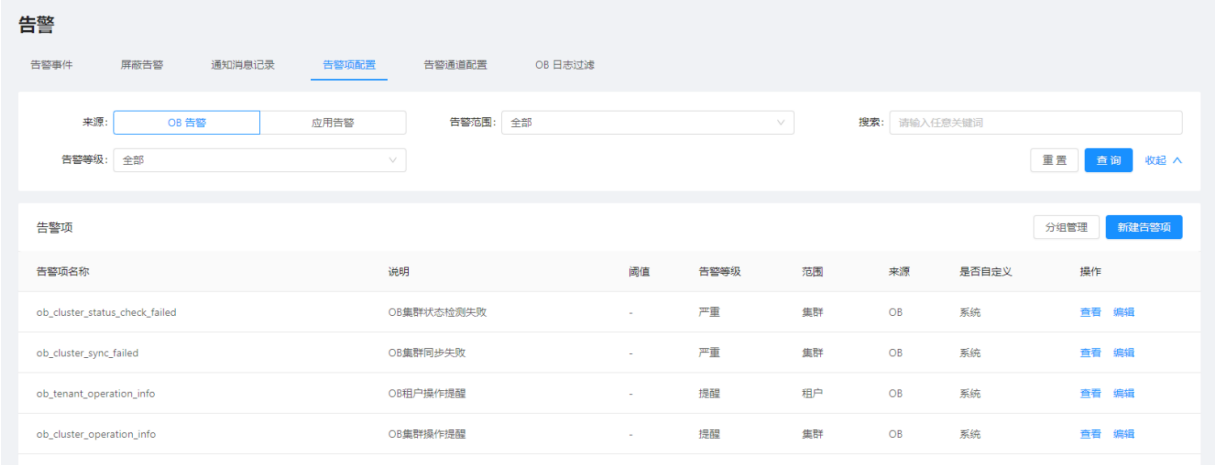
5. 配置告警项的基本信息。
   i. 设置告警项名称和中文名称。
   ii. 设置告警等级。
   iii. 根据页面提示设置告警概述模板。
   iv. 根据页面提示设置告警详情模板。
   - 告警概述模板用于在通道的模板字段中引用，对应的变量名为 alarm_summary。
   - 告警详情模板用于在通道的模板字段中引用，对应的变量名为 alarm_description。



6. 单击**确定**。

3.4.3.2. 查看告警项

在左侧导航栏中单击 系统管理 > 告警，在 告警项配置 页签中可以查看当前所有的告警项。



告警项列表每一行显示一个告警项，最右侧**操作**列可以对告警项进行查看、编辑或删除操作。

> **说明**
>
> 系统告警项无法删除，编辑操作也只能调整有限的配置项，如阈值、持续时长和告警等级。

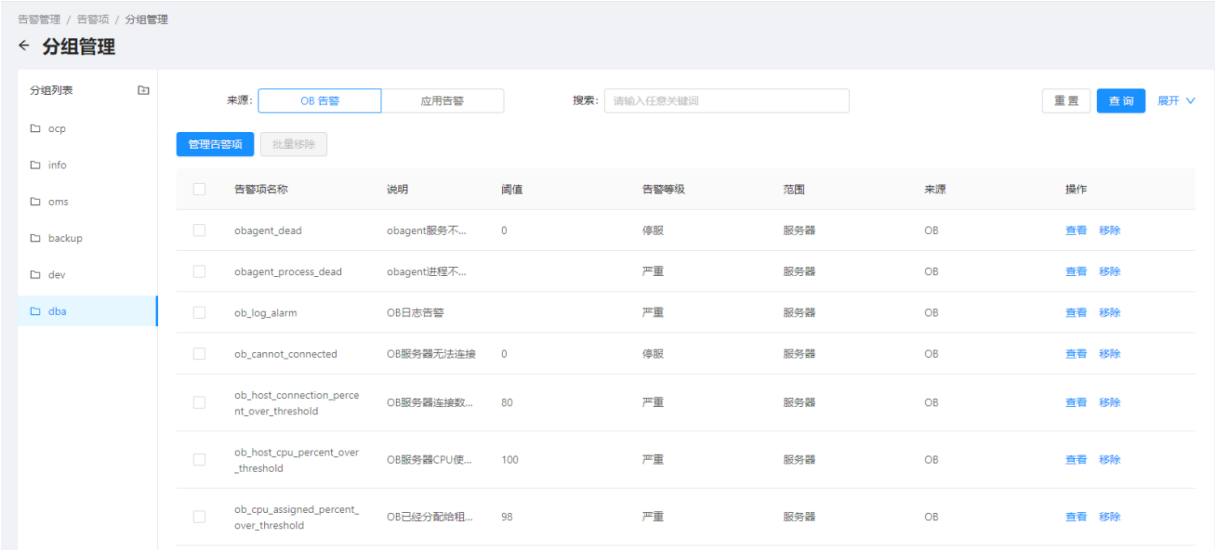
单击**查看**，可以查看告警项的详细配置。



3.4.3.3. 告警分组

为了方便配置告警订阅，OCP 支持告警分组功能，可以对每个告警项进行简单分类。

在左侧导航栏中单击**系统管理 > 告警**，并于**告警项配置**页签中单击**分组管理**，可以查看当前所有的告警分组。单击分组名称可以查看各个分组中的告警项。

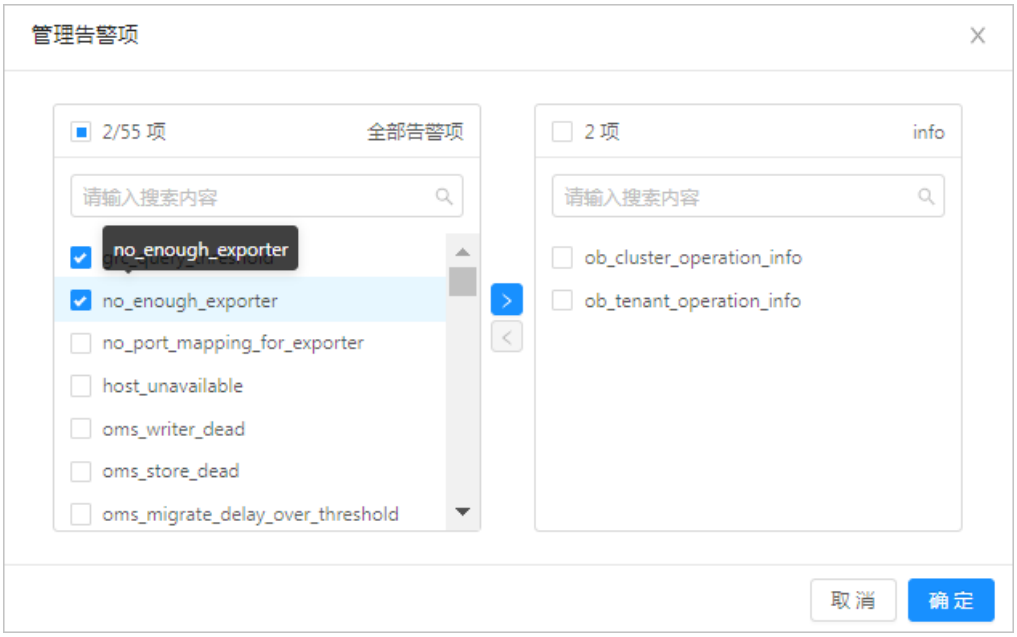


管理分组

- 告警项检索：OCP 支持多种告警项检索方式，例如通过**来源**、**关键词**、**应用**、**告警等级**和**告警范围**搜索告警项。



- 为告警项分组：单击左侧告警分组名称进入指定分组，并单击**管理告警项**，将左侧的告警项添加到右侧。



- 移除分组中的告警项：单击左侧告警分组名称进入指定分组，选择需要从该组移除的告警项，并单击**批量移除**。

> **说明**
>
> 该操作只是从分组中移除告警项，并不会删除告警项本身。



- 创建自定义分组：单击分组列表右侧的创建图标，输入新分组的名称。



- 修改分组名称：展开指定分组右侧的隐藏菜单，并单击**重命名**。



- 删除分组展开指定分组右侧的隐藏菜单，并单击**删除**。

默认分组

系统内建 4 个初始化分组，系统内建分组可以编辑，无法删除。

分组名称	分组描述
dba	数据库管理员关注的告警项
dev	开发者关注的告警项
oms	OMS应用管理员关注的告警项
backup	备份恢复管理员关注的告警项
info	提醒信息，如运维操作结果提醒
ocp	OCP基础模块告警项，如主机告警

3.4.3.4. 编辑告警项

根据业务需求，系统管理员和 OCP 告警管理员可以修改告警项。

前提条件

当前登录用户具备系统管理员角色、OCP 告警管理角色。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 告警**。
3. 在**告警项配置**页签中找到指定告警项。
4. 重新配置告警项的规则信息。
   i. 设置**告警项所属范围**。
   ii. 设置**匹配对象**，确定告警规则适用的对象。
   iii. 设置**触发条件**。
   - 根据告警项所属范围的不同，可以在运算规则里选择不同的指标。
   - 持续时长为 **0** 表示立即触发，可以设定时长，用于避免因指标毛刺产生的告警误报。

   iv. 设置**检测周期**和**消除周期**。



5. 配置告警项的**基本信息**。
   i. 设置**告警项名称**和中文**名称**。
   ii. 设置**告警等级**。
   iii. 根据页面提示设置**告警概述模板**。
   iv. 根据页面提示设置**告警详情模板**。
   - 告警概述模板用于在通道的模板字段中引用，对应的变量名为 alarm_summary。
   - 告警详情模板用于在通道的模板字段中引用，对应的变量名为 alarm_description。



6. 单击**确定**。

3.4.3.5. 删除告警项

根据业务需求，系统管理员和 OCP 告警管理员可以删除告警项。系统预设的告警项不可删除。

前提条件

- 当前登录用户具备系统管理员角色、OCP 告警管理角色。
- 需要删除的告警项是自定义告警项，不是系统预设的告警项。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 告警**。
3. 在**告警项配置**页签中找到指定告警项。
4. 在其对应的操作列，单击**删除**。



5. 在弹出对话框中单击**确定**。



3.4.4. 告警通道配置

3.4.4.1. 通道配置说明

告警本身是独立的功能，未配置通道和订阅时，只能通过控制台的告警事件页面查看告警。通过配置告警通道和告警订阅，用户可以接收告警通知消息。

告警通道是通知消息发送的通道，支持以下通道类型：

- HTTP
- 自定义脚本

> **说明**
>
> 告警聚合的时间参数在通道进行配置，更新告警通道之后，该通道会重新开始聚合计算。如果此时有有效的告警项，可能会更快收到告警通知。

3.4.4.2. 新建告警通道

本章节为您介绍如何新建告警通道。

前提条件

- 已提前配置接收用户的基本信息。

- 如果需要新建自定义脚本的告警通道且**配置方式**为**配置脚本文件名称**，请确认已将该自定义脚本文件存放在每个 OCP 服务节点的 `/home/admin` 目录下。

对该自定义脚本的脚本内容的要求如下：

- 当前仅支持 Bash 或 Python 脚本。
- 脚本内容的第一行需包含 **Shebang**（#！）指定执行程序。自定义脚本配置示例请参见本文档中**告警管理** > **告警通道配置** > 配置样例章节的内容。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理** > **告警**。
3. 在**告警通道配置**页面单击**新建通道**。
4. 设置通道的基本信息，并单击**下一步**。



基本信息项	说明
通道名称	用户可自定义填写通道名称
接收人字段	• 接收人字段可设置为 **用户ID**、**邮箱**、**手机号** 或 **员工号** 。 • 接收人字段和 OCP 用户的基本信息关联。
通道类型	用户可选择 **HTTP** 或**自定义脚本**两种不同类型的通道，

基本信息项	说明
是否群消息通道	• 如果是群消息通道，那么当无人订阅时，系统也会推送通知。 • 如果是非群消息通道，会根据接收人字段推送给订阅对应告警的用户。

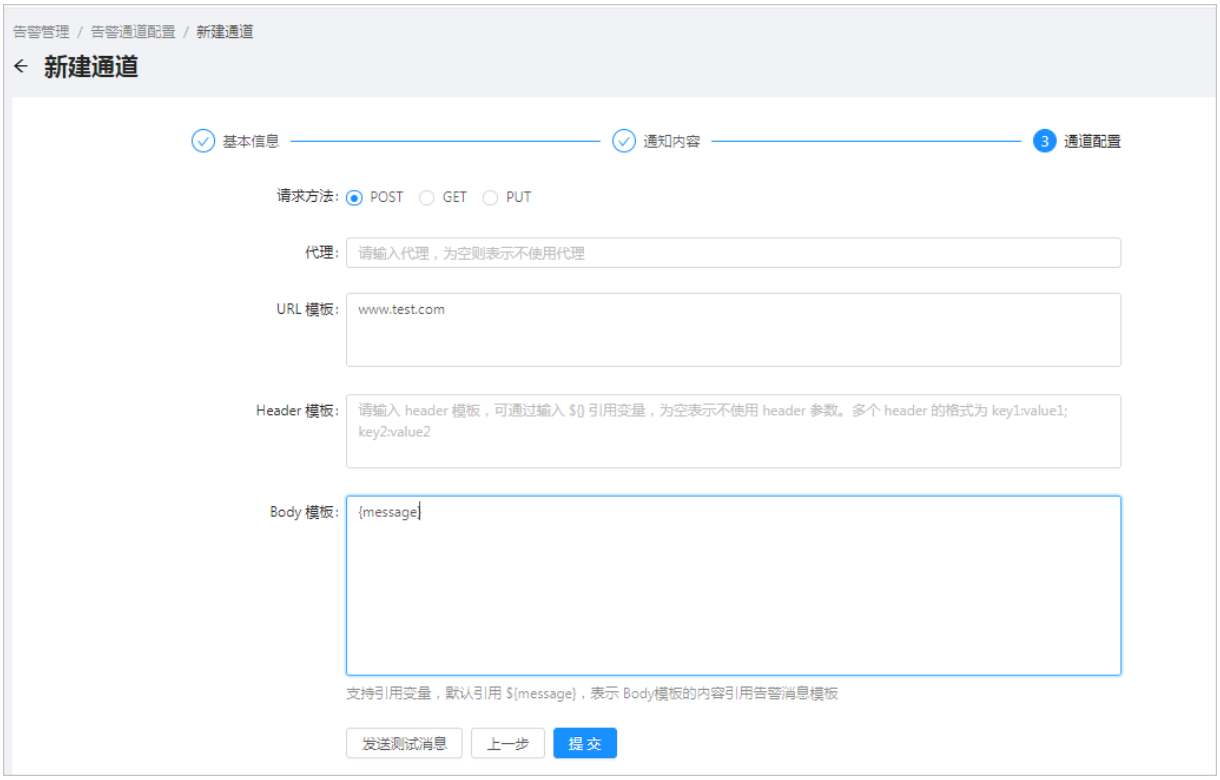
5. 设置**通知内容**并单击**下一步**。



通知内容填写项	说明
告警消息模板	• 支持带变量的告警消息模板。 • 您可以参考页面提示重新自定义告警。

通知内容填写项	说明
告警消息聚合	• 告警消息聚合可以避免在短时间内收到过多的消息，导致重要消息被淹没。您可以根据页面提示查看聚合规则。 • 如果选择开启消息聚合，还需要配置聚合消息模板等相关参数。

6. 进行通道配置。需要根据选择的通道类型，配置不同的参数。
   - 如果选择的**通道类型**为 **HTTP**，则参考以下操作进行配置。



通道配置项	说明
请求方法	当前支持 Post、Get 和 Put 方式发送请求。
代理	为空表示不使用代理。
URL模板	可通过输入 $ 和 {} 引用变量。
Header模板	可通过输入 $ 和 {} 引用变量，为空则表示不适用 Header 参数。

通道配置项	说明
Body模板	支持引用变量，默认引用 **${message}**，表示 Body 模板的内容引用告警消息模板。

- 如果选择的**通道类型**为**自定义脚本**，则参考以下操作进行配置。默认会选择**配置方式**为**配置脚本内容**，输入脚本内容。

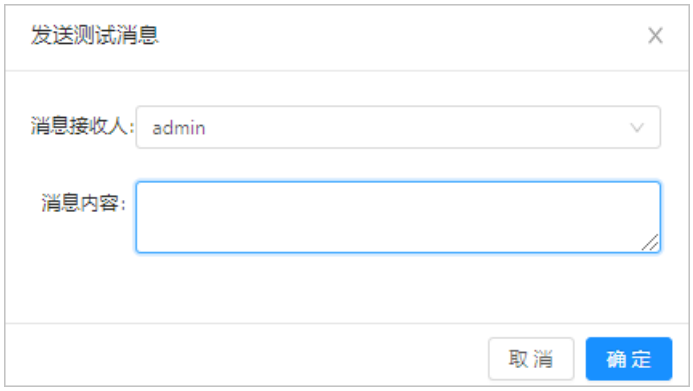


如果脚本文件已提前放入指定的 `/home/admin` 目录，您也可以选择**配置方式**为**配置脚本文件名称**，并输入脚本文件名。

- 脚本文件名称只能由字母、数字、下划线（_）、圆点（.）组成，且符号均为半角符号。
- 为了避免脚本名称字段引入 shell 注入安全风险，脚本名称字段中不能包含空格、制表符等符号。



7. 配置完成后，单击**发送测试消息**，并选择消息接收人，输入消息内容后，单击**确定**。如果消息发送成功，可以直接单击**提交**。OCP 在脚本通道发送通知时，会调用所配置的脚本，并把通知相关的所有变量通过环境变量传递到脚本中。通知相关的变量信息请参见附录4. OCP 告警模板变量。



3.4.4.3. 查看告警通道

本章节为您介绍如何查看告警通道。

告警通道列表

告警通道列表中可以查看所有通道，管理员也可以进行编辑、删除等操作。



默认通道

OCP 中预设默认通道，用于运维任务自动创建的订阅。其中：

- 默认通道无法删除。
- 默认通道需要经过配置才能使用。

查看默认通道配置：



3.4.4.4. 编辑告警通道

出于业务需求，创建告警通道后，管理员可以修改告警通道的基本信息、通知内容和通道配置。

前提条件

已创建告警通道。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 告警**。
3. 在**告警通道配置**页面找到需要修改的通道。
4. 在其对应的**操作**列单击**编辑**。
5. 重新设置通道的**基本信息**。
   i. 设置**通道名称**。
   ii. 设置**接收人字段**。
   - 接收人字段可设置为**用户ID**、**邮箱**、**手机号**或**员工号**。
   - 接收人字段和 OCP 用户的基本信息关联。

   iii. 设置通道类型。
   - **HTTP**
   - **自定义脚本**

   iv. 是否为**群消息通道**。
   - 如果是群消息通道，那么当无人订阅时，系统也会推送通知。

- 如果是非群消息通道，会根据接收人字段推送给订阅对应告警的用户。

v. 单击 **下一步**。



5. 重新设置**通知内容**。

i. 填写**告警消息模板**。

- 支持带变量的告警消息模板。
- 您可以参考页面提示重新自定义告警消息模板。

ii. 设置是否**开启告警消息聚合**。

- 告警消息聚合可以避免在短时间内收到过多的消息，导致重要消息被淹没。您可以根据页面提示查看聚合规则。
- 如果选择开启消息聚合，还需要配置聚合消息模板等相关参数。

iii. 单击**下一步**。



6. 重新设置**通道配置**。

需要根据选择的通道类型，配置不同的参数。

如果选择的**通道类型**为 **HTTP**，则参考以下操作进行重新配置：

i. 设置**请求方法**。当前支持 Post、Get 和 Put方式发送请求。

ii. 设置**代理**。为空表示不使用代理。

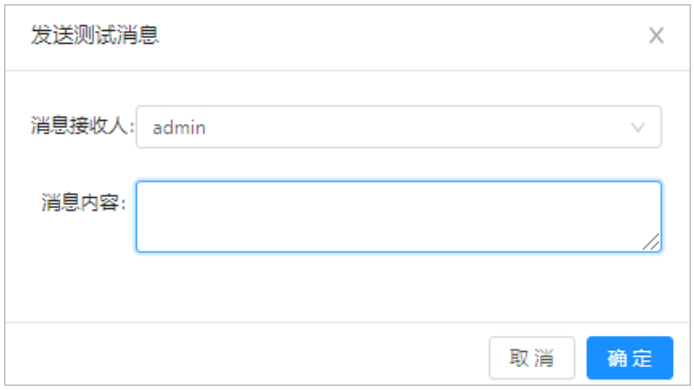
iii. 设置**URL模板**。可通过输入 **$** 和 **{}** 引用变量。

iv. 设置 **Header 模板**。可通过输入 **$** 和 **{}** 引用变量，为空表示不适用 Header 参数。

v. 设置 **Body 模板**。支持引用变量，默认引用 **${message}**，表示 Body 模板的内容引用告警消息模板。



vi. 单击**发送测试消息**，并选择消息接收人，输入消息内容后，单击**确定**。如果消息发送成功，可以直接单击**提交**。



如果选择的**通道类型**为**自定义脚本**，则参考以下操作进行重新配置：

i. 选择脚本通道的配置方式。如果配置方式为**配置脚本内容**，则重新输入脚本内容即可。对脚本内容的要求如下：

- 当前仅支持 bash 或 python 脚本。
- 脚本内容的第一行需包含 shebang 指定执行程序。

自定义脚本配置示例请参见本文档中**告警通道配置** > **配置样例**章节的内容。

如果**配置方式**为**配置脚本文件名称**，则按要求重新设置脚本文件名即可。

- 脚本文件名称只能由字母、数字、下划线（_）、圆点（.）组成，且符号均为半角符号。
- 为了避免脚本名称字段引入 shell 注入安全风险，脚本名称字段中不能包含空格、制表符等符号。

b. 单击**发送测试消息**，并选择消息接收人，输入消息内容后，单击**确定**。

c. 如果发送成功，可以直接单击**提交**。

OCP 在脚本通道发送通知时，会调用所配置的脚本，并把通知相关的所有变量通过环境变量传递到脚本中。通知相关的变量信息请参见附录4. OCP 告警模板变量。

3.4.4.5. 删除告警通道

出于业务需求，管理员可以删除自定义创建的告警通道。

前提条件

- 需要删除的通道非默认通道。
- 当前登录用户具备系统管理员或 OCP 告警管理角色。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 告警**。
3. 在**告警通道配置**页面找到需要删除的通道。
4. 在其对应的**操作**列单击**删除**。



5. 在弹出的对话框中单击**确定**。



3.4.4.6. 配置样例

本文档为您列举常见的告警通道配置样例，供您参考。

钉钉群消息

可以基于钉钉群机器人的 Webhook 实现群消息发送，群机器人介绍参见 钉钉群机器人官方文档 。

配置步骤

1. 在钉钉群中配置机器人。在钉钉群中配置群机器人，需要记录Webhook URL。目前钉钉群自定义机器人需要进行安全配置才可以使用，OCP告警HTTP通道仅支持 **自定义关键词** 和 **IP地址（段）**安全设置。钉钉机器人的安全设置界面如下图所示。



   - 设置**自定义关键词。**设置的自定义关键词和 OCP 告警通道配置的消息模板匹配即可，如果默认模板的第一行内容包含“OCP告警通知”，可以配置自定义关键词为 “OCP告警”。
   - 设置 **IP 地址（段）。**这里需要注意，设置的 IP 地址必须是 OCP 服务器的公网IP，可使用命令 curl -w "\n\n" -s http://whatismijnip.nl 查看服务器公网 IP。由于服务器可能没有固定的公网 IP，请联系您的网络管理员获取服务器的公网 IP 网段。
2. 在告警通道中新建通道，通道类型选择 **HTTP**，接收人字段选择**手机号**，是否群消息通道勾选**是**。
3. 配置通知内容。
   - 消息模板

```
OCP告警通知-单条告警
- 名称：${alarm_name}
- 级别：${alarm_level}
- 告警对象：${alarm_target}
- 概述：${alarm_summary}
- 生成时间: ${alarm_active_at}
- 详情：${alarm_description}
- OCP链接：${alarm_url}
```

   - 聚合消息模板

```
OCP告警通知-多条告警
- 名称：${alarm_name}
- 级别：${alarm_level}
- 告警数量：${alarm_count}
- 聚合分组：${alarm_group_by}
- 告警对象：${alarm_target}
- 生成时间: ${alarm_active_at}
```

4. 配置通道
   - 请求方法：POST
   - Header 模板：`Content-Type:application/json; charset=utf-8`

     > **说明**
     >
     > URL模板即为步骤1中记录的Webhook URL。

   - Header 模板：`Content-Type:application/json; charset=utf-8`
   - Body 模板：

```
{
 "msgtype": "text",
 "text": {
  "content": ${message_json}
 },
 "at": {
  "atMobiles": ${recipients_json_array},
  "isAtAll": false
 }
}
```

> **说明**
>
> 这里 Body 的格式为 json，引用 `_json` 和 `json_array` 结尾的变量时，不使用 `"` 括起来，因为变量值已经加了 json 序列化处理，加上 `"` 会导致 json 格式错误。 作为对比，引用非json变量时需要使用 `"` 括起来，例如 `"{message}"` 。

钉钉群消息 markdown 格式

钉钉消息支持 markdown，以下样例借助 markdown 的格式化能力配置富文本消息。 配置方式和文本格式的消息是一样的，差别在于几个模板的配置。 比如通过引用

`${alarm_level_color}`

实现消息字体颜色的配置（告警等级对应颜色参见 告警等级）。

钉钉支持消息类型参见 钉钉开发文档。

- 消息模板

```
<font color=${alarm_level_color}>OCP聚合通知-单条告警</font>
> - 名称：`${alarm_name}`
> - 级别：`${alarm_level}`
> - 告警对象：`${alarm_target}`
> - 概述：`${alarm_summary}`
> - 生成时间:`${alarm_active_at}`
> - 详情：`${alarm_description}`
> - [到OCP查看](${alarm_url})
```

- 聚合消息模板

```
<font color=${alarm_level_color}>OCP聚合通知-多条告警</font>
> - 名称：`${alarm_name}`
> - 级别：`${alarm_level}`
> - 告警数量：`${alarm_count}`
> - 聚合分组：`${alarm_group_by}`
> - 告警对象：`${alarm_target}`
> - 生成时间:`${alarm_active_at}`
```

- Body 模板

```
{
  "msgtype": "markdown",
  "markdown": {
    "title": "OCP告警钉钉消息markdown测试",
    "text": ${message_json}
  }
}
```

自定义脚本通道

自定义脚本通道，是为了满足特殊的告警对接场景而设计的。一般用于无法通过 http 的方式配置或者对告警格式有特殊要求的场景下，您可以在脚本中定制一些特殊的逻辑。比如：告警时间的转换，对第三方的告警网关的默认参数的添加等，均可以通过自定义脚本通道来实现。

1. 配置基本信息。通道类型选择**自定义脚本**，其他选项根据具体需求进行选择。
2. 配置通知内容。
   - 消息模板

```
[OCP告警通知]
- 名称：${alarm_name}
- 级别：${alarm_level}
- 告警对象：${alarm_target}
- 概述：${alarm_summary}
- 生成时间: ${alarm_active_at}
- 详情：${alarm_description}
- 链接：${alarm_url}
```

- 聚合消息模板

```
[OCP告警通知]
- 名称：${alarm_name}
- 级别：${alarm_level}
- 告警数量：${alarm_count}
- 聚合分组：${alarm_group_by}
- 告警对象：${alarm_target}
- 生成时间: ${alarm_active_at}
```

3. 进行通道配置。

- 脚本文件名：进行通道配置时选择

  配置方式

  为

  配置脚本文件名称

  。

系统默认有一个自定义脚本样例 alarm_send_script_demo.sh，您可以在其基础上进行修改，也可以新增自定义的脚本。在修改或新增文件时，请确保该脚本文件已存放在每个 OCP 的 /home/admin 目录下。

当前仅支持 bash 或 python 脚本。

- **shell 脚本示例**

发送钉钉群消息：

```
#!/bin/sh

echo "Below is a list of alarm paras"

# below variables can be referenced by prefix "$", for example, $alarm_name or ${alarm_name}
echo "alarm_name:$alarm_name"
echo "app_type:$app_type"
echo "alarm_threshold:$alarm_threshold"
echo "alarm_time=$alarm_time"
echo "alarm_last_interval:$alarm_last_interval"
echo "alarm_time:$alarm_time"
echo "alarm_level:$alarm_level"
echo "alarm_type:$alarm_type"
echo "alarm_summary:$alarm_summary"
echo "alarm_url:$alarm_url"
echo "app:$app"
echo "alarm_duration:$alarm_duration"
echo "alarm_status:$alarm_status"
echo "alarm_scope:$alarm_scope"
echo "alarm_active_at:$alarm_active_at"
echo "alarm_target:$alarm_target"
echo "alarm_description:$alarm_description"
echo "message:$message"
echo "receiver:$receiver"
echo "alarm_id:$alarm_id"

# this function defines to how to assembly request by yourself according to your requirements
# this demo shows you how to send alarm to ding ding
function send(){
  # this token is ding ding group token, please apply and assign it to variable token
  token=''
  # URL="https://oapi.dingtalk.com/robot/send?access_token=$token"
  URL=''
  curl -X POST ${URL} -H 'Content-Type: application/json' -d '{"msgtype":"text","text":{"content":"""${messa
ge}"""}}'
}

# invoke function to
send
```

- **python 脚本样例**

输出告警信息到文本文件：

```
#!python
# -*- coding: utf-8 -*-
import os

#获取环境变量的值
message = os.environ['message']
print "hello {}".format(message)
with open("./sample.txt", "a") as file_object:
  file_object.write("{}".format(message))
```

进行脚本编写时，应注意以下事项：

- 脚本的返回码 exitCode 会用于判断执行是否成功，

  `exitCode==0`

  表示执行成功，非 0 表示执行失败。
- 如果脚本执行失败，可以将错误信息输出到 stderr，OCP 会记录下来用于后续的问题排查。
- 脚本应尽可能快速执行完成，建议执行耗时控制在 1 秒钟以内，OCP 对调用脚本有超时设计，默认超时时间为 10 秒，如果10 秒未返回，OCP 会强制停止脚本进程。
- 脚本应避免采用多进程编程的方式，避免因 OCP 停止脚本进程出错而产生脚本执行的进程堆积。

- 脚本内容：进行通道配置时选择

  配置方式

  为

  配置脚本内容

  。

自定义脚本的通道配置，除了脚本文件名方式，同时还支持直接配置脚本内容。您可以将 bash 或 python 脚本内容直接输入在**脚本内容**文本框中。

4. 发送测试消息。

调试脚本时，可以使用发送测试消息来测试，默认消息内容使用脚本中的变量 ${message}，请确保该变量在发送的消息体中。由于自定义脚本发送成功的判断逻辑需要视具体情况而定，系统认为只要脚本执行成功，即代表发送成功。不过，在浏览器的开发者模式中，用户可以根据发送测试消息对应的 _test 请求的 Response 内容进行调试。

3.4.5. 告警订阅配置

默认情况下，告警只能在告警页面查看。管理员配置成功告警通道之后，用户才能订阅告警。

用户能够接收告警通知需满足2个条件：

1. 管理员配置告警通道
2. 用户订阅告警

配置告警发送的步骤：

1. 配置通道。管理员配置告警通道。
2. 新建订阅。用户可以配置自己的订阅。
3. 个人接收配置。用户可以配置是否接收通知以及接收限制。

查看告警订阅列表

登录OCP之后，在右上角个人中心处展开隐藏列表，并单击**告警订阅**。在订阅列表页签中可以根据**来源**和**集群**筛选告警。

页面中默认展示**来源**、**订阅集群**、**告警项组**、**订阅人**和**订阅时间**。



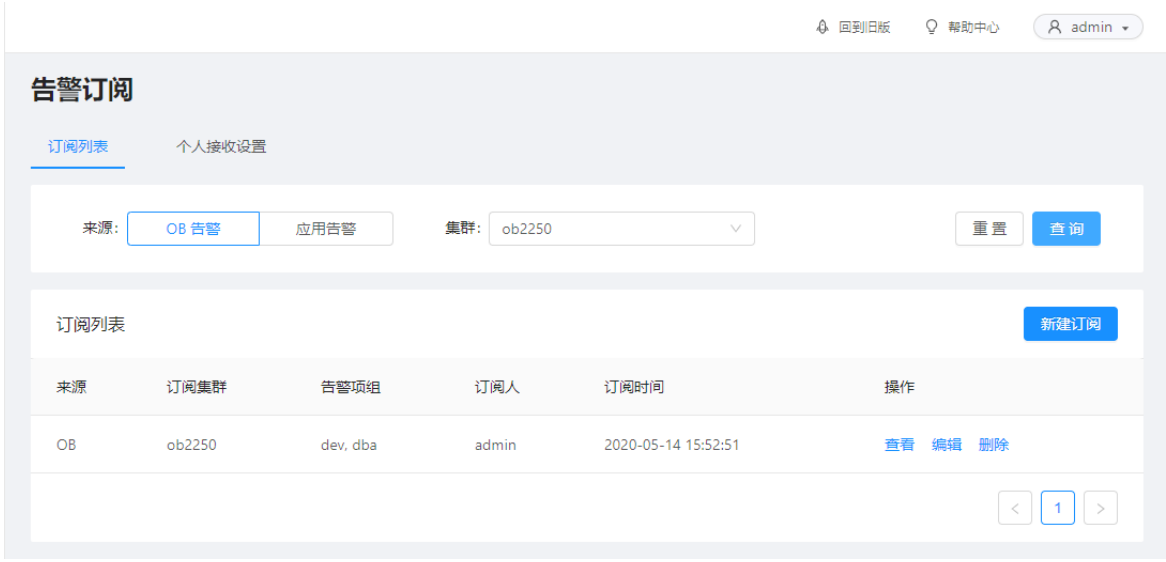
新建告警

1. 登录OCP。
2. 在右上角个人中心处展开隐藏列表，并单击**告警订阅**。
3. 单击**新建订阅**。
4. 根据业务需求，选择 **OB** 或者**应用**页签，并选择 OB 集群或指定应用。
5. 选择订阅的**告警项分组**。

6. 设置**告警方式**。为每个告警等级设置告警通道。每个等级可以同时配置多个通道，至少配置一个告警等级的通道。



7. 单击**确定**。配置好告警订阅之后，可以在**订阅列表**页面查看详情、编辑或删除订阅。

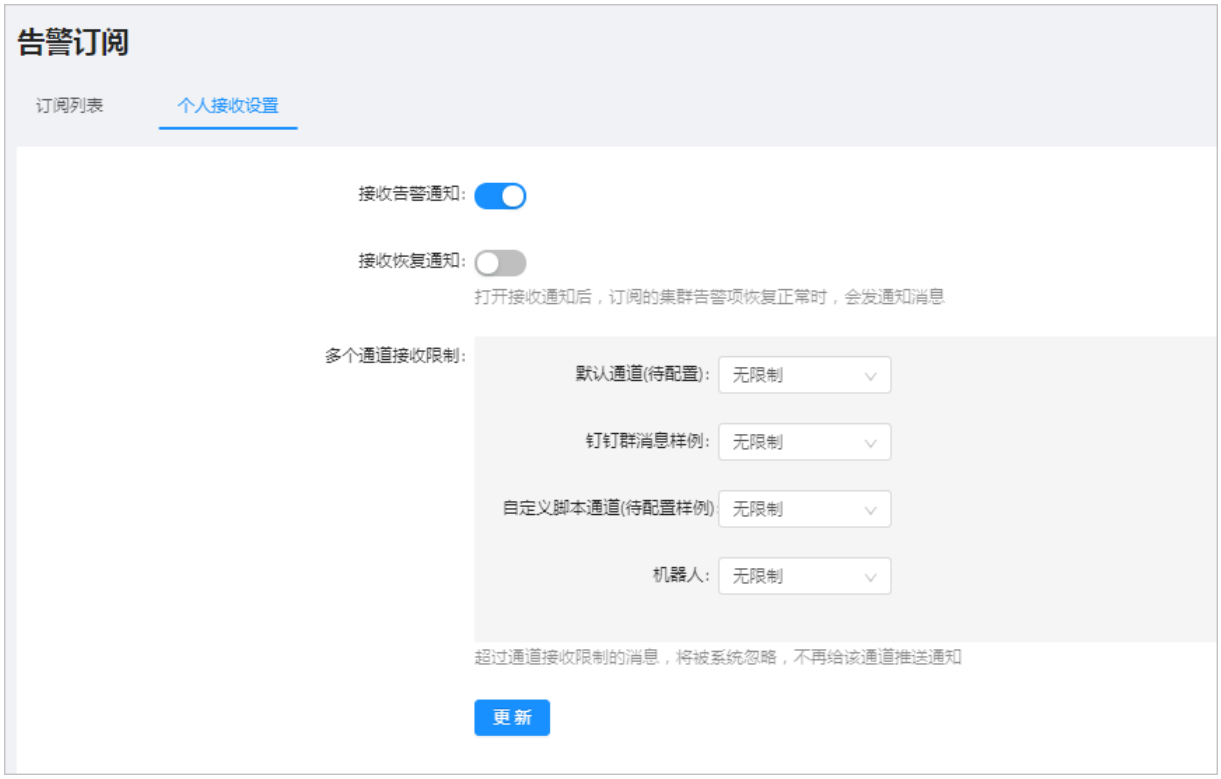


> **说明**
>
> 配置告警订阅时，用户只能选择有读权限的集群。如果配置告警订阅之后，用户对集群的读权限被撤销，那么告警订阅配置虽然仍然存在，但是用户仍然无法接收没有读权限集群的告警消息。

个人接收设置

用户可以配置个人接收设置。

告警有故障恢复提醒消息，故障恢复消息不会进行聚合，消息的数量可能比较多，建议关闭。个人接收设置项如下，请根据业务需求自定义配置。



3.4.6. 查看告警事件

本章节为您介绍查看告警事件详情。

告警事件列表

告警事件列表用于查看和检索全部告警。告警事件列表支持丰富的搜索条件，其中关键字搜索会匹配告警概述、告警详情和所有标签的值。例如对于OB告警，关键字可匹配OB集群组、OB集群、OB租户、主机IP等标签的值。



告警事件详情

在告警事件列表点击某条告警，进入告警事件详情页。支持：

- 查看告警事件的详细信息。



- 单击右上角**查看告警规则**按钮查看告警规则。


查看告警规则
规则信息
告警项所属范围：OB-服务器
匹配对象：
触发条件：
运算规则：ob_host_disk_readonly_flag == 1
持续时长：0 分钟
检测周期：60 秒
消除周期：5 分钟
基本信息
告警项名称：ob_host_disk_readonly
说明：OB服务器磁盘只读
告警等级：严重
告警概述模板：${alarm_target} ${alarm_name}
告警详情模板：${alarm_target} ${alarm_name}
修改信息
最近修改人：
最近修改时间：2020-05-16 13:33:40
所在告警项组：dba
关闭

- 单击右上角**屏蔽告警**按钮快速屏蔽告警。



- 查看告警对应的通知记录。

3.4.7. 查看告警通知记录

通知消息记录 页面可以查看通知发送记录和状态。支持查看90天内的通知记录，超过90天的通知记录会自动归档。

支持查看指定消息的消息内容、消息形式、通知渠道、来源、所属集群、告警时间和接收者。

其中 **接收人** 搜索条件默认为当前登录用户。



> **说明**
>
> 如果需要查看群消息通道发送的告警，请在**接收人**筛选条件中选择**全部**来搜索。



3.4.8. 屏蔽告警

本章节为您介绍屏蔽告警的操作。

告警屏蔽列表页展示全部告警屏蔽配置，默认显示登录用户创建的告警屏蔽配置。

- 告警屏蔽创建之后，可能不会立刻生效，最多需要30秒生效。
- 超过屏蔽截止时间7天以上的屏蔽配置，会自动归档。
- 执行 OB 集群的运维任务时，运维任务会自动设置告警屏蔽。



操作步骤

在告警事件和通知页面屏蔽告警可以快速创建告警屏蔽配置，只需要选择屏蔽结束时间即可。本文档为您展示在**屏蔽告警**页面屏蔽告警的操作步骤。

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 告警**。

3. 在屏蔽告警页签中单击**新建屏蔽条件**。
4. 根据业务需求选择 **OB** 页签或**应用**页签。此处以 **OB** 为例。
   i. 选择 OB 集群。集群为必选且只能选择一个。
   ii. 设置**屏蔽范围**。
   iii. 选择**屏蔽项**。
   iv. 设置屏蔽**结束时间**。



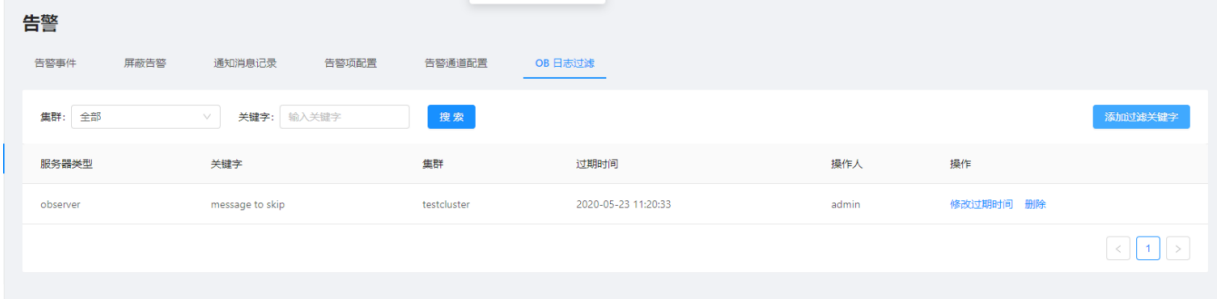
5. 单击确定。

3.4.9. OB 日志过滤

OB日志告警不基于告警规则触发，实现原理是在OB服务器上监视日志，发现ERROR级别的日志触发告警。如果发现有误报的情况，可以在日志过滤页面配置日志过滤规则。

查看关键字过滤规则

日志过滤列表展示已配置的日志过滤规则。



添加过滤关键字

在页面右上角单击添加过滤关键字，并指定集群、服务器类型、关键字和过滤时间即可。关键字支持正则表达式语法，匹配关键字的日志内容不会触发告警。



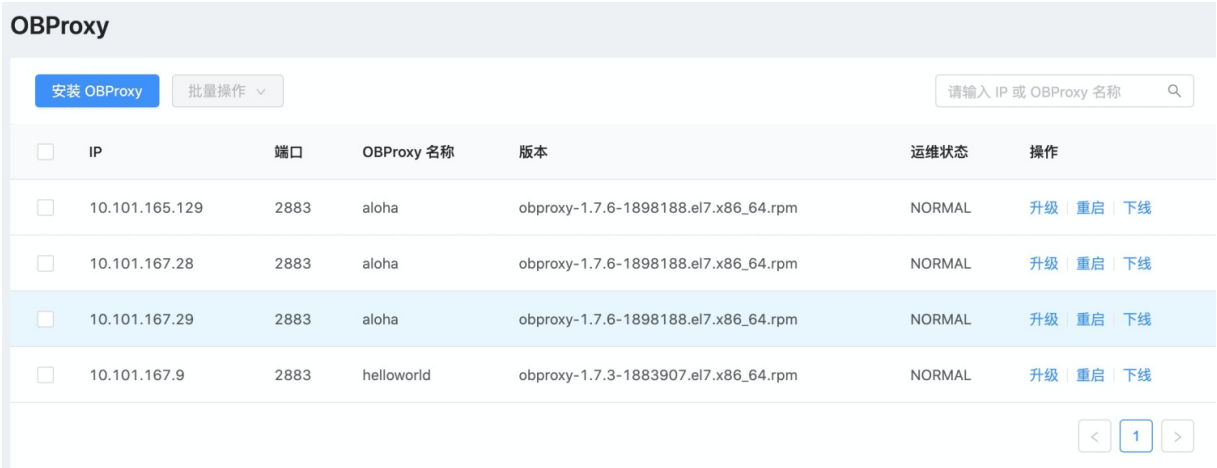
3.5. OBProxy 运维

3.5.1. 安装 OBProxy

在界面右上角单机返回旧版，即可进入 OBProxy 界面进行安装。

操作步骤

1. 单击导航栏的**运维**，进入 **OBProxy** 界面。
2. 在 OBProxy 页面查看 IP、端口、OBProxy 名称、版本、运维状态、操作等信息：



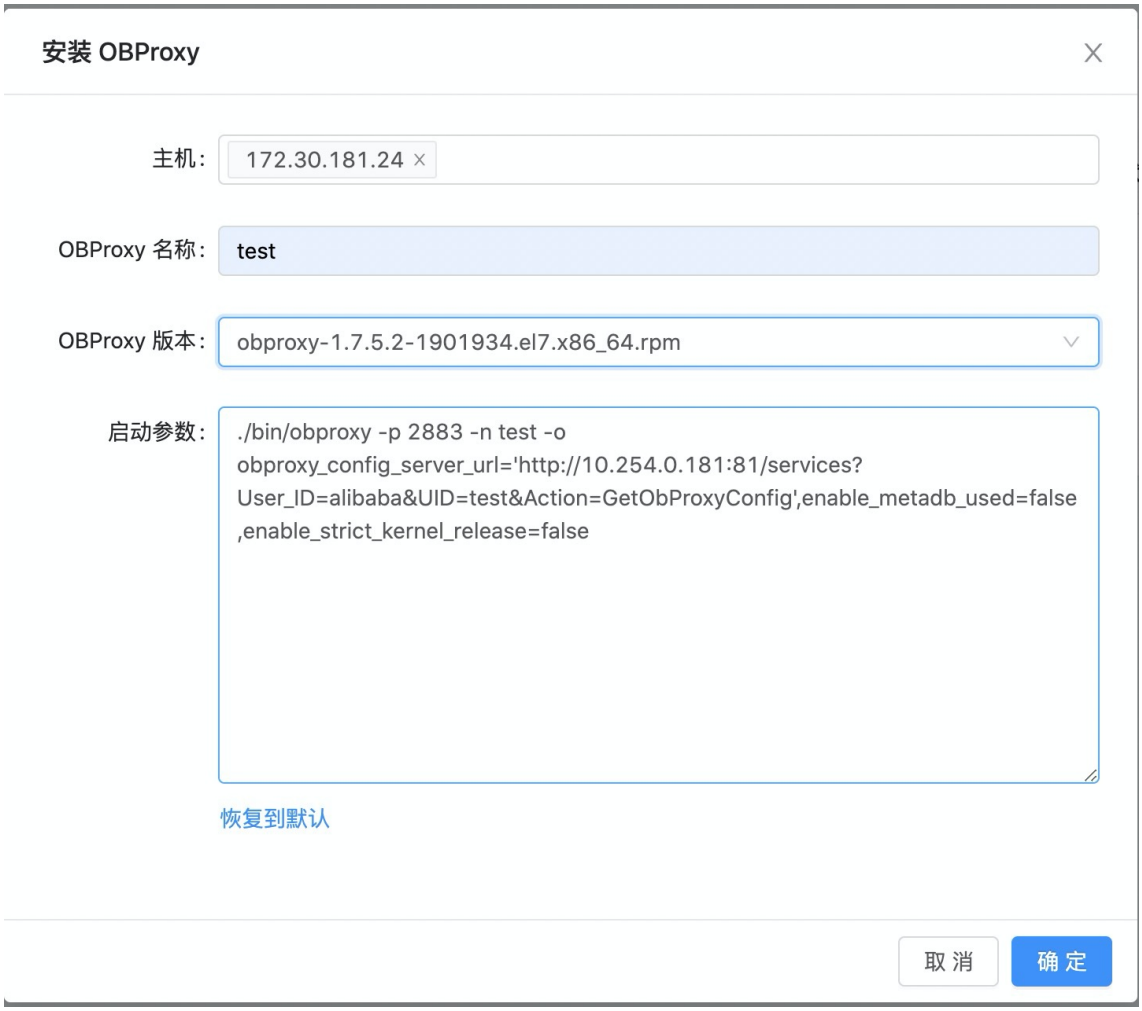
字段	说明

运维状态	• OPERATING 状态是正在执行运维操作； • NORMAL 状态时可以发起运维操作。
操作	• 单击操作列表下**升级**后详细操作参见：升级 OBProxy； • 单击操作列表下**重启**后详细操作参见：重启 OBProxy； • 单击操作列表下**下线**后详细操作参见：下线 OBProxy。

3. 单击**安装 OBProxy**。
4. 在弹出的对话框中，填写如下信息：
   - **主机**：可在下拉框中选择主机，或者单击**添加主机**添加新的机器。



   - **OBProxy 版本**：在下拉框中选择已有的 OBProxy 版本，或者单击**添加版本**在弹出框中**点击上传** 1.7.3 及以上版本的 OBProxy 文件。



- **启动参数**：参考启动参数说明，对启动参数进行设置，参数分为如下两种：
  a. 第一类是更新后可以直接生效；
  b. 第二类是更新后需要重启OBProxy才能生效。

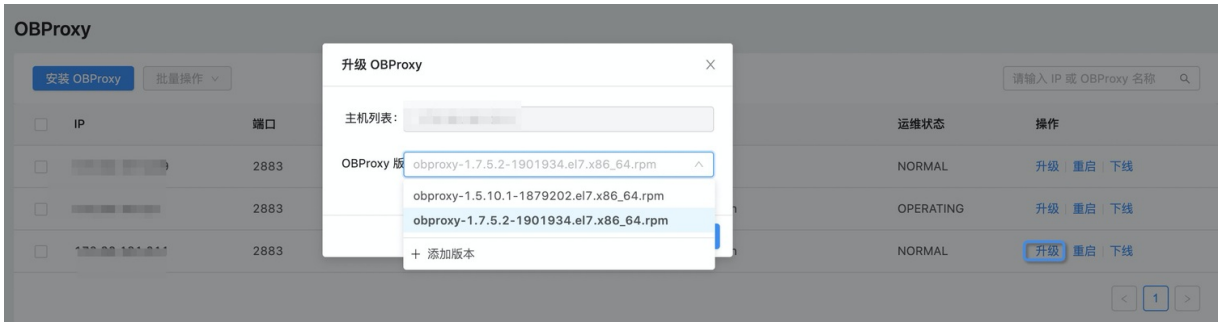
5. 单击**确定**，会生成一个 OBProxy 运维任务，可在**系统管理>任务**中，查看安装进展。

3.5.2. 升级 OBProxy

在界面右上角单机返回旧版，对目标 OBProxy 进行升级操作。

操作步骤

1. 单击导航栏的**运维**，进入 OBProxy。
2. 单击对应机器 IP 后的**升级**。
3. 在弹出框中选择要升级到的目标 OBProxy 版本，或单击**添加版本**添加 OBProxy 版本。



4. 单击**确定**，可在**系统管理>任务**中查看升级进展。
   - 当状态列下为**完成**时，升级成功；



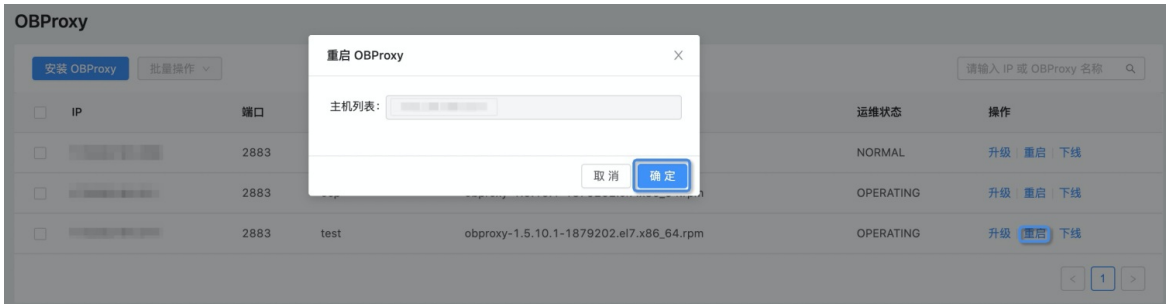
   - 如状态为**失败**，单击**任务名**，查看具体原因。

3.5.3. 重启 OBProxy

在界面右上角单机返回旧版，对正在运行中的 OBProxy 进行重启操作。

操作步骤

1. 单击导航栏的**运维**，进入 OBProxy。
2. 单击对应机器后的**重启**。



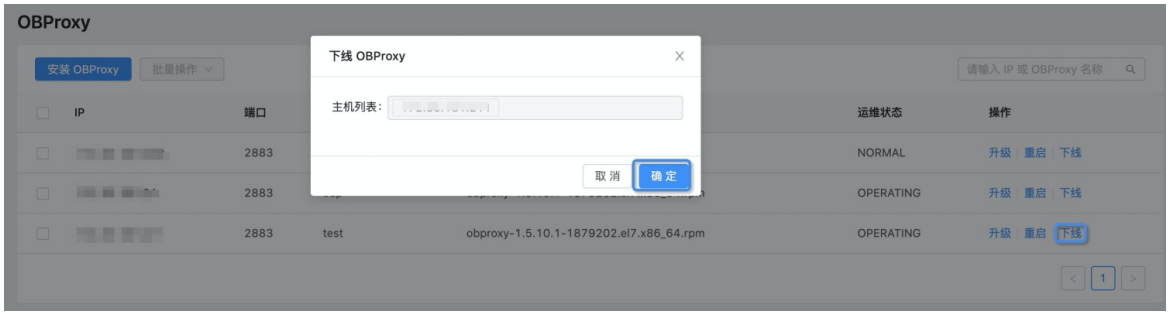
3. 单击**确定**，可在**系统管理>任务**中查看进展。

3.5.4. 下线 OBProxy

在界面右上角单击返回旧版，下线冗余的 OBProxy 。

操作步骤

1. 单击导航栏的**运维**，进入 OBProxy。
2. 在单击对应机器后的**下线**。



3. 单击**确定**，可在**系统管理>任务**中查看下线进展。
   - 当状态列下为**完成**时，升级成功；



   - 如状态为**失败**，单击**任务名**，查看具体原因。

3.5.5. 启动参数说明

本章节为您相信介绍启动参数说明。

安装 OBProxy 时会在弹出框中填写启动参数，启动参数分为两类：

- 一类更新后即可直接生效；
- 一类更新后需要重启 OBProxy 方可生效。

更新后直接生效的参数说明

name	info	default	range	need reboot
mysqlclient_query_timeout	proxy内部使用mysql客户端与observer建立连接时所设置的查询超时时间	120s	[1s]	false

name	info	default	range	need reboot
mysqlclient_net_wait_timeout	proxy内部使用mysql客户端与observer建立连接时设置的连接超时时间以及 read_timeout 、 write_time_out , -1表示永不超时	10s	[-1,]	false
mysqlclient_connection_refresh_interval	刷新proxy内部mysql连接池间隔,实际配置时间是value*50	200ms	[200ms,]	false
mysqlclient_connection_pool_warn_time	连接池告警时间,如果一条连接被占用时间过长就会产生一条告警日志	1s	[0s,]	false
mysqlclient_conn_limit_per_observer	每个连接池处理的sql连接数上限	10	[1,]	false
proxy_info_check_interval	proxy检测配置更新的间隔时间	20s	[1s,1h]	false
schema_refresh_interval	刷新schema时间间隔	20s	[1s,1h]	false
server_state_refresh_interval	刷新observer状态的时间间隔	20s	(0s,1h]	false
config_server_refresh_interval	刷新config server时间间隔	20s	[60s,1d]	false

name	info	default	range	need reboot
stat_sync_interval	同步统计全局信息task的时间间隔, 0表示不执行该task	60s	[0s,1d]	false
stat_dump_interval	dump统计全局统计信息task的时间间隔	6000s	[0s, 1d]	false
stat_table_sync_interval	更新内部表全局统计项task的时间间隔	60s	[0s,1d]	false
fetch_proxy_bin_random_time	proxy执行热升级时, 在fetch新的bin文件之前需要随机等待时间的最大值	300s	[1s,1h]	false
hot_upgrade_rollback_timeout	热升级之后, 默认执行rollback的等待时间	24h	[1s,30d]	false
hot_upgrade_failure_retries	热升级失败后重试次数	5	[1,20]	false
fetch_proxy_bin_timeout	热升级fetch binary的超时时间	120s	[1s,1200s]	false
hot_upgrade_graceful_exit_timeout	proxy 自动退出的等待时间	120s	[0s, 30d]	false
log_cleanup_interval	执行cleanup任务的时间间	3600s	[5s,30d]	false

name	info	default	range	need reboot
log_dir_size_threshold	proxy日志大小阈值，超过阈值即可进行日志清理	64GB	[256MB,1T]	false
conn_id_fetch_count	一次获取的conn_id数目	10240	[10,100000]	false
long_async_task_timeout	可能需要长时间访问observer的异步task的超时时间	60s	[1s,1h]	false
short_async_task_timeout	可能需要短时间访问observer的异步task的超时时间	5s	[1s,1h]	false
client_max_connections	proxy所能接受的客户端最大连接数	8192	[0,65535]	false
observer_query_timeout_delta	考虑到网络传输延时，给observer的ob_query_timeout设置的增量	20s	[1s,30s]	false
connect_observer_max_retries	proxy重连observer最大次数	3	[2,5]	false
net_config_poll_timeout	网络事件超时时间	10ms	[0,]	false
default_inactivity_timeout	tcp连接默认超时时间	180000s	[1s,30d]	false
sock_send_buffer_size_out	socket的发送缓冲区大小	0	[0,8MB]	false

name	info	default	range	need reboot
sock_recv_buffer_size_out	socket的接收缓冲区大小	0	[0,8MB]	false
sock_option_flag_out	socket flag参数	0	[0,1]	false
sock_packet_mark_out	socket mark参数	0	[0,1]	false
sock_packet_tos_out	socket tos参数	0	[0,1]	false
server_tcp_init_cwnd	新建server连接时设置tcp初始拥塞窗口值	0	[0,64]	false
proxy_mem_limited	proxy运行时内存上限, 超过上限proxy自动退出	800MB	[100MB,4GB]	false
enable_flow_control	判断是否在tunnel中开启流量控制	false		false
flow_high_water_mark	tunnel中流量控制的上限	0	[0,16MB]	false
flow_low_water_mark	tunnel中流量控制的下限	0	[0,16MB]	false
flow_consumer_reenable_threshold	tunnel流量控制配置数据转发次数阈值	0	[0,131072]	false

name	info	default	range	need reboot
flow_event_queue_threshold	tunnel流量控制配置工作线程队列中任务数阈值	0	[0,20]	false
tunnel_request_size_threshold	proxy接收的request size的阈值, 超过该阈值, proxy直接使用tunnel转发客户端请求, 不解析	84KB	[4KB, 16MB]	false
default_buffer_size	request和response的默认块大小	8KB	[1KB, 64KB]	false
default_buffer_water_mark	默认的buffer water mark	32KB	[4B, 64KB]	false
enable_trans_detail_stats	判断是否开启transaction状态统计	true		true
enable_mysqlsm_info		true		false
enable_report_session_stats	session级别统计项汇报到内部表的开关	true		false
slow_transaction_time_threshold	事务执行时间门限值, 超过此时间打印一条warn日志	5s	[0s,30d]	false
slow_proxy_process_time_threshold	OBProxy 预执行请求执行时间阈值, 超过该时间打印一条 warn 日志	2ms	[0s,30d]	false

name	info	default	range	need reboot
query_digest_time_threshold	请求执行时间阈值，超过该时间打印一条日志到obproxy_digest.log 日志里	100ms	[0s,30d]	false
slow_query_time_threshold	慢请求执行时间阈值，超过该时间打印一条日志到obproxy_slow.log 日志里	500ms	[0s,30d]	false
congestion_failure_threshold	proxy判断是否将server列入黑名单的依据，在congestion_fail_window周期内server出错次数超过该值即加入黑名单	5	[0,]	false
min_keep_congestion_interval	黑名单中的server在洗白之前需要等待的最短时间	20s	[1s,1d]	false
congestion_fail_window	server出错统计周期,与congestion_failure_threshold配合使用	120s	[1s,1h]	false
congestion_retry_interval	被加入活着不可用黑名单的server，多少时间后重试	20s	[1s,1h]	false
enable_congestion	黑名单开关	true		false

更新后需要重启生效的参数说明

name	info	default	range	need reboot
stack_size	线程栈大小, 创建线程时使用	1MB	[1MB,10MB]	true
work_thread_num	工作线程数, 初始化工作线程时使用	8	[1,128]	true
task_thread_num	task线程数, 初始化task线程时使用	2	[1,4]	true
dedup_queue_thread_num	执行fetch任务线程数	2	[1,4]	true
dedup_queue_bucket_num	dedup_queue 的 bucket_num 参数	256	[256,1024]	true
dedup_queue_task_queue_size	dedup_queue 的 task_queue_size 参数	1024	[512,1024]	true
kv_cache_bucket_num	初始化kvcache时使用	1024	[512,2048]	true
net_accept_threads	执行accept的线程数	2	[0,8]	true

frequent_accept	初始化net accept 参数	true		true

3.6. 备份与恢复

3.6.1. 概述

OCP 备份恢复功能支持针对 OceanBase 集群和租户级别进行全量备份、增量备份，支持日志备份、完全恢复和不完全恢复功能。

在 OCP 中备份并恢复数据的主要操作流程如下：

1. 新增备份与恢复配置。
2. 上传并安装备份恢复组件。
3. 创建集群备份调度。
4. 观察调度发起的备份任务的结果。
5. 发起恢复任务。
6. 连接恢复租户，查看恢复数据。

3.6.2. 进入备份恢复界面

备份恢复页面在旧版控制台中，您需要切换到旧版控制台才能使用该功能。您可以在备份恢复界面中进行备份恢复组件相关的运维管理，备份调度和任务查看等操作。

操作步骤

1. 登录 OCP 。
2. 在页面右上角单击**回到旧版**。



3. 在左侧导航栏中单击**运维 > 备份恢复**。



3.6.3. 准备工作

准备工作需要进行备份恢复元数据库的准备和存储介质的准备。

创建备份恢复元数据库

在基于 Docker 镜像安装 OCP时，会在MetaDB 租户下默认创建 backup1472、backup147x、backup21、backup2230 四个不同版本的元数据库，用户可以根据具体的 OB 版本选择使用。

准备存储介质

可以选择 OSS 和 NFS 其中一种方式作为备份恢复的存储介质。其中，NFS一般适用于独立对外场景，OSS一般适用于专有云场景。

OSS的配置

需要获取配置管理中需要填写的 OSS 相关配置参数：

- OSS账号
- 存储空间（Bucket）
- Endpoint（访问域名）
- AccessKey（访问密钥）、访问身份验证中用到的 AccessKeyId和 AcessKeySecret

NFS的配置

备份组件、恢复组件、OCP 和恢复的目标 OB 集群都需要能够访问备份配置的NFS目录，所以需要把NFS目录挂载到这些机器，挂载的本地目录必须与备份配置文件中的目录相同。

- NFS server的配置

在示例中，统一以 obbackup 目录名命名。

```
yum install -y nfs-utils portmap
service nfs start
echo '/obbackup *(rw,all_squash,anonuid=500,anongid=500)' >/etc/exports
chmod 777 /obbackup
service nfs restart
exportfs
```

- NFS client的配置

备份组件、恢复组件、OCP 和恢复的目标 OB 集群，都需要作为 NFS 的客户端进行挂载，否则会出现无法恢复的情况。

```
showmount -e 11.166.84.52 #该IP为示例ip
mkdir /obbackup
chmod 777 /obbackup
mount -o soft 11.166.84.52:/docker /obbackup
```

3.6.4. 管理备份恢复配置

3.6.4.1. 新增备份配置

本章节为您介绍新增备份配置的操作。

前提条件

- 已创建备份恢复元数据库。
- 已准备存储介质。

操作步骤

1. 登录 OCP。
2. 切换到旧版控制台，进入备份恢复界面。
3. 在**配置管理**页签左上角单击**新增配置**。
4. 设置**基本配置**。
   i. 设置**配置名称**。
   ii. 设置**配置类型**为**备份**。
   iii. 选择**存储类型**。

默认会自动填写 OCP 元数据库所在的数据库信息，因为其中也有一份备份恢复的元数据库。请根据实际所用的元数据库信息进行修改。

- 存储类型为 **OSS**：需要填写 OSS 文件路径、、access_key_id、secret_access_key、rs_list_url、meta_ip、meta_port、metadb_user_name、metadb_user_passwd 和 metadb_database_name。页面也自动填充部分参数，请确认内容是否正确。
- 存储类型为 **File**：填写挂载 NFS 文件相关信息即可。

配置项	说明
文件路径	OSS 文件路径，或挂载 NFS 的文件地址。
host	Endpoint。
access_key_id	OSS 账号的 AcceesKey ID 。
secret_access_key	OSS 账号的 AcceesKeySecret。
rs_list_url	默认填写当前 OCP 提供的 RS List URL。
meta_ip	备份恢复元数据库的 IP 地址。
meta_port	备份恢复元数据库的端口号。
metadb_user_name	备份恢复元数据库的用户名。
metadb_user_passwd	备份恢复元数据库的密码。
metadb_database_name	根据页面提示选择合适的元数据库。



5. （可选）在**高级配置**页签中设置**高级配置**。一般情况下，高级配置已自动填写相关信息，无需修改。
6. 单击**确定**。

3.6.4.2. 新增恢复配置

本章节为您介绍新增恢复配置的操作

前提条件

- 已创建备份恢复元数据库。
- 已准备存储介质。

操作步骤

1. 登录 OCP。
2. 切换到旧版控制台，进入备份恢复界面。
3. 在**配置管理**页签左上角单击 **新增配置**。

4. 设置**基本配置**。
   i. 设置**配置名称**。
   ii. 设置 **配置类型**为**恢复**。
   iii. 填写相关配置。

默认会自动填写 OCP 元数据库所在的数据库信息，因为其中也有一份备份恢复的元数据库。请根据实际所用的元数据库信息进行修改。

配置项	说明
ocpMetaDb.clusterName	备份恢复元数据库所在的集群名，改集群必须在OCP管控下。
ocpMetaDb.tenantName	备份恢复元数据库所属的租户。
ocpMetaDb.userName	备份恢复元数据库所在的用户。
ocpMetaDb.dbName	根据提示信息选择合适的元数据库。
ocpMetaDb.encryptedPassword	备份恢复元数据库的密码。
ocpMetaDb.configUrl	备份恢复元数据库所在集群的 configUrl。填写集群时，此处自动填充，无需修改。



5. （可选）在**高级配置**页签中设置**高级配置**。一般情况下，高级配置已自动填写相关信息，无需修改。
6. 单击**确定**。

3.6.4.3. 管理配置

备份和恢复配置文件中的相关参数和配置项必须完全正确，否则备份组件和恢复组件无法正常工作，导致备份或恢复失败。

根据业务需求，您可以在配置管理界面，对配置数据进行基本的管理操作，包括：

- **查看配置详情**
- **编辑配置信息**
- **删除备份或恢复配置**



3.6.5. 安装备份恢复组件

3.6.5.1. 安装备份恢复组件

本章节为您介绍安装备份恢复组建的操作。

前提条件

安装备份或恢复组件需要有主机资源，请确认当前具备空闲主机资源。如果没有主机资源，请参考**主机管理**部分文档，在 OCP 新版控制台的**主机管理**页面添加主机。

操作步骤

1. 登录 OCP。
2. 切换到旧版控制台，进入备份恢复界面。
3. 在**组件管理**页签中单击**安装组件**。
4. 选择**组件类型**，并指定主机。
5. 选择**部署版本**。如果没有合适的版本，可以单击添加版本上传新的安装包。
6. 指定 平台会自动根据您所选的部署版本筛选出于该版本匹配的备份恢复配置。如果此处没有配置文件，请先新增备份配置。
7. 单击 确定



8. 提交成功后，页面将弹出对话框，您可以点击链接查看安装进度。

安装进度页面在新版控制台中，查看进度后需要切回到旧版控制台继续组件的运维。

安装恢复组件的流程与备份组件一致，同时支持同一台主机安装备份和恢复两个组件，但版本是以先安装的角色的版本号为准，不可单独指定不同版本。

3.6.5.2. 运维备份恢复组件

已安装的备份恢复组件会显示在备份恢复页面的组件管理页签中。您可以对这些已安装的组件进行一些常见的运维操作。

例如：

- 查看任务历史。在**任务历史**页签中单击**查看**可以查看该组件的历史运维情况。
- 重启
- 下线
- 升级
- 更新配置

  当组件的配置文件需要更新时，可以先在**配置管理**页签中修改配置，然后再更新配置。系统会自动推送最新配置到组件端。



3.6.6. 执行备份

3.6.6.1. 集群备份调度

创建备份配置和恢复配置，并安装备份组件之后，就可以随时进行数据备份了。

当前支持以下备份粒度：

- 对当前OCP指定集群进行备份
- 对指定集群的指定租户进行备份

当前支持以下两种备份模式：

- 周期性的自动备份调度：通过预设的备份配置规则，系统自动进行周期性的备份。
- 手动执行备份：随时手动执行一次数据备份。

创建集群备份调度

前提条件

- 已创建对应版本的备份配置。
- 已安装备份组件。

操作步骤

1. 登录 OCP。
2. 在页面右上角单击**返回旧版**。
3. 在左侧导航栏中单击**集群**。
4. 在指定集群对应的**操作**列中单击**集群备份**。
5. 填写相关配置。
   i. 设置备份模式和备份周期。
   - 备份模式可设置为按周备份或按月备份。
   - 备份周期根据备份模式的不同，可设置为指定周几或几号进行备份。

   ii. 设置备份配置。
   - 下拉列表中仅显示版本匹配的备份配置。
   - 如果没有合适的备份配置，请创建备份配置后再设置集群备份调度。

iii. 设置备份开始时间。

iv. 确认是否同时发起增量备份。增量备份即日志备份。



6. 单击**提交**。

管理集群备份调度

创建集群备份调度之后，系统会根据预设的周期和时间，自动发起备份任务。您也可以根据业务需要，修改备份调度任务。当前支持以下管理调度的操作：

- 暂停备份调度任务

在指定备份调度任务对应的操作列单击操作>暂停，即可暂停该任务。

- 重新启动备份调度任务

在指定备份调度任务对应的操作列单击操作>重新，即可重新启动该任务，系统根据预设的时间和周期继续定期发起任务。

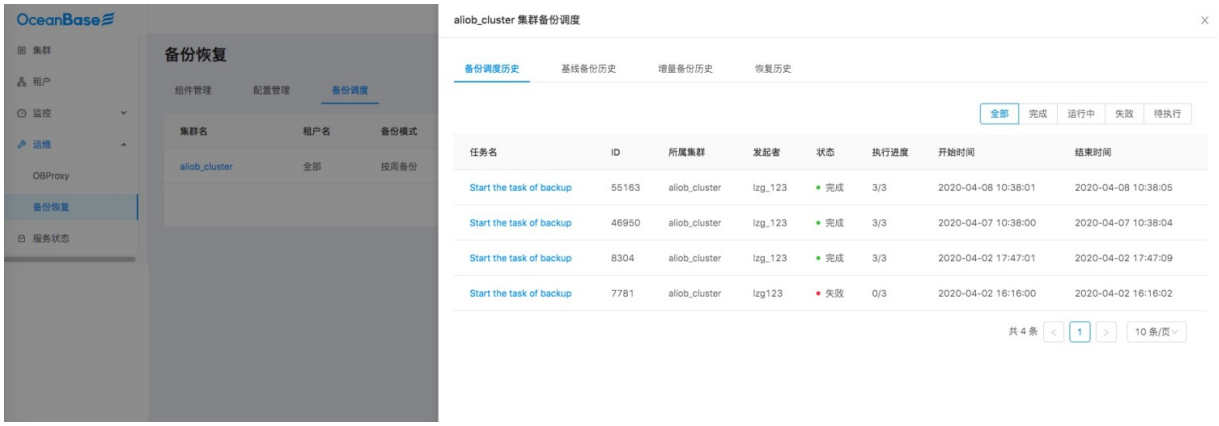


查看备份历史

在备份恢复页面的备份调度 页签中，可以查看所有的调度任务的基本信息。包括集群名、租户名、备份模式、备份周期、当前调度任务状态、基线备份任务状态、增量备份任务、发起人。

单击集群名，可以查看指定集群的备份调度历史、基线备份历史、增量备份历史和恢复历史。其中，备份调度历史页签中还可以根据备份调度任务的状态进行筛选。

- 备份调度历史：主要显示任务名、ID、所属集群、发起者、状态、执行进度、开始时间和结束时间。



- 基线备份历史：**基线备份历史**页签主要显示任务 ID、备份类型、任务类型、任务数量、集群、租户、开始时间、结束时间、数据版本、备份 URL、备份状态、备份大小。您可以指定备份任务来发起一次数据恢复。



- 增量备份历史：**增量备份历史**页签主要显示集群名、租户白名单、majorVersion、备份URL、agentServerIp、位点、延时（秒）、状态、停止状态和错误信息。



3.6.6.2. 租户备份调度

租户备份和该租户所在的集群备份是互斥关系，不能单独同时存在。

一个集群中只能创建一个租户备份调度。

- 没有备份调度时，可以创建集群备份调度或租户备份调度。
- 如果已创建集群备份调度，该集群中所有租户的备份调度默认继承集群的配置，不能另外单独设置租户调度任务。
- 若已创建租户备份调度，不能再次创建集群维度的备份调度。需要先取消租户的备份调度，并暂停增量备份；才能重新配置集群备份调度。

前提条件

该租户所属的集群中没有进行中的备份调度任务或增量备份。

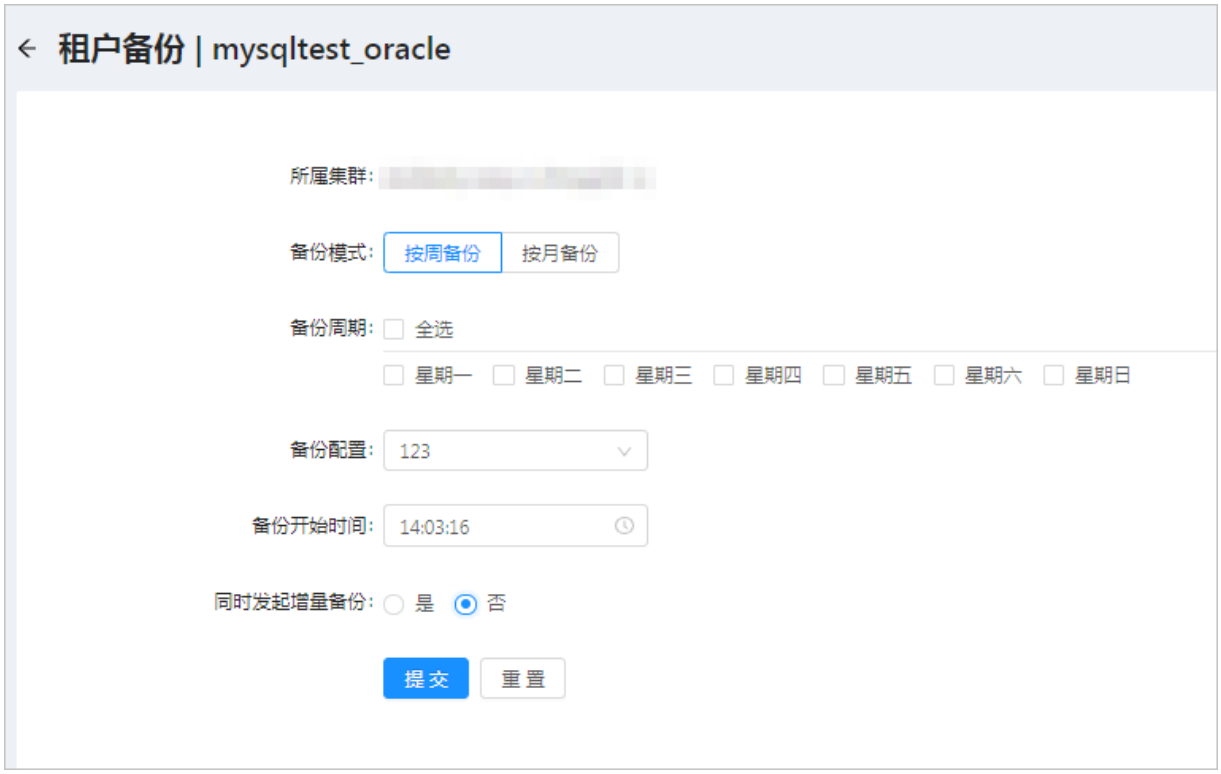
操作步骤

1. 登录 OCP。
2. 在页面右上角单击**返回旧版**。
3. 在左侧导航栏中单击**租户**。
4. 在指定租户对应的**操作**列中单击**租户备份**。
5. 填写相关配置。
   i. 设置**备份模式**和**备份周期**。
   - **备份模式**可设置为**按周备份**或**按月备份**。
   - 备份周期根据备份模式的不同，可设置为指定周几或几号进行备份。

   ii. 设置**备份配置**。
   - 下拉列表中仅显示版本匹配的备份配置。
   - 如果没有合适的备份配置，请创建备份配置后再设置集群备份调度。

   iii. 设置**备份开始时间**。

   iv. 确认是否**同时发起增量备份**。增量备份即日志备份。



6. 单击**提交**。

租户备份调度管理

创建租户备份调度之后，系统会根据预设的周期和时间，自动发起备份任务。您也可以根据业务需要，修改备份调度任务。当前支持以下管理调度的操作：

- 暂停备份调度任务：在指定备份调度任务对应的操作列单击**操作 > 暂停**，即可暂停该任务。
- 重新启动备份调度任务：在指定备份调度任务对应的操作列单击**操作 > 重新**，即可重新启动该任务，系统根据预设的时间和周期继续定期发起任务。

3.6.6.3. 立即备份

立即备份指对已设置备份调度的集群或租户，手动发起一个即时的备份任务。如果已配置了备份调度，可以直接手动进行一次基线备份。

> **说明**
>
> 一个 OceanBase 合并版本只能做一次基线备份，如果基于同一版本，发起多次备份，则后续的备份会直接报错，但这个版本的备份是正常可用的。

前提条件

该集群或租户已设置备份调度任务。

操作步骤

1. 登录 OCP。
2. 在页面右上角单击**返回旧版**。

3. 在左侧导航栏中单击**租户**或**集群**。
4. 在指定租户或集群对应的**操作**列中单击立即**备份**。



5. 在弹出对话框中单击**确定**。



3.6.7. 恢复备份数据

系统执行完一次备份之后，就可以随时根据这次备份发起一次恢复任务。目前恢复任务的最小粒度为租户级别数据，可以批量恢复多个租户。

前提条件

需要恢复的集群或租户已成功执行过一次基线备份。

操作步骤

1. 登录 OCP。
2. 进入备份恢复界面。
3. 在**备份调度**页签中单击指定集群名称。
4. 在**基线备份历史**页签中找到需要执行恢复的基线备份。
5. 在其对应的**操作**列单击**发起恢复**。此处的点击位置与恢复版本无关，主要取决于恢复时间，系统会自动计算使用哪个基线版本进行恢复。
6. 设置**恢复集群**和**恢复到的时间点**。目标集群可以是源集群，也可以是不同集群。
7. 填写用户密码。
8. 选择需要恢复的租户，并确认恢复**租户名**、**恢复 locality**、**恢复 pool list**、**恢复 priority zone**。系统会根据备份信息系统填充内容。请确认自动填充的内容是否正确。
   - 如果恢复到源集群，需要将恢复租户名修改为新名称。
   - 恢复 pool list指该租户相关的资源配置，资源池名需要在目标集群中不存在。



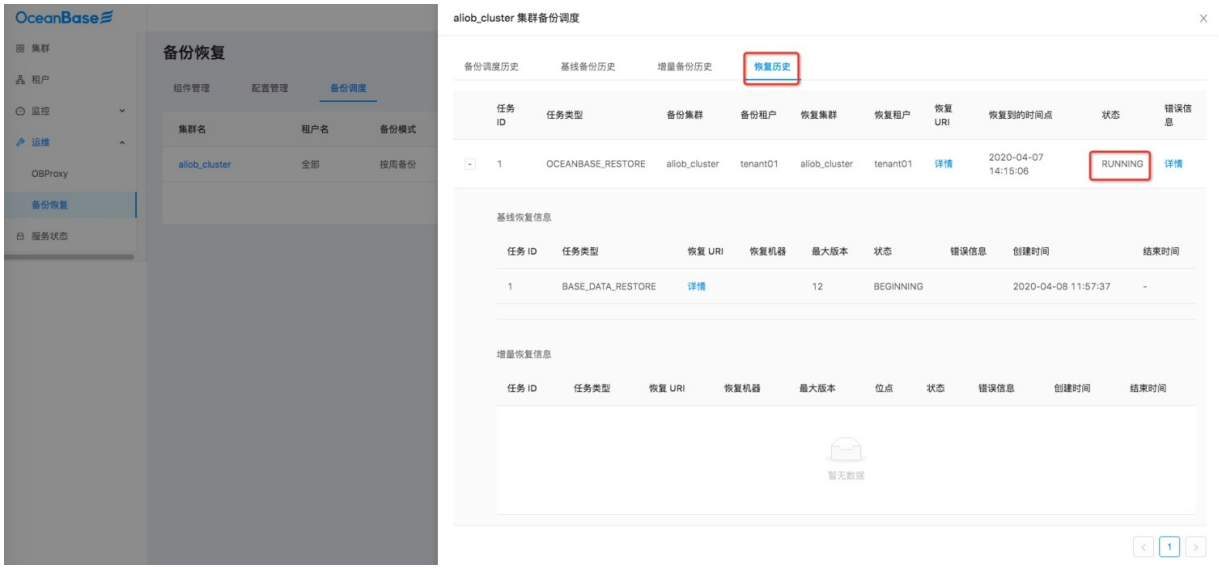
9. 单击 **确定**。发起恢复请求成功后，将会有成功提示。

查看恢复任务

在备份历史最右侧的 **恢复历史** 中，可以看到恢复任务的情况。该页面更新需要手动刷新查看。

恢复过程包含以下两个阶段，当两个阶段都成功时，才可认定为租户恢复成功。

- 基线恢复
- 增量恢复



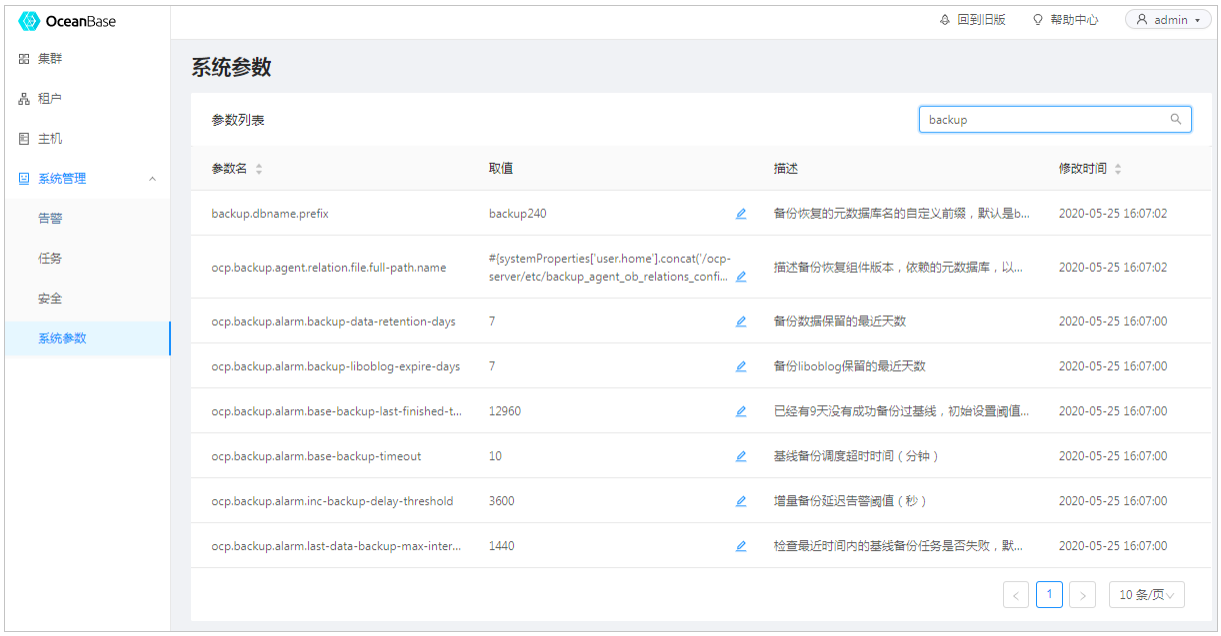
使用恢复租户

恢复租户的连接密码为空，即当弹出密码输入提示时，可以直接回车确认。

3.6.8. 监控和告警配置

备份恢复相关的配置在OCP中以系统参数的形式进行配置。

在新版控制台的系统管理>系统参数页面，搜索 backup 即可查看备份恢复涉及到的参数配置。



监控配置

当前 OCP 版本中，以下备份恢复相关的设置支持通过系统参数进行管理和配置：

- 组件的状态监控
- 基线备份任务失败的监控
- 增量备份任务延迟的监控，参见 `ocp.backup.alarm.inc-backup-delay-threshold`
- 备份调度任务失败的监控
- 备份文件的过期清理，参见 `ocp.backup.alarm.backup-data-retention-days`
- 备份组件的日志清理，参见 `ocp.backup.alarm.backup-liboblog-expire-days`

> **说明**
>
> 当前版本暂时不支持对备份文件容量进行监控。

告警配置

告警参数配置：

配置项	默认值	描述
ocp.backup.alarm.base-backup-last-finished-threshold	12960	已经有9天没有成功备份过基线，初始设置阈值为9天（12960分钟），单位为分钟

配置项	默认值	描述
ocp.backup.alarm.base-backup-timeout	10	基线备份调度超时时间（分钟）
ocp.backup.alarm.inc-backup-delay-threshold	3600	增量备份延迟告警阈值（秒）
ocp.backup.alarm.last-data-backup-max-interval-minutes	1440	检查最近时间内的基线备份任务是否失败，默认1天，单位分钟

告警配置请参照告警相关文档进行订阅和配置。

3.7. 用户与权限管理

3.7.1. 访问控制概述

在 OCP 平台中，用户是 OCP 平台的使用者、功能系统的操作者。角色是一组权限的集合，是权限的载体。每个用户都必须扮演一个或多个角色，具备这些角色所包含的权限，才能在 OCP 平台中执行对应的操作。用户的权限是其扮演的所有角色权限的并集。

OCP 平台中的初始账号 admin 是一个具备系统管理员角色的用户，拥有 OCP 平台所有集群和功能模块的管理权限。系统管理员可以创建其他用户，并为其分配一个或多个角色，使其在业务架构中扮演对应的职能，例如创建一个具备用户管理权限 USER_MANAGER 的用户，使其可以代替管理员创建其他用户。

3.7.2. 用户管理

3.7.2.1. 新建用户

出于信息安全和传统安全等方面的考虑，当创建用户时，建议授于用户所需的最小权限。OCP 的最小权限集合是 PROFILE角色，具备该角色的用户可以访问个人中心、修改个人信息、修改密码箱等。

OCP 初始预置了常见的角色，称为默认角色。默认角色包括具备全部管理权限的系统管理员角色 ADMIN、具备所有OceanBase 租户管理权限的 TENANT_MANAGER 角色等，基本覆盖常见对用户的权限管理的要求。如果需要通过用户对集群和租户提供更细粒度的权限，可以在创建用户时新建自定义角色并指定具体的集群ID和租户ID。

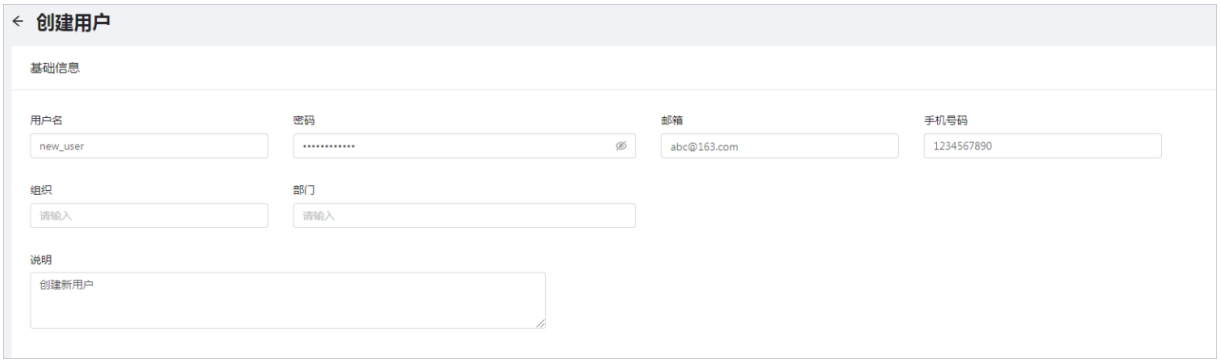
前提条件

当前登录用户拥有系统管理员（ADMIN）或用户管理（USER_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理> 安全**。
3. 在页面右上角单击**创建用户**。
4. 设置新用户的基础信息。

- 为新用户设置**用户名**、**密码**、**邮箱**、**手机号码**、**组织和部门**，并填写角色说明。
- 其中，用户名、密码和邮箱不能为空。
- 用户名应为小写英文字母开头，支持小写英文、数字、下划线、中横线和'.'的组合，长度为 4 ~ 48 位。用户名创建后不可修改。
- 密码长度8~32位，至少包含2位数字、2位大写字母、2位小写字母和2位特殊字符（即：._+@#$%）。



5. 设置新用户的角色信息。
   - 必须为用户指定一个或多个角色。
   - 如果 OCP 中没有合适的角色，可以在区域右上角单击 **新建角色**，创建一个新的角色，并将其分配给新用户。



6. 在页面右下角单击**提交**。

3.7.2.2. 查看用户列表

系统管理员或用户管理员可以查看当前 OCP 中所有的用户信息，包括邮箱、手机号、部门、组织、说明和其所拥有的的角色。

前提条件

当前登录用户拥有系统管理员（ADMIN）或用户管理（USER_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。

2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 在搜索框中通过用户名或用户名关键字搜索指定用户。
4. 在搜索结果中单击用户名，进入**用户详情**页面。您可以在**用户详情**页面查看用户的邮箱、手机号、部门、组织、说明和其所拥有的的角色。



3.7.2.3. 查看用户详情

系统管理员或用户管理员可以查看当前 OCP 中所有的用户信息，包括邮箱、手机号、部门、组织、说明和其所拥有的的角色。

前提条件

当前登录用户拥有系统管理员（ADMIN）或用户管理（USER_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 在搜索框中通过用户名或用户名关键字搜索指定用户。
4. 在搜索结果中单击用户名，进入**用户详情**页面。您可以在**用户详情** 页面查看用户的邮箱、手机号、部门、组织、说明和其所拥有的的角色。



3.7.2.4. 复制用户

需要新建用户时，如果新用户的基本信息和已存在的用户相同，也可以通过复制用户的方式操作。复制用户可以不必反复填写相同的基本信息，适用于需要批量创建用户的场景。

前提条件

当前登录用户拥有系统管理员（ADMIN）或用户管理（USER_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 找到需要复制的用户，并在其对应的**操作**列单击**复制**。
4. 为新用户设置基础信息和角色信息。除用户名和密码以外，新用户的基础信息和角色与被复制的用户相同，您可以在此基础上填写。



5. 在页面右下角单击**提交**。

3.7.2.5. 编辑用户

创建用户后可以修改用户的密码、邮箱、手机号等基本信息和权限信息。修改用户的角色后，用户下次登录时生效。

前提条件

当前登录用户拥有系统管理员（ADMIN）或用户管理（USER_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 找到指定用户，并在其对应的**操作**列单击**编辑**。
4. 重新输入用户的基础信息，也可以为其重新指定角色。



5. 在页面右下角单击**提交**。

3.7.2.6. 删除用户

用户信息涉及用户过往操作历史记录和跟踪，因此在 OCP 中，删除用户后，用户的账户处于被禁用状态，账户信息并不会消失（如用户名、Email 等信息）。 此时，如果创建同用户名或同 email 的用户，页面会报错“相同用户名或Email的用户已经存在”。请可以根据提示，选择其他的用户名和邮箱后重试。

前提条件

当前登录用户拥有系统管理员（ADMIN）或用户管理（USER_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 找到指定用户，并在其对应的**操作**列展开隐藏列表，单击**删除**。



4. 在弹出对话框中单击**删除**。

3.7.3. 角色管理

3.7.3.1. 概述

OCP 中的角色是一组权限的集合，角色需要被分配给用户才能实际生效。OCP 角色包括默认角色和自定义角色。

OCP 角色包括默认角色和自定义角色。

- 默认角色是系统自带的预置角色，基本覆盖常见对用户的权限管理的要求。例如具备全部管理权限的系统管理员角色 ADMIN、具备所有 OceanBase 租户管理权限的 TENANT_MANAGER 角色等。
- 自定义角色是用户在 OCP 中自定义创建的角色，相对于初始默认角色来说，可以具备集群和租户级别更细粒度管理权限，例如指定用户仅能查看指定集群、指定集群中的指定租户。但自定义角色只能被赋予集群、租户、备份恢复的相关管理权限、创建集群的系统权限。

默认角色

默认角色不可编辑或删除。OCP 默认角色及对应权限如下。

角色名称	说明	权限列表
TENANT_VIEWER	租户只读角色，拥有对 OCP 管理的所有OceanBase 租户及关联资源（集群、主机、后台任务、告警）的只读权限。	CLUSTER:*:TENANT:*:READ, CLUSTER:*:READ, HOST:*:READ, TASK:*:READ, ALARM:*:READ
TENANT_MANAGER	租户管理员角色，拥有对 OCP 管理的所有OceanBase 租户的管理权限，及关联资源（集群、主机、后台任务、告警）的只读权限。	CLUSTER:*:TENANT:*:*, CLUSTER:*:READ, HOST:*:READ, TASK:*:READ, ALARM:*:READ
BACKUP_MANAGER	集群备份恢复管理角色，拥有对 OCP 管理的所有 OceanBase 集群和租户的备份恢复管理权限，及主机、告警的只读权限，任务的管理权限。如果需要增加主机，需要给用户额外赋予 HOST_MANAGER角色。	CLUSTER:*:BACKUP:*:*, CLUSTER:*:READ, CLUSTER:*:TENANT:*:READ, HOST:*:READ, ALARM:*:READ, TASK:*:*
CLUSTER_VIEWER	集群只读角色，拥有对 OCP 管理的所有OceanBase 集群及关联资源（主机、后台任务、告警）的只读权限。	CLUSTER:*:READ, HOST:*:READ, TASK:*:READ, ALARM:*:READ

角色名称	说明	权限列表
CLUSTER_MANAGER	集群管理员角色，拥有对 OCP 管理的所有OceanBase 集群及关联资源（主机、后台任务、告警）的管理权限，拥有对间接关联资源（用户）的只读权限。	CLUSTER:*:*, HOST:*:*, TASK:*:*, ALARM:*:*, USER:*:READ
ALARM_MANAGER	OCP 告警管理角色，拥有告警和订阅管理权限及关联资源（集群、租户、主机、用户）的只读权限。	ALARM:*:*, CLUSTER:*:READ, HOST:*:READ, USER:*:READ
TASK_MANAGER	OCP 后台任务管理角色。	TASK:*:*
HOST_MANAGER	OCP 主机管理角色，具备主机层面的所有管理权限。	HOST:*:*
PROPERTY_MANAGER	OCP 系统配置参数管理角色。	PROPERTY:*:*
ROLE_MANAGER	OCP 角色管理角色，具备角色相关的所有管理权限。	ROLE:*:*
USER_MANAGER	OCP 用户管理角色，具备用户相关的所有管理权限。	USER:*:*
PROFILE	OCP 用户个人信息角色，用来登陆和访问个人中心。	PROFILE:*:*
ADMIN	OCP 系统管理员角色，拥有OCP的最大权限。	**

自定义角色

自定义角色只能被赋予集群、租户、备份恢复的相关管理权限、创建集群的系统权限，和默认角色配合使用时，可以提供多样化的授权策略。

目前，在自定义角色中可被授予的权限如下。

权限类别	权限内容
集群权限	指定集群的**创建租户**、**只读**、**更新**、**删除**或**以上全部**权限。

权限类别	权限内容
租户权限	**指定OB 集群**中**指定 OB租户**的**只读**、**更新**、**删除**或**以上全部**权限。
备份恢复权限	指定集群的以下权限： • **创建**：安装组件、发起备份和恢复 • **只读**：查看备份和配置情况 • **更新**：更新配置、组件运维 • **删除**：删除配置
系统权限	**创建 OB 集群**的权限。

3.7.3.2. 新建角色

如果 OCP 预置的默认角色无法满足实际需求，系统管理员或角色管理员可以创建新的自定义角色，为自定义角色赋予更细粒度的管理权限。和默认角色配合使用时，可以提供多样化的授权策略。

> **说明**
>
> 自定义角色只能被赋予集群、租户、备份恢复的相关管理权限、创建集群的系统权限。

前提条件

当前登录用户拥有系统管理员（ADMIN）或角色管理（ROLE_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 在**角色管理**页签右上角单击**新建角色**。
4. 输入**角色名称**和**说明**。
5. 根据业务需要，为角色赋予集群、租户、备份恢复的相关管理权限、创建集群的系统权限。
   - 创建、编辑等管理权限和只读权限之间没有交集，如果需要为角色赋予管理权限，需要同时添加只读权限。
   - 如果需要租户相关的只读或管理权限，必须同时赋予角色此租户对应集群的只读或管理权限。



6. 在页面右下角单击**提交**。

3.7.3.3. 查看角色信息

系统管理员、角色管理员可以查看当前 OCP 中指定角色或所有的角色信息，包括角色名称、说明、权限内容。

前提条件

当前登录用户拥有系统管理员（ADMIN）或角色管理（ROLE_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 在**角色管理**页签的搜索框中输入指定角色名称或关键字，并单击搜索图标。列表中可以查看指定角色的**说明**、**角色类型**、**权限**内容和**已分配用户**。



4. 在搜索结果中单击角色名称。角色详情中会显示该角色的所有信息。



3.7.3.4. 复制角色

如果需要在已有角色的基础上增加其他细粒度的权限，可以通过复制角色的方式创建新的角色。

即使复制默认角色，复制出来的新角色仍旧是自定义角色，仅能被赋予集群、租户、备份恢复的相关管理权限、创建集群的系统权限。不具备默认角色的系统管理相关功能。

前提条件

当前登录用户拥有系统管理员（ADMIN）或角色管理（ROLE_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 在**角色管理**页签中，找到需要复制的角色，在其对应的**操作**列单击**复制**。
4. 编辑新角色的基础信息和权限信息。



5. 在页面右下角单击**提交**。

3.7.3.5. 编辑角色

系统管理员、角色管理员创建自定义角色以后，可以随时编辑自定义角色的信息和权限。

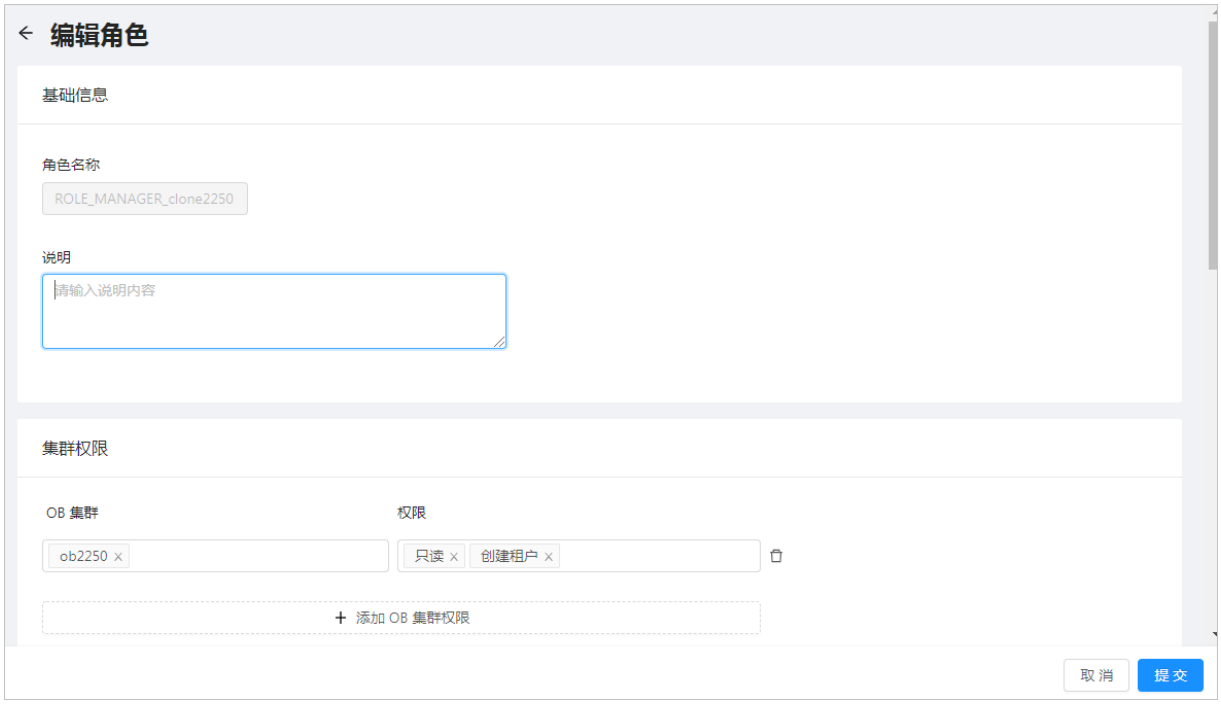
> **说明**
>
> - 修改已分配用户的角色后，该用户下次登录时生效。
> - 不支持修改默认角色。

前提条件

当前登录用户拥有系统管理员（ADMIN）或角色管理（ROLE_MANAGER）角色，具备对应的权限。

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 在**角色管理**页签中找到指定角色，并在其对应的**操作**列单击**编辑**。
4. 根据业务需求修改角色的说明或权限。



5. 在页面右下角单击**提交**。

3.7.3.6. 删除角色

根据业务需求，系统管理员、角色管理员可以随时删除自定义角色。不支持删除默认角色。

前提条件

当前登录用户拥有系统管理员（ADMIN）或角色管理（ROLE_MANAGER）角色，具备对应的权限。

> *注意：*

操作步骤

1. 登录 OCP。
2. 在左侧导航栏中单击**系统管理 > 安全**。
3. 在**角色管理**页签中找到指定角色，并在其对应的**操作**列单击**删除**。



4. 在弹出对话框中单击**删除**。

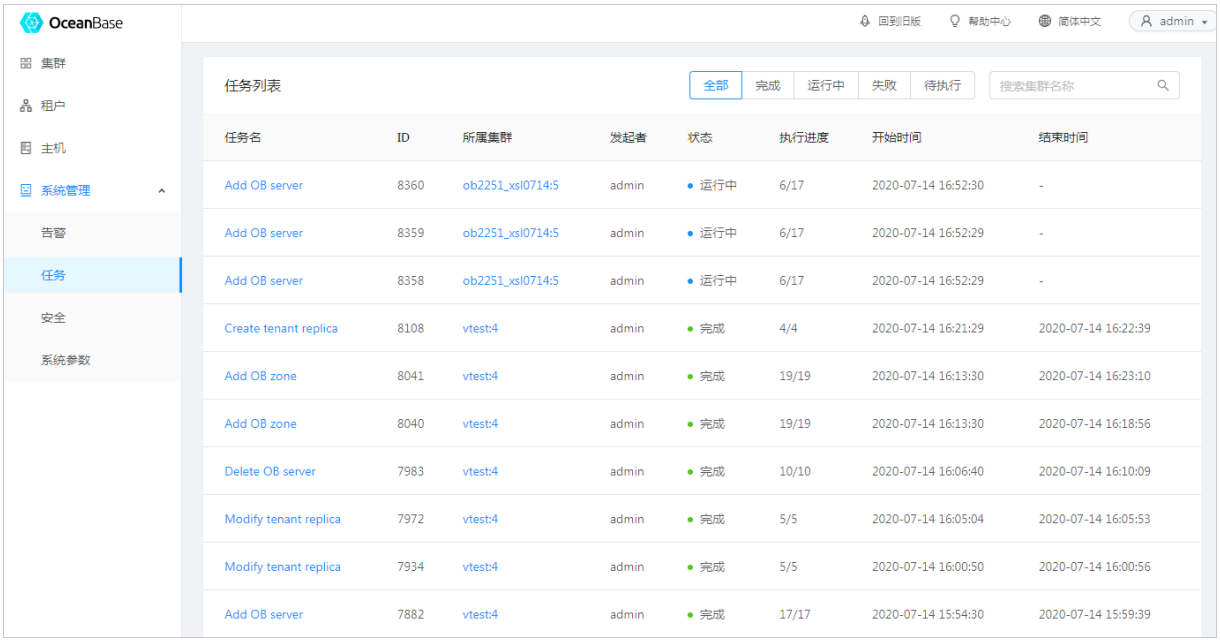


3.8. 任务管理

3.8.1. 查看任务列表

本章节为您介绍任务列表可查看的信息。

任务列表页面默认显示当前登录用户有权限查看的集群对应的所有任务。列表中展示任务的**任务名**、**ID**、**所属集群**、**发起者**、**状态**、**执行进度**、**开始时间**和**结束时间**。其中**所属集群**通过“集群名称:集群ID”的格式展示。在**任务列表**页面可以根据任务状态筛选任务，任务状态包括**完成**，**运行中**，**失败**，**待执行**，默认展示所有状态的任务。还可以通过集群名搜索指定集群的所有相关任务。



3.8.2. 查看任务详情

在左侧导航栏中单击系统管理 > 任务，在任务列表页面单击指定的任务名称可以查看任务详情。

任务详情页中通过拓扑图展示各个步骤的上下游关系和每个步骤的状态。通过鼠标在空白处拖拽可以移动拓扑图，通过鼠标滚轮滑动可以进行缩放，也可以通过右上方的缩放控件调整画布大小。拓扑图缩放是以鼠标指针为中心进行的，对于任务流程比较长的情况，可以先缩小，快速定位到关心的步骤，再进行放大，达到快速定位的目的。

重试或放弃失败任务

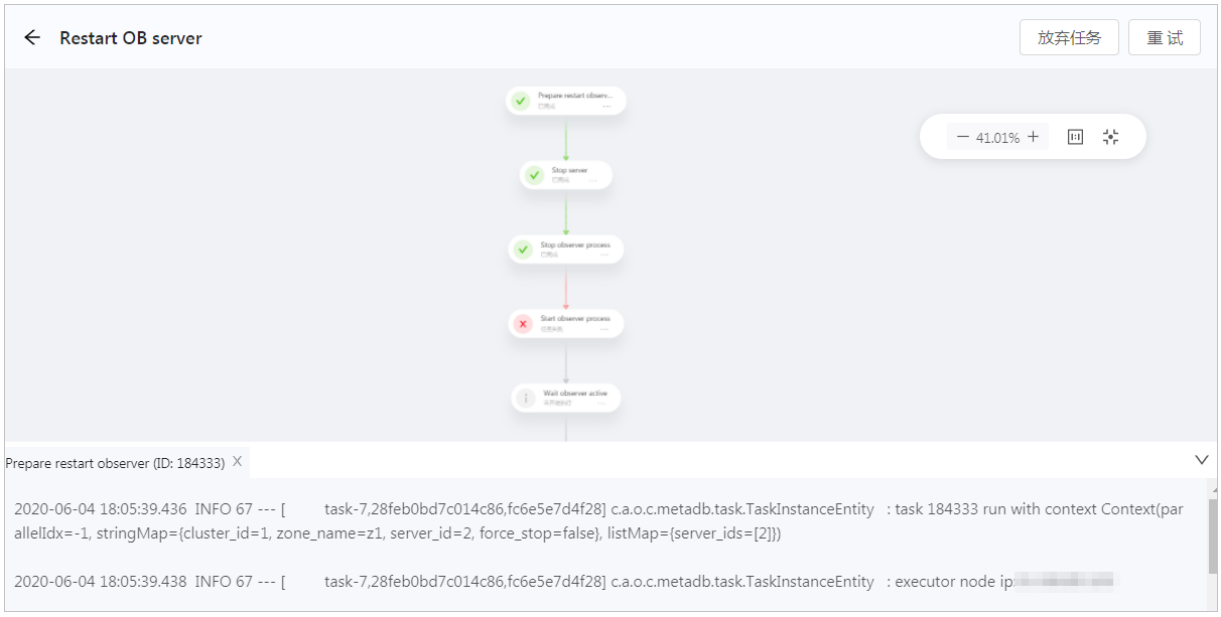
针对失败任务，OCP在任务拓扑图右上角提供重试或放弃任务的操作。

- 放弃任务：从失败处向任务的根结点，执行每一个步骤的回滚操作，主要用于任务的资源回收、状态修改等目的。
- 重试：从失败的节点处开始，依次执行回滚和再次执行的步骤，如果执行成功继续执行下游的节点。

查看步骤详情

对于任务中的指定步骤，根据节点的状态可以进行不同的操作，例如查看步骤ID和日志等。展开指定步骤右侧的隐藏菜单即可查看当前节点支持的相关操作。

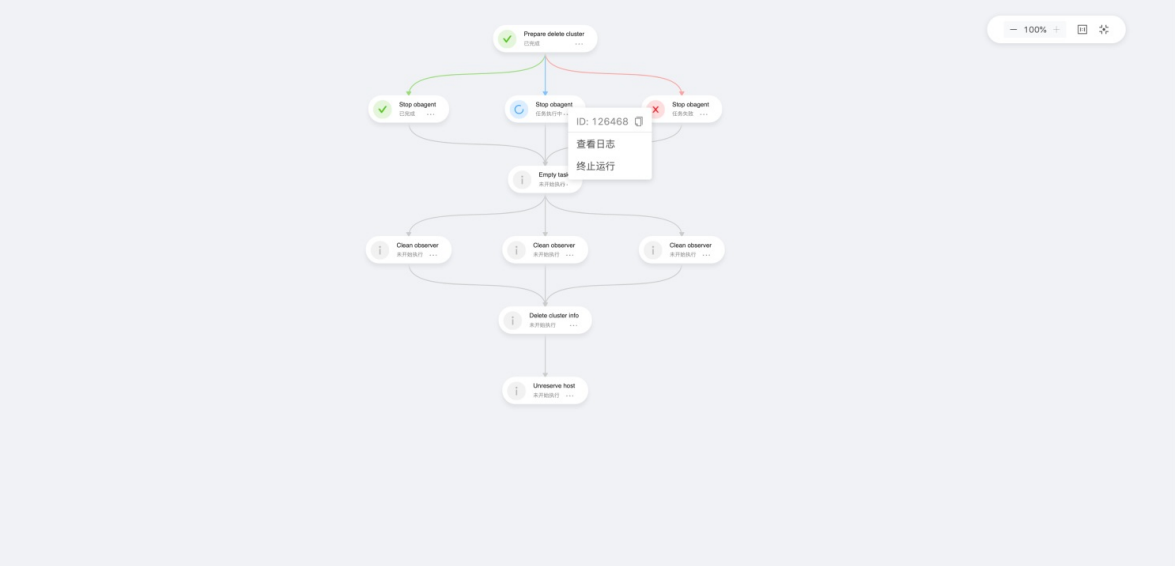
- 成功状态的节点可以进行日志查询。展开折叠菜单后，在屏幕的下方会出现折叠窗口展示日志信息，窗口可以拖动进行放大。



- 失败状态的节点除了可以查看日志信息，还可以重新运行和设置为成功。重新运行指重新运行当前节点，即先回滚再执行，成功执行的话继续执行下游节点。设置为成功是将当前节点的状态直接设置为成功状态，继续执行下游节点。



- 运行状态的节点可以查看日志或者终止运行，终止运行后该节点会变为失败状态。



- 等待运行的节点不可进行任何操作。

3.9. 系统参数管理

3.9.1. 查看系统参数

在 OCP 中，系统参数一般用于进行账户登录、备份恢复等功能相关的配置。

操作步骤

1. 登录 OCP 。默认进入**集群概览** 页面。
2. 在左侧导航栏中单击**系统管理 > 系统参数**。系统参数页面显示目前 OCP 支持修改取值的所有参数列表。
3. 在页面右上角的搜索框里输入需要查看的系统参数名称或参数关键字，并单击搜索图标，快速查找指定系统参数。
   - 页面显示的参数信息包含**参数名**、**当前取值**、**详细描述**和**修改时间**。
   - 参数列表也支持分页查询。

- 参数描述中，详细描述了该参数用途、取值范围、是否启动生效等信息

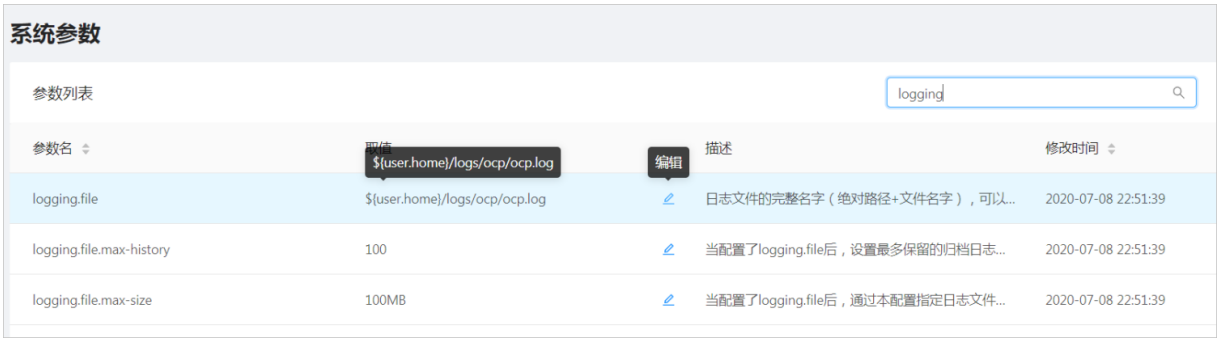


3.9.2. 修改系统参数

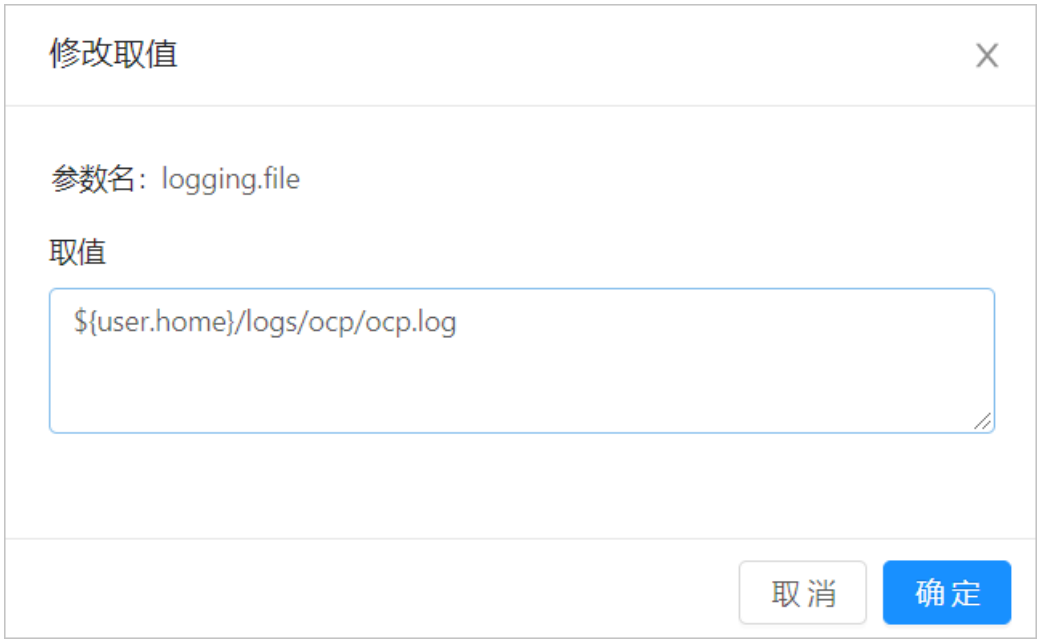
OCP中的系统参数一般用于进行账户登录、备份恢复等功能相关的配置，您可以通过修改系统参数的参数值来修改相关设置。

操作步骤

1. 登录 OCP 。默认进入**集群概览**页面。
2. 在左侧导航栏中单击 **系统管理 > 系统参数**。系统参数页面显示目前 OCP 支持修改取值的所有参数列表。
3. 在页面右上角的搜索框里输入需要查看的系统参数名称或参数关键字，并单击搜索图标，快速查找指定系统参数。
4. 单击指定参数对应的编辑图标。



5. 输入新的参数取值。



6. 单击**确定**，提交修改。
   - 修改参数后约10分钟生效。
   - 部分参数需要重启才能生效，请在附录1. OCP 配置参数列表中查看具体参数的详细说明。

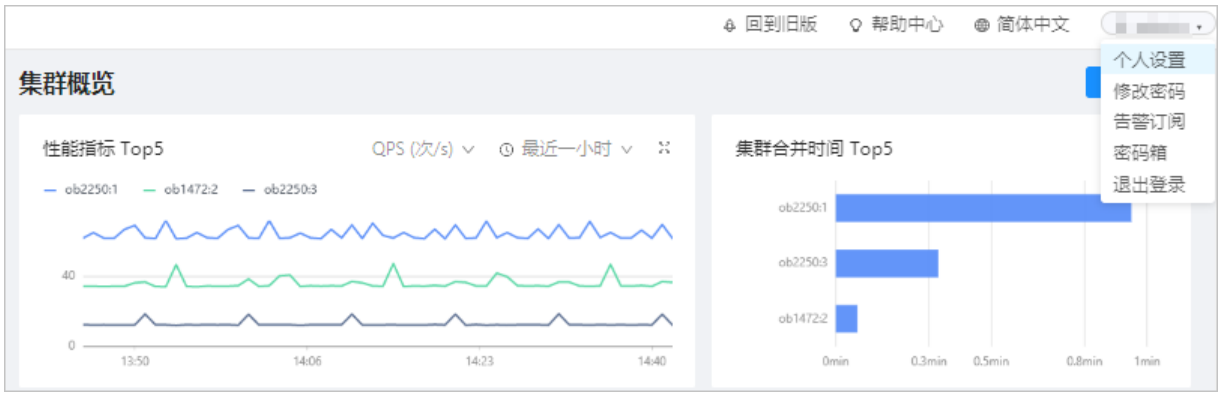
3.10. 个人中心

3.10.1. 设置个人信息

成功登录 OCP 后，系统右上角会显示当前登录用户名，单击当前登录用户名即可进入个人中心。您可以在个人中心设置您的个人信息，包括联系方式、所属组织和部门等信息。

操作步骤

1. 在 OCP 系统界面的右上角，单击当前登录用户名，选择**个人设置**。



2. 在**基本信息**页面，设置您的邮箱、手机号码、组织、部门和个人说明信息。



3. 完成后，单击**更新**。

3.10.2. 修改登录密码

首次登录 OCP 后，您需要尽快修改密码以免账号泄露。

背景信息

为确保账号安全，建议您定期修改密码。

操作步骤

1. 在 OCP 系统界面的右上角，单击当前登录用户名，选择**修改密码**。
2. 在弹出的**修改密码** 对话框中，输入旧密码、新密码及确认新密码。



修改密码时，新密码需要满足大小写字母、数字组合和长度要求：

- 长度：8位~32位
- 至少包含两位数字、两位大写字母、两位小写字母和两位特殊字符。特殊字符如下：._+@#$%

3. 单击**确定**。

3.10.3. 密码箱

个人密码箱用于集中管理当前登录用户登录 OceanBase 的用户名、密码等连接信息。当您在 OCP 中对某个 OceanBase 集群或租户执行管理操作时，OCP 需要验证当前用户密码箱对应的连接信息。

操作步骤

1. 在 OCP 系统界面的右上角，单击当前登录用户名，选择**密码箱**。进入 **密码箱** 页面。出于安全考虑，页面上的密码均显示为*******。
2. 您可以进行如下操作：

- 新建连接信息单击 **新建连接**，在弹出的 **新建连接** 对话框中，配置相关连接信息后，单击 **确定**。



- 删除连接信息选择待删除的连接信息，在对应的 **操作** 列中，单击 **删除**。
- 更新连接信息

> 说明
>
> - 如果您需要更新某个集群的连接信息，您需要先删除该条连接信息后重新添加。
> - 如果不删除原来的连接信息直接添加，则新添加的连接信息不会覆盖旧的连接信息，系统会提示重复的记录已存在。

3.11. 附录

3.11.1. 附录1. OCP 配置参数列表

本节列取了当前 OCP 所有系统参数，供 OCP 管理员参考。参数修改后默认约 10 分钟即可自动生效。部分参数需要重启OCP Server 后生效，详见参数说明列中包含“重启生效”的特别说明。

参数名	默认值	参数说明
ocp.active.standby.cluster.enabled	TRUE	是否支持 OceanBase 热备库功能，默认开启。OCP 老版本升级到 2.4.x 版本时，默认关闭此功能，如要开启，需要先确保 OBProxy 版本不小于 1.7.2
ocp.config-url.site.url		外部访问 OCP config url 服务的地址，默认为空，表示使用和 ocp.site.url 相同的值；如果 config url 服务单独部署，需配置和 ocp.site.url 不同的值
ocp.env	private	OCP 环境，2.3.x 之前版本兼容配置项
ocp.site.url	http://localhost:8080	外部访问 OCP 网站的地址: 要求以 http 开始（不支持https），包含 VIP 地址/域名/端口的网址，且结尾不含斜杠（/），比如 http://localhost:8080
ocp.system.default.language	zh-CN	系统默认语言（非前端语言设置），若不设置则使用 zh-CN，重启生效
ocp.system.default.timezone		系统默认时区，若不设置则使用 system default time zone，重启生效
ocp.version	2.4.2	OCP 版本号
server.port	8080	设置应用服务器启动的端口号，默认为 8080，重启生效
springdoc.api-docs.enabled	FALSE	OpenAPI 3 文档接口：是否启用; 默认是FALSE

安全与登录模块

参数名	默认值	参数说明
server.servlet.session.timeout	30m	登陆会话 /Session 超时的时间，默认是 30m，最少60s。如果不加后缀单位，则默认是秒
ocp.iam.auth	local	Web页面登录选项：目前支持 local（本地 MetaDB中的OCP 账户）、和 buc （Alibaba 集团员工账户）；默认local。重启生效。
ocp.iam.auth.basic.enabled	TRUE	是否启用 Basic Auth 登录模式，通常供程序和 SDK等客户端场景使用，默认true。本配置与ocp.iam.auth 可同时开启。重启生效。
ocp.iam.auth.whitelist	/api/v2/time,.*Action=(ObRootServiceInfo|ObRootServiceRegister|ObIDCRegionInfo|GetObProxyConfig|AllocateClusterIdByNameAndIdx|DeleteObRootServiceInfoByClusterName).*	OCP 免登录接口的白名单列表，以逗号分隔的接口URL字符串，支持正则表达式匹配，默认：/api/v2/time。重启生效。
ocp.iam.csrf.enabled	TRUE	是否启用CSRF跨站点请求伪造安全保护，通常基于网页登陆的方式都推荐要启用，默认true。重启生效。
ocp.iam.csrf.url.excluded	/api/v2/iam/login.*,.*Action=(ObRootServiceInfo|ObRootServiceRegister|ObIDCRegionInfo|GetObProxyConfig|AllocateClusterIdByNameAndIdx|DeleteObRootServiceInfoByClusterName).*	可指定跳过 CSRF 保护的 URL 列表，支持正则表达式匹配，默认为：/api/v2/iam/login.*。如果启用CSRF跨站点请求伪造安全保护，该选项才生效；URL是包含 path 和 query param 的完整字符串
ocp.iam.login.lockout-minutes	30	如果超过登录失败限制，则会临时阻止客户端IP的登录的时间，单位是分钟，默认是30分钟
ocp.iam.login.max-attempts	5	登录时允许连续输错用户名或密码的次数，如果超过限制则会临时阻止客户端IP的登录，默认是5次

OCP 元信息数据库连接（MetaDB）

参数名	默认值	参数说明
spring.datasource.druid.initialSize	10	初始化时建立物理连接的个数。重启生效
spring.datasource.druid.keepAlive	TRUE	连接池中的minIdle数量以内的连接，空闲时间超过minEvictableIdleTimeMillis（缺省值1800秒），则会执行keepAlive操作。重启生效
spring.datasource.druid.maxActive	100	最大连接池数量。重启生效
spring.datasource.druid.maxWait	2000	获取连接时最大等待时间，单位毫秒。重启生效
spring.datasource.druid.minIdle	2	最小连接池数量。重启生效
spring.datasource.druid.name	metadb-connect-pool	MetaDB 的 druid 连接池名称。重启生效
spring.datasource.druid.testWhileIdle	TRUE	建议配置为 true ，不影响性能，并且保证安全性。申请连接的时候检测。重启生效
spring.datasource.druid.validationQuery	SELECT 1 FROM DUAL	用来检测连接是否有效的 SQL。重启生效

OCP 监控数据库连接（MonitorDB）

参数名	默认值	参数说明
ocp.monitordb.database	chacha_dev	监控数据存储数据库database名
ocp.monitordb.driverClassName	com.alipay.oceanbase.obproxy.mysql.jdbc.Driver	jdbc驱动。重启生效

参数名	默认值	参数说明
ocp.monitordb.druid.initialSize	10	初始化时建立物理连接的个数。重启生效
ocp.monitordb.druid.keepAlive	TRUE	连接池中的minIdle数量以内的连接，空闲时间超过minEvictableIdleTimeMillis（缺省值1800秒），则会执行keepAlive操作。重启生效
ocp.monitordb.druid.maxActive	100	最大连接池数量。重启生效
ocp.monitordb.druid.maxWait	2000	获取连接时最大等待时间，单位毫秒。重启生效
ocp.monitordb.druid.minIdle	2	最小连接池数量。重启生效
ocp.monitordb.druid.name	monitordb-connect-pool	druid连接池名称。重启生效
ocp.monitordb.druid.testWhileIdle	TRUE	建议配置为true，不影响性能，并且保证安全性。申请连接的时候检测。重启生效
ocp.monitordb.druid.validationQuery	SELECT 1 FROM DUAL	用来检测连接是否有效的sql。重启生效
ocp.monitordb.host	10.101.194.179	监控数据存储数据库主机名
ocp.monitordb.password	root	监控数据存储数据库密码
ocp.monitordb.port	2888	监控数据存储数据库端口
ocp.monitordb.username	root@ocp_meta#obocp	监控数据存储数据库用户名

告警模块

参数名	默认值	参数说明
ocp.alarm.send.batch-max	20	一次捞取的新通知的最大数目
ocp.alarm.send.once-retry-times	3	每次发送告警的最大重试次数
ocp.alarm.send.once-timeout-ms	10000	每次发送发送告警的超时时间
ocp.alarm.send.period.ms	2000	通知发送间隔（毫秒），默认2000毫秒，重启生效
ocp.alarm.send.retry-timeout-minutes	60	重试超时时间（分钟），超过此时间的通知不再会重试发送，默认60分钟
ocp.alarm.send.total-failed-retry-times	3	通知发送失败时最大重试次数
ocp.alarm.send.total-timeout-seconds	60	单台机器发送一条通知的超时时间
ocp.backup.agent.relation.file.full-path.name	#{systemProperties['user.home'].concat('/ocp-server/etc/backup_agent_ob_relations_config.yaml')}	描述备份恢复组件版本，依赖的元数据库，以及与业务OB版本的匹配关系文件

备份恢复模块

参数名	默认值	参数说明
ocp.backup.alarm.backup-data-retention-days	7	备份数据保留的最近天数
ocp.backup.alarm.backup-liboblog-expire-days	7	备份liboblog保留的最近天数
ocp.backup.alarm.base-backup-last-finished-threshold	12960	已经有9天没有成功备份过基线，初始设置阈值为9天（12960分钟），单位为分钟

参数名	默认值	参数说明
ocp.backup.alarm.base-backup-timeout	10	基线备份调度超时时间（分钟）
ocp.backup.alarm.inc-backup-delay-threshold	3600	增量备份延迟告警阈值（秒）
ocp.backup.alarm.last-data-backup-max-interval-minutes	1440	检查最近时间内的基线备份任务是否失败，默认1天，单位分钟
backup.dbname.prefix	backup	备份恢复的元数据库名的自定义前缀，默认是backup开头。如果修改，则只会在ocp初始化或者升级时影响新建库的名称，对已建库不做任何修改

参数名	默认值	参数说明
ocp.agent.rpc.port	62888	OCP-Agent 服务端口，RPC使用
ocp.ob-agent.version	t-oceanbase-ob-agent-2.4.0-1884988.alios7.x86_64.rpm	OB-Agent 版本
ocp.ocp-agent.version	t-oceanbase-ocp-agent-2.4.0-1884049.alios7.x86_64.rpm	OCP-Agent 版本

OCP-Agent 与 OB-Agent

参数名	默认值	参数说明
ocp.agent.rpc.port	62888	OCP-Agent 服务端口，RPC使用
ocp.ob-agent.version	t-oceanbase-ob-agent-2.4.0-1884988.alios7.x86_64.rpm	OB-Agent 版本
ocp.ocp-agent.version	t-oceanbase-ocp-agent-2.4.0-1884049.alios7.x86_64.rpm	OCP-Agent 版本

OBProxy 连接

参数名	默认值	参数说明
ocp.system.obproxy.address	localhost	OBProxy 地址，用于连接 OB 集群
ocp.system.obproxy.metadb.cluster-name	obdv1	OBProxy 元数据所在集群（如OBProxy部署未使用MetaDB，本配置可忽略）
ocp.system.obproxy.metadb.database	obproxy	OBProxy 元数据库 DataBase（如OBProxy部署未使用MetaDB，本配置可忽略）
ocp.system.obproxy.metadb.user	root@obproxy	OBProxy 元数据库用户 （如OBProxy部署未使用MetaDB，本配置可忽略）
ocp.system.obproxy.port	2883	OBProxy 端口，用于连接OB集群

后台任务与调度

参数名	默认值	参数说明
ocp.task.executor.core-pool-size	16	任务执行线程池核心线程数，默认16，重启生效
ocp.task.executor.keep-alive-seconds	120	任务执行线程池IDLE线程保持存活时间（单位：秒），默认120秒，重启生效
ocp.task.executor.max-pool-size	64	任务执行线程池最大线程数，默认64，重启生效
ocp.task.executor.queue-capacity	1000	任务执行线程队列大小，默认1000，重启生效

OceanBase 远程连接管理

参数名	默认值	参数说明

obsdk.connectors.cache.cleanup.period.second	300	obsdk 清理连接器缓存的时间间隔，单位秒，取值范围30~1800，默认值300
obsdk.connectors.cache.max.idle.second	3600	obsdk 空闲连接器的过期时间，单位秒，取值范围 300~18000，默认值3600
obsdk.connectors.cache.size	50	obsdk 连接器缓存大小，取值范围 5~100，默认值 50
obsdk.connectors.print.sql	TRUE	obsdk 中 sql 打印开关，默认开启
obsdk.connectors.slow.query.threshold.millis	1000	obsdk中慢查询日志阈值，单位毫秒，默认值 1000毫秒
obsdk.ob.connection.mode	proxy	连接 OB 的方式，取值 proxy|direct，默认为 proxy

日志模块

参数名	默认值	参数说明
logging.file	${user.home}/logs/ocp/ocp.log	日志文件的完整名字（绝对路径+文件名字），可以用Linux/MacOS系统环境变量如${HOME}或者java系统变量${user.home}；默认是${user.home}/logs/ocp/ocp.log
logging.file.max-history	100	当配置了logging.file后，设置最多保留的归档日志文件的个数
logging.file.max-size	100MB	当配置了logging.file后，通过本配置指定日志文件大小，如 30MB、1GB等；默认是100MB
logging.level.com.alipay.ocp	INFO	设置ocp程序的日志级别，默认为INFO
logging.level.org.hibernate.SQL	INFO	设置spring sql框架的日志级别，默认为INFO

参数名	默认值	参数说明
logging.level.web	INFO	设置spring web框架的日志级别，默认为INFO

软件包管理

参数名	默认值	参数说明
ocp.file.default-block-split-size	1048576	OCP 文件模块 默认文件块大小（单位：字节），默认 1MB
ocp.file.local.built-in.dir	#{systemProperties['user.home'].concat('/ocp-server/lib')}	OCP 文件模块 内置文件的本地路径
ocp.file.local.dir	#{systemProperties['user.home'].concat('/data/files')}	OCP 文件模块 文件本地路径
ocp.file.max-concurrent-count	16	OCP 文件模块 单个节点同时处理文件的最大并发数，默认16
ocp.file.try-lock-timeout-milliseconds	10000	OCP 文件模块 单个节点锁超时毫秒数，默认10000

3.11.2. 附录2. OCP 资源单元规格表

本章节为您介绍 OCP 内置的资源单元规格（OB 2.0 及以上版本，支持的最小规格为 S1）。

规格名	最小 CPU 数	最大 CPU 数	最小内存	最大内存	分组
B	0.25	0.25	1G	1G	basic
LogOnlyNormal	1	1	2G	2G	logonly

规格名	最小 CPU 数	最大 CPU 数	最小内存	最大内存	分组
LogOnlySystem	5	5	35G	35G	logonly
S0	0.5	0.5	2G	2G	standard
S1	1.5	1.5	6G	6G	standard
S2	3	3	12G	12G	standard
S3	6	6	20G	20G	standard
S4	12	12	40G	40G	standard

3.11.3. 附录3. OCP 错误信息表

本章节为您介绍 OCP 错误信息表。

错误码	错误信息
1001	传入的参数为空或者不合法：{0}，请检查后重试
1002	没有找到指定{0}类型的记录，参数：{1}，请检查后重试
1003	
1004	请求处理超时，请重试
1005	请求处理遇到冲突，请重试
1006	重名的资源已存在（类型：{0}，参数：{1}），请换其他的名字后重试
1007	不支持的单位：{0}

错误码	错误信息
1010	传入的起止时间不正确，请修改后重试
1011	不支持的排序字段{0}
1012	未找到配置项: {0}
1113	序列获取下一个值失败
1200	rpc调用io异常, server: {0} port: {1}
1201	rpc调用执行失败, server: {0} port: {1} command: {2} code: {3} result: {4}
1500	操作OB失败，错误信息: {0}
1998	
1999	发生一个未知的错误：错误原因={0}，错误信息={1}，请联系管理员
3000	当前用户没有登录或登录出错了，请登陆后重试
3001	当前登录账户已被禁用，请联系管理员
3002	当前登录账户已超期，请联系管理员
3003	当前登录账户已被锁定，请联系管理员
3004	用户名或者密码错误，请重试。您还剩余{0}次机会
3005	密码已过期，请联系管理员重制密码
3006	用户名或者密码错误，请重试。您还剩余{0}次机会

错误码	错误信息
3007	当前账户的登录已被暂时禁止，请联系管理员
3008	用户名{0}不合法，小写英文字母开头，支持小写英文、数字、下划线和'.'的组合，长度为 4 ~ 48 位
3009	密码不合法，密码长度8~32位，至少包含2位数字、2位大写字母、2位小写字母和2位特殊字符(._+@#$%)
3010	该操作需要集群{0}下租户{1}的{2}用户密码，请先添加到密码箱
3011	账户的登陆密码与提供的密码不匹配，请检查后重试
3012	错误的密码ID格式
3013	密码箱中不存在ID为{0}的密码
3100	没有权限进行对应操作，请联系管理员
3101	权限的表达式不能为空，请检查权限设置后重试
3102	给定的权限表达式{0}无效，请检查权限设置后重试
3103	系统默认的角色{0}不可以被删除
3104	系统默认的账户{0}不可以被删除
3105	{0}是系统预留的用户名，请选择其他用户名后重试，请尽量避免含admin/sys/dba/proxy等词汇
4000	
4001	任务 {0} 不存在
4002	Dag {0} 不存在

错误码	错误信息
4003	存在循环结构
4004	任务上下文错误
4005	在主机{1}上执行命令{0}失败
4006	在主机{1}上保存文件{0}失败
4007	任务{0}, 状态{1}, 不能进行此操作
5000	
6000	
7000	指定的资源{0}:{1}包含一个或多个子资源，请清空子资源后再重试
7001	主机<{0}:{1}>已经存在，请修改后重试
7002	测试建立SSH连接到目标主机{0}@{1}:{2}失败，原因是{3}，请修改连接信息后重试
7004	当前主机{0}离线或使用中，请释放主机并恢复状态为{1}再重试
7005	主机<{0}:{1}>不存在
7006	当前主机{0}状态不支持此操作
7007	主机{0}所在idc与目标observer的idc不匹配
7008	主机{0}所在idc与目标zone的idc不匹配
7009	主机{0}所在region与目标observer的region不匹配

错误码	错误信息
7010	不支持在主机{0}部署多个同类型的服务{1}，请选择其他主机再重试
7011	不支持在容器类型主机{0}部署多个服务{1}，请选择其他主机再重试
7012	当前主机{0}已经部署了agent服务，请卸载后重试
7013	IP为<{0}>的主机不存在
7014	IP为<{0}>的主机不唯一
7500	文件 {0}:{1} 不存在
7501	找不到文件块 {0}:{1}
7502	已存在类型为 {0} 名称为 {1} 的文件
7503	文件后缀校验错误，后缀为{0}， 应为{1}
7504	文件真实类型校验错误，类型为{0}，应为{1}
7505	等待其它请求操作文件完成超时，请重试
8000	目前无法访问你的个人中心，请过一段时间重试
8001	不允许删除别人的密码箱内容，参数{0}.
8002	相同的连接信息已存在，请删除原记录后重建，参数{0}.
8500	参数{0}为空或者不合法：{1}，需要http/https开头包含域名、端口号的网站根路径，请检查后重试
10000	

错误码	错误信息
11000	
11001	指定的集群id={0}不存在
11002	指定的集群name={0}不存在
11003	指定的集群id={0}，当前类型不支持此操作
11004	指定的集群id={0}，当前版本不支持此操作，最小支持版本={1}
11005	传入的root server信息不正确
11007	集群{0}当前状态不支持此操作
11008	集群{0}状态为{1}时才支持此操作
11009	rootserver json 不合法 {0}
11010	集群id={0}下未找到root server
11011	密码错误，请确认输入的旧密码是否正确
11012	传入的密码参数不能为空
11013	传入的rpm包名不能为空
11014	待创建集群已经存在
11015	当前集群存在备集群，不允许此操作
11016	备集群暂不支持升级操作

错误码	错误信息
11017	集群{0}的主集群未找到
11018	集群{0}下发现多个主集群
11019	集群{0}的server列表为空
11020	该集群上一次运维任务还未结束
11021	传入的机器已经部署了集群
11022	集群热备库功能未开启
11023	指定的集群{0}，不具备切换条件
11024	指定的备集群{0}，添加还未完成
11030	传入的机器数量与待导入集群observer数量不符
11031	传入的机器IP与待导入集群observer的IP不符
11032	请先导入主集群，再导入备集群
11033	待导入的集群已经存在
11034	待导入的集群下zone={0}缺少region信息
11035	待导入的集群下zone={0}缺少idc信息
11050	未找到指定参数={0}
11051	参数{0}的值{1}格式错误
11052	设置集群参数生效范围只允许指定集群、ZONE或SERVER，当前的值{0}不合法，请重试

错误码	错误信息
11100	连接集群{0}失败, 请检查密码箱中该集群sys租户{1}用户密码是否正确，sys租户白名单设置是否正确，以及网络是否连通
11201	集群id={0}的version={1}合并版本号不存在
11202	合并查询超时，请重试
11203	不支持的合并统计指标名称 {0}
11204	合并参数更新失败，可能存在冲突请求，请几秒钟之后查看确认是否更新成功
11205	无法发起从非PRIMARY集群的合并，集群id={0}的集群是STANDBY集群
11500	
11501	在主机{0}上启动ob agent失败
12000	
12001	输入的zone名格式不正确，需要以英文字母开头，可包括大小写字母、数字和下划线，长度在2与64之间
12002	集群id={0}下未找到zone={1}
12003	集群id={0}已存在zone={1}
12004	指定的zone={0}，当前状态不支持此操作
12005	停止zone={0}失败

错误码	错误信息
12006	指定的zone={0}上存在unit，不允许删除
12007	该zone上一次运维任务还未结束
12008	传入的zone名有重复
13000	
13001	未找到指定的observer，id={0}
13002	未找到指定的observer，ip={0}，port={1}
13003	指定的observer={0}，当前状态不支持此操作
13004	observer {0} 未退出
13005	observer {0} 启动失败
13006	目标observer的ip不合法
13007	目标observer的rpc端口不合法
13008	目标observer的sql端口不合法
13009	剩余的server数不能小于租户={0}在zone={1}下的unit数
13010	obagent {0} 未退出
13011	observer无法删除，原因：{0}
13012	该observer上一次运维任务还未结束
13013	未找到指定的observer，ip={0}

错误码	错误信息
13014	指定的observer不唯一，ip={0}
15000	
15001	指定的规格id={0}不存在
15002	指定的规格名={0}不存在
15003	输入的规格名格式不正确，需要以英文字母开头，可包括大小写字母、数字和下划线，长度在2与32之间
15004	输入的规格名={0}已经存在
15005	输入的 cpu 数值不正确，最小值0.5
15006	输入的 memory 数值不正确，最小值1073741824，单位Byte
15007	输入的 disk 数值不正确，最小值10737418240，单位Byte
15008	输入的 iops 值不正确，最小值128
15009	
15010	输入的单元规格类型不正确，可选值 SYSTEM|CUSTOM
15020	输入的租户名格式不正确，需要以英文字母开头，可包括大小写字母、数字和下划线，长度在2与64之间
15021	集群{0}下租户{1}已经存在
15022	集群{0}下未找到租户{1}
15023	指定的租户id={0}不存在

错误码	错误信息
15024	指定的租户id={0}，已经删除
15025	指定的租户{0}，当前状态不支持此操作
15026	指定的租户{0}，不允许删除
15027	指定的租户{0}，不允许进行该操作
15028	指定的租户id={0}，设置密码失败
15030	输入的租户模式不正确，可选值MYSQL|ORACLE
15031	输入的primary zone长度不正确，需小于128
15032	输入的primary zone不正确
15033	输入的白名单长度不正确，需小于65535
15034	输入的密码长度不正确，需小于32
15035	输入的description长度不正确，需小于1024
15036	oracle模式最小支持版本={0}, 当前版本={1}
15037	传入的zone不正确
15040	传入的zone={0}不合法
15041	未找到zone={0}
15042	指定的zone={0}下，没有可用的资源池
15043	未找到指定的unit={0}

错误码	错误信息
15044	指定unit={0}为所属zone下唯一的资源单元，不允许删除
15045	传入的unit个数不合法
15046	传入的unit个数不能超过该zone下处于active状态的server数量
15050	租户{0}下没有可用server
15060	未找到指定的租户参数={0}
15061	指定的租户参数={0}为只读类型
15062	指定的租户参数={0}的旧值获取不到
15080	传入的副本类型不正确
15082	租户{0}, locality修改进度未找到
15083	租户{0}, locality修改未完成
15084	租户{0}, 资源池缩容未完成
15100	连接租户{0}失败，请检查密码箱中所存的该租户下root用户的密码是否正确，以及租户白名单设置是否正确
20000	备份恢复不支持{0}版本的ob
20001	存在配置文件{0},但没有适用于{1}版本的集群{2}的备份组件
20002	ip为{0}的安装目录{1}已存在，请手动检查
20003	集群{0}需要先进行备份调度配置

错误码	错误信息
20004	参数错误
20005	组件中未配置适用于{0}版本的集群{1}的备份配置，请配置！
20006	适用于该集群的组件{0}均不在线，请检查！
20007	集群或者租户的基线备份任务已经存在
20008	集群的立即备份失败
20009	发起恢复数据失败
20010	备份配置{0}中的URI为空
20011	备份配置{0}中的endpoint为空
20012	备份配置{0}中的accessKeyId为空
20013	备份配置{0}中的secretAccessKey为空
20014	备份文件中找不到租户{0}的备份版本号
20015	解析备份文件中的租户信息失败
20016	新增配置文件失败
20017	配置正在被组件{0}使用，删除失败！
20018	配置名{0}已存在！
20020	{0}组件失败！

错误码	错误信息
20021	该主机已经有在运行中的运维任务(TaskId:{0})，请等该任务完成后再发起新的运维操作！
20030	数据备份未启动
20031	日志备份未启动
20032	同一集群或者同一集群下的租户，只能有一份调度配置。已存在cluster:{0},tenant:{1},请删除已有配置后，再增加。
20040	在恢复租户过程中，创建ResourceUnit:{0}失败,{1}
20041	输入的pool list中{0}格式有误，请检查！
20042	输入的pool list中的pool已存在，请修改pool name!
20043	提交的新租户{0}已经在恢复中,请检查！
20050	集群{0}可能发生过主备库切换，暂不支持备库{1}备份。请将当前主库{1}备份到新的备份目录！
21000	
22000	

3.11.4. 附录4. OCP 告警模板变量

告警模板变量用于告警规则配置、告警通道配置。

类别	名称	描述
告警规则	app_type	应用类型
告警规则	alarm_type	告警类型

类别	名称	描述
告警规则	alarm_target	告警对象
告警规则	alarm_scope	范围
告警规则	alarm_level	告警级别
告警规则	alarm_evaluation_interval	告警评估间隔（秒）
告警规则	alarm_duration	持续时长（秒）
告警规则	alarm_status	告警状态
告警规则	alarm_active_at	告警触发时间
告警规则	alarm_resolved_at	告警消除时间
告警规则	alarm_last_interval	持续时长（秒）
告警规则	alarm_name	告警名称
告警规则	value	指标值
告警规则	alarm_threshold	告警阈值
告警规则	alarm_updated_at	告警更新时间
告警规则	service	服务名
告警规则	ob_cluster_group	OB集群组
告警规则	ob_cluster	OB集群
告警规则	ob_cluster_id	OB集群ID

类别	名称	描述
告警规则	ob_tenant	OB租户
告警规则	host_ip	主机IP
告警规则	app_cluster	应用集群
通知消息	app_type	应用类型
通知消息	alarm_type	告警类型
通知消息	alarm_target	告警对象
通知消息	alarm_scope	范围
通知消息	alarm_level	告警级别
通知消息	alarm_evaluation_interval	告警评估间隔（秒）
通知消息	alarm_duration	持续时长（秒）
通知消息	alarm_status	告警状态
通知消息	alarm_active_at	告警触发时间
通知消息	alarm_resolved_at	告警消除时间
通知消息	alarm_last_interval	持续时长（秒）
通知消息	alarm_name	告警名称
通知消息	value	指标值
通知消息	alarm_threshold	告警阈值

类别	名称	描述
通知消息	alarm_updated_at	告警更新时间
通知消息	alarm_summary	告警概述
通知消息	alarm_description	告警详情
通知消息	alarm_level_color	告警等级对应的颜色
通知消息	alarm_id	告警ID
通知消息	alarm_url	告警在OCP站点的访问URL
通知消息	service	服务名
通知消息	ob_cluster_group	OB集群组
通知消息	ob_cluster	OB集群
通知消息	ob_cluster_id	OB集群ID
通知消息	ob_tenant	OB租户
通知消息	host_ip	主机IP
通知消息	app_cluster	应用集群
聚合消息	app_type	应用类型
聚合消息	alarm_type	告警类型
聚合消息	alarm_target	告警对象
聚合消息	alarm_scope	范围

类别	名称	描述
聚合消息	alarm_level	告警级别
聚合消息	alarm_evaluation_interval	告警评估间隔（秒）
聚合消息	alarm_duration	持续时长（秒）
聚合消息	alarm_status	告警状态
聚合消息	alarm_active_at	告警触发时间
聚合消息	alarm_resolved_at	告警消除时间
聚合消息	alarm_last_interval	持续时长（秒）
聚合消息	alarm_name	告警名称
聚合消息	value	指标值
聚合消息	alarm_threshold	告警阈值
聚合消息	alarm_updated_at	告警更新时间
聚合消息	alarm_summary	告警概述
聚合消息	alarm_description	告警详情
聚合消息	alarm_level_color	告警等级对应的颜色
聚合消息	alarm_id	告警ID
聚合消息	service	服务名
聚合消息	ob_cluster_group	OB集群组

类别	名称	描述
聚合消息	ob_cluster	OB集群
聚合消息	ob_tenant	OB租户
聚合消息	host_ip	主机IP
聚合消息	app_cluster	应用集群
聚合消息	alarm_group_by	告警聚合分组
聚合消息	alarm_count	告警数量
通道配置	app_type	应用类型
通道配置	alarm_type	告警类型
通道配置	alarm_target	告警对象
通道配置	alarm_scope	范围
通道配置	alarm_level	告警级别
通道配置	alarm_name	告警名称
通道配置	alarm_summary	告警概述
通道配置	alarm_description	告警详情
通道配置	service	服务名
通道配置	ob_cluster_group	OB集群组
通道配置	ob_cluster	OB集群

类别	名称	描述
通道配置	ob_tenant	OB租户
通道配置	host_ip	主机IP
通道配置	app_cluster	应用集群
通道配置	message	消息（由消息模板或聚合消息模板生成）
通道配置	message_json	消息 json字符串（message值的json格式）
通道配置	recipients_uids	接收人用户ID列表，逗号分隔形式
通道配置	at_recipients	@接收人列表 的值
通道配置	recipients_json_array	接收人列表 json 数组，用于钉钉群消息 @接收人

3.11.5. 附录5. 后台任务列表

本章节主要列举了OCP内部的常驻的后台任务。这些后台任务主要是为了维护OCP以及被管理目标的状态检查、数据清理等目的，由OCP Server定期触发并执行的一类任务总称。

后台任务的调度和运行历史，可以参考OCP 元数据库（MetaDB）中的 'dag_instance' 和 'task_instance' 表的详情（通过 dag_instance.id = task_instance.dag_id 进行join查询）。 dag_instance 记录的是整个后台任务的执行详情，而 task_instance记录的是当前后台任务下每个子任务的执行详情。

后台任务执行的日志，可以到"logging.file"配置参数指定的OCP日志文件中，根据特定关键字搜索。OCP日志文件默认为 ${user.home}/logs/ocp/ocp.log。

以下是常驻后台任务列表，包含任务名、任务描述、备注、状态排查等信息。这个信息可以做OCP系统管理员的参考和问题排查指南。

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP日志

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Archive alarm history data	归档过期的告警屏蔽配置与通知信息	0 0 3 * * *	ocp2_alarm_filter（告警屏蔽配置） ocp2_alarm_notification（告警通知） ocp2_alarm_filter_history（屏蔽配置历史） ocp2_alarm_notification_history（通知历史）	"Archive expired alarm history data: result=archive [filter|notification] success, affectRows=xxx"
Sync all cluster info	定期同步OCP管理的OceanBase集群信息，并更新到ob_cluster表。 仅当ob_cluster表的operateStatus为NORMAL，且clusterStatus为RUNNING时进行更新。	0 * * * * *	ob_cluster（OCP管理的集群信息）	"cluster sync success, cluster=[id=xx,name=xx,..]" -- 成功 "cluster sync failed, cluster=xxx, exceptionType=xxx, message=xxx" -- 异常 -- 同步过程日志 " syncForCluster start, clusterId=xx init clusterSyncContext, context=xx create clusterOperator, clusterId=xx get current from ob cluster, clusterId=xx updateZones done, clusterId=xx syncForCluster done, clusterId=xx "

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Refresh config properties	刷新OCP的配置信息	0 */10 * * * *	config_properties（OCP 系统参数配置）	"Refresh config properties on [host:port] at [date & time] "
Maintain partition for monitordb	处理monitor租户的分区，每天会按照特定的策略为分区表新建新的分区，删除过期的分区	0 0 1 * * *	Monitor DB数据库: ob_cluster_system_event (采集的OceanBase Root Service事件) ob_metric_data_1（每秒采集的监控数据） ob_metric_data_60 (每60秒采集的监控数据)	-- 分区维护成功 "partition maintain start partition maintain success" -- 分区维护失败 "partition maintain failed [exception msg]"
Sync system event	同步系统事件，从OB集群中，定期同步Root Service事件到monitordb的ob_cluster_system_event	0 */10 * * * *	Monitor DB数据库: ob_cluster_system_event (采集的OceanBase Root Service事件) OceanBase内部表: oceanbase.__all_rootservice_event_history	-- 同步成功 "Batch saved [记录数] events to database for cluster [cluster id] Sync-ed [记录数] system events for cluster [cluster id], startTime [开始时间], endTime [结束时间], limit [limit], offset [offset] Finished sync system events: cluster [cluster id], startTime [起始时间], endTime [结束时间] " -- 同步失败 "An exception occurred when saving events [exception log]"

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Check all cluster status	定期检查 ob_cluster表中 OceanBase 集群的“存活状态”（通过测试建立数据库连接来检查）。如果不成功，则打印错误日志，并发送告警。	0 * * * * *	ob_cluster （OCP 管理的集群信息）	-- cluster能连接 "check cluster status okay, clusterId=[cluster id], obVersion=[ob version]" -- cluster不能连接 "check cluster status unavailable, clusterId=[cluster id], failedReason=[exception msg]" -- 告警类别与关键字 "ob_cluster_status_check_failed" + "cluster connect check"

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Check host status	检查host上pos_proxy能否执行远程命令（whoami），如果执行失败会发送报警，并修改主机状态(OFFLINE)，如果正常会将状态修改为正常状态(AVAILABLE, ONLINE) 检查host上node_exporter数量，如果数量不对，则打印异常信息并发送告警	0 * * * * *	compute_host.status compute_host_agent.last_available_time ocp_exporter_address	"Host agent check result: current OS user is root" -- agent检查成功 "Check exporter status: host [id] desired exporter count: [number], actual exporter count: [number]" -- exporter检查成功 "Host agent check failed: host [host id] reason [exception msg]" -- agent检查失败 "Alarm: host_unavailable for host ip [ip address]" -- exporter检查失败 "Alarm: no_enough_exporter for host [id]" -- exporter检查失败 "host_unavailable" -- agent失败告警类别 "no_enough_exporter" -- exporter失败告警类别

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Submit scheduled dag	实现了类似于主机上crontab的功能，用于提交用户自定义的定时任务，会检查用户定义的调度表达式和当前实际比较，满足条件，则提交用户定义的任务；否则更新dag_schedule表对应记录状态为ABNORMAL	0 * * * * *	dag_schedule （任务调度详情） dag_instance (后台任务运行详情)	-- 当前无用户自定义任务 "no cron job available in dag_schedule" -- 提交自定义任务出错 "failed to get template for class [java class name], got exception: [exception msg]"
Clean ocp task log files	清理OCP主机上task日志	0 0 1 * * *		OCP 清理日志的 Python 脚本： 'user.home' -> '/ocp-server/python-task/src/task/ob_task_clean_log.py' 待清理的Task 日志目录： 'user.home' -> '/logs/task/' 清理日志的Python进程日志文件： 'user.home' -> '/logs/task/task.2.xxx.log' OCP 清理任务异常日志： "failed to clean task log files: [exception msg]"

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Collect all cluster compaction info	收集所有集群的合并信息	0 * * * * *	ob_cluster_compaction (OB 集群合并记录) ob_zone_compaction （OB Zone合并记录） ob_server_compaction （OB Server 合并记录） ob_tenant_compaction_stats （OB 租户合并记录）	-- 收集成功 "Update cluster [cluster id] compaction status: from [xx] to [xx]" "Compaction post collect done, clusterId= [cluster id], version= [version]" -- 收集失败 "Skip collect for not RUNNING cluster, clusterId= [cluster id], status=[xx]" "Skip collect for version 1, the first compaction will be version 2" "Validate compaction version failed: [reason]"

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Sync tenant information	同步租户信息	30 0/2 * * * ?	ob_tenant (OB 租户信息表)	-- 同步成功 "[SyncAllTenantInfoTask] begin to sync tenant." "[SyncAllTenantInfoTask] sync tenants of specified cluster, clusterId=[cluster id]" "[SyncAllTenantInfoTask] sync tenant finished." -- 同步失败 "Zero tenants found for cluster [cluster id]"
Detect alarms on schedule	备份恢复相关的告警检测器	0 */1 * * * *	dag_schedule （任务调度详情） dag_instance (后台任务运行详情)	--执行成功 generate tasks, templateName=Detect alarms, nodeCount=1, taskCount=1 set state running for task: [task id] set state successful for task: [task id] --执行失败 打印异常调用栈 关键词：BackupAlarmService

任务名称	任务描述	调度表达式	问题排查 - 数据库	问题排查 - OCP 日志
Clean expired data and log files on schedule	备份恢复相关的定时清理器，清理过期数据和清理过期日志	0 0 0 * * *	dag_schedule （任务调度详情） dag_instance (后台任务运行详情)	--执行成功 generate tasks, templateName=Clean expired data and log files, nodeCount=1, taskCount=1 set state running for task: [task id] set state successful for task: [task id] --执行失败 打印异常调用栈 关键词：BackupCleanService
Check system obproxy	检查OCP 所使用的系统obproxy的健康状态	0 * * * * *	OCP 系统参数： ocp.system.obproxy.address ocp.system.obproxy.port	--系统obproxy状态正常 打印日志“system obproxy is running” --系统obproxy状态不正常 打印日志“system obproxy is down” 同时，会推送一条告警 --检查方法执行失败 打印异常调用栈，“check connection of system obproxy failed. exception: xxx”

调度表达式

调度使用的如下格式的Cron表达式，包含了6位字段，如下：

```
second, minute, hour, day of month, month, day(s) of week
```

Cron表达式字段和取值

字段名	取值范围	支持的特殊字符
second	0-59	,-*/
minute	0-59	,-*/
hour	0-23	,-*/
day of month	1-31	,-*/?
month	1-12 或 JAN-DEC	,-*/
day of week	1-7 或 MON-SUN	,-*/?

表达式示例：

表达式	说明
*0 0 * * * **	“0秒0分” - 每小时运行一次，在0秒0分启动
**/10 * * * * **	每10秒钟运行一次
*0 0 8-10 * * **	每天的8, 9 和10点钟运行一次，在0秒0分启动
*0 0/30 8-10 * * **	每天8:00, 8:30, 9:00, 9:30,10:00和10:30运行
*0 0 9-17 * * MON-FRI*	从周一到周五的上午9点到下午5点运行
*0 0 9-17 * * 1-5*	从周一到周五的上午9点到下午5点运行，使用数字表达周几，1代表周一，7代表周日。与crontab的表达式稍有不同

0 0 0 25 12 ?	每年12/25日凌晨0点0分0秒

3.11.6. 附录6. 监控指标

本章节为您介绍监控指标。

指标分组	指标名称	指标解释	计算表达式
CPU 使用率	cpu_percent	CPU 使用率	100 * (1 - sum(rate(node_cpu_seconds_total{mode="idle", @LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(node_cpu_seconds_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS))
IO 吞吐率	read	每次读取数据量	avg(rate(node_disk_read_bytes_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / 1048576
IO 吞吐率	write	每次写入数据量	avg(rate(node_disk_written_bytes_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / 1048576
IO 耗时	read	每秒读取平均耗时	1000000 * avg(rate(node_disk_read_time_seconds_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
IO 耗时	write	每秒写入平均耗时	1000000 * avg(rate(node_disk_write_time_seconds_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
IOPS	read	每秒读次数	avg(rate(node_disk_reads_completed_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
IOPS	write	每秒写次数	avg(rate(node_disk_writes_completed_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
Linux 系统负载	load1	过去1分钟系统平均负载	avg(node_load1{@LABELS}) by (@GBLABELS)
Linux 系统负载	load15	过去15分钟系统平均负载	avg(node_load15{@LABELS}) by (@GBLABELS)
Linux 系统负载	load5	过去5分钟系统平均负载	avg(node_load5{@LABELS}) by (@GBLABELS)
MEMStore	active	活跃 MEMStore 大小	sum(sysstat_value{metric_group="sysstat",stat_id="130000",@LABELS}) by (@GBLABELS) / 1048576
MEMStore	limit	MEMStore的limit	sum(sysstat_value{metric_group="sysstat",stat_id="130004",@LABELS}) by (@GBLABELS) / 1048576
MEMStore	total	MEMStore 总大小	sum(sysstat_value{metric_group="sysstat",stat_id="130001",@LABELS}) by (@GBLABELS) / 1048576

指标分组	指标名称	指标解释	计算表达式
MEMStore	trigger	触发合并阈值	sum(sysstat_value{metric_group="sysstat",stat_id="130002",@LABELS}) by (@GBLABELS) / 1048576
QPS	all	每秒处理 SQL 语句数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40004",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40006",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40008",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
QPS	delete	每秒处理 Delete 语句数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40008",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
QPS	insert	每秒处理 Insert 语句数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
QPS	replace	每秒处理 Replace 语句数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40004",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
QPS	select	每秒处理 Select 语句数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
QPS	update	每秒处理 Update 语句数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40006",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
			(sum(rate(sysstat_value{metric_group="sysstat",stat_id="40003",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40005",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40007",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",

指标分组	指标名称	指标解释	计算表达式
			stat_id="40009",@LABELS}[@INTERVAL]) by (@GBLABELS)) by
SQL 响应时间	all	服务端每条 SQL 语句平均处理耗时	(@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat", stat_id="40001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)) /(sum(rate(sysstat_value{metric_group="sysstat",stat_id="40002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat", stat_id="40004",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat", stat_id="40006",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat", stat_id="40008",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat", stat_id="40000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS))
SQL 执行计划类别	distributed	每秒处理分布式执行计划数	sum(rate(sysstat_value{metric_group="sysstat", stat_id="40012",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
SQL 执行计划类别	local	每秒处理本地执行数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40010",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
SQL 执行计划类别	remote	每秒处理远程执行计划数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40011",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
TPS	trans_count	每秒处理事务数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="30005",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
事件等待_时间	wait_time	等待事件平均耗时	sum(rate(time_wait{metric_group="waitevent",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(total_waits{metric_group="waitevent",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
事件等待_次数	wait_count	每秒等待事件次数	sum(rate(total_waits{metric_group="waitevent",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
事务响应时间	trans_time	服务端每个事务平均处理耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="30006",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="30005",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
事务日志数	log_count	每秒提交的事务日志数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="30002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
事务日志耗时	sync_time	每次事务日志网络同步平均耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="30000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="30001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
事务日志耗时	write_disk	每次事务日志写盘平均耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="80041",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="80040",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
事务日志量	log_size	每秒提交的事务日志大小	sum(rate(sysstat_value{metric_group="sysstat",stat_id="80057",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
内存	buffers	内核 Buffer Cache 大小	avg(node_memory_Buffers_bytes{@LABELS}) by (@GBLABELS) / 1073741824
内存	free	可用物理内存大小	avg(node_memory_MemFree_bytes{@LABELS}) by (@GBLABELS) / 1073741824
内存	used	使用物理内存大小	(avg(node_memory_MemTotal_bytes{@LABELS}) by (@GBLABELS) - avg(node_memory_MemFree_bytes{@LABELS}) by (@GBLABELS) - avg(node_memory_Cached_bytes{@LABELS}) by (@GBLABELS) - avg(node_memory_Buffers_bytes{@LABELS}) by (@GBLABELS)) / 1073741824
			(sum(rate(sysstat_value{metric_group="sysstat",stat_id="40003",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40005",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40007",@LABELS}[@INTERVAL]) by (@GBLABELS)) by

指标分组	指标名称	指标解释	计算表达式
响应时间	all	服务端每条 SQL 语句平均处理耗时	(@GBLABELS) + sum(rate(sysstat_value{ metric_group="sysstat", stat_id="40009",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{ metric_group="sysstat", stat_id="40001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)) /(sum(rate(sysstat_value{metric_group="sysstat",stat_id="40002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{ metric_group="sysstat", stat_id="40004",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{ metric_group="sysstat", stat_id="40006",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{ metric_group="sysstat", stat_id="40008",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{ metric_group="sysstat", stat_id="40000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS))

指标分组	指标名称	指标解释	计算表达式
响应时间	delete	服务端每条 Delete 语句平均处理耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40009",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="40008",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
响应时间	insert	服务端每条 Insert 语句平均处理耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40003",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="40002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
响应时间	replace	服务端每条 Replace 语句平均处理耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40005",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="40004",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
响应时间	select	服务端每条 Select 语句平均处理耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="40000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
响应时间	update	服务端每条 Update 语句平均处理耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="40007",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="40006",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
容量_分区数量	partition_count	分区数量	sum(partition_count{metric_group="all_meta_table",@LABELS}) by (@GBLABELS)
容量_表数量	table_count	表数量	max(table_count{metric_group="all_table",@LABELS}) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
查询响应时间	all	服务端每条 SQL 语句平均处理耗时	(sum(rate(sysstat_value{metric_group="sysstat",stat_id="40003",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40005",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40007",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40009",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)) /(sum(rate(sysstat_value{metric_group="sysstat",stat_id="40002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40004",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40006",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="40008",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",

指标分组	指标名称	指标解释	计算表达式
			stat_id="40000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS))
活跃会话数	active_session	当前活跃会话数	sum(active_sessions{metric_group="all_virtual_processlist",@LABELS}) by (@GBLABELS)
等待事件	wait_count	每秒等待事件次数	sum(rate(total_waits{metric_group="waitevent",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
等待事件耗时	wait_time	等待事件平均耗时	sum(rate(time_wait{metric_group="waitevent",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(total_waits{metric_group="waitevent",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
等锁耗时	wait_time	写锁平均等待耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="60023",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / (sum(rate(sysstat_value{metric_group="sysstat",stat_id="60021",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) + sum(rate(sysstat_value{metric_group="sysstat",stat_id="60022",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS))

指标分组	指标名称	指标解释	计算表达式
缓存命中率	block_cache	块缓存命中率	100 * 1 / (1 + sum(rate(sysstat_value{metric_group="sysstat",stat_id="50009",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="50008",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS))
缓存命中率	plan_cache	执行计划缓存命中率	100 * sum(rate(hit_count{metric_group="plan_cache_stat",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(access_count{metric_group="plan_cache_stat",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
缓存命中率	row_cache	行缓存命中率	100 * 1 / (1 + sum(rate(sysstat_value{metric_group="sysstat",stat_id="50001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="50000",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS))
缓存大小	block_cache	块缓存大小	sum(cache_size{metric_group="all_virtual_kvcache_info",cache_name="user_block_cache",@LABELS}) by (@GBLABELS) / 1048576

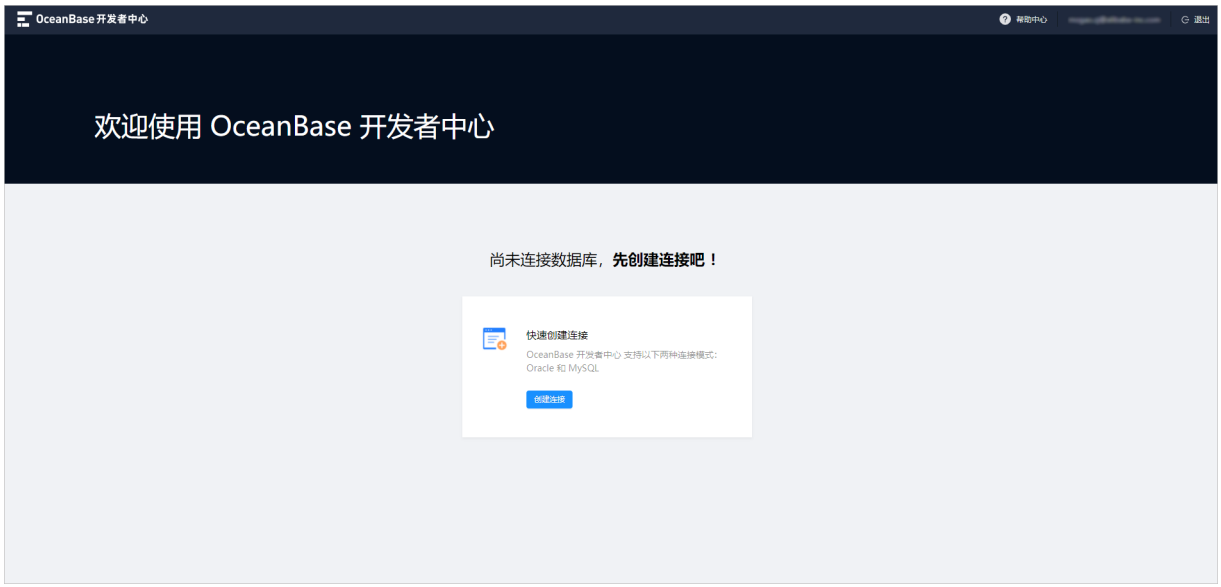
指标分组	指标名称	指标解释	计算表达式
缓存大小	plan_cache	执行计划缓存大小	sum(mem_used{metric_group="plan_cache_stat",@LABELS}) by (@GBLABELS) / 1048576
缓存大小	row_cache	行缓存大小	sum(cache_size{metric_group="all_virtual_kvcache_info",cache_name="user_row_cache",@LABELS}) by (@GBLABELS) / 1048576
网络吞吐率	receive	每秒接收数据量	avg(rate(node_network_receive_bytes_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / 1048576
网络吞吐率	send	每秒发送数据量	avg(rate(node_network_transmit_bytes_total{@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / 1048576
请求等待队列	queue_count	平均每秒 SQL 进等待队列的次数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="20001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

指标分组	指标名称	指标解释	计算表达式
请求等待队列耗时	queue_time	SQL 在等待队列中等待耗时	sum(rate(sysstat_value{metric_group="sysstat",stat_id="20002",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS) / sum(rate(sysstat_value{metric_group="sysstat",stat_id="20001",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
锁等待	fail	写锁等待失败次数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="60022",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)
锁等待	success	写锁等待成功次数	sum(rate(sysstat_value{metric_group="sysstat",stat_id="60021",@LABELS}[@INTERVAL]) by (@GBLABELS)) by (@GBLABELS)

4.OB 开发者中心

4.1. 创建连接

登录 OceanBase 开发者中心（OceanBase Developer Center，ODC）后，如果您尚未创建任何连接配置，页面将如下图所示并提示您创建连接。之后当您再次进入 ODC 时，页面会展示已保存的连接配置列表。



> **注意**
>
> - ODC V2.3.0 版本后，导入导出功能需要在创建连接的高级配置中设置root 用户，否则会提示无法使用导入导出功能，因为是从该版本开始，导入导出功能查询视图信息时依赖 root 账号。
> - 分区信息查看也依赖 root 账号，所以您也需要在创建连接的高级配置中设置root 用户。

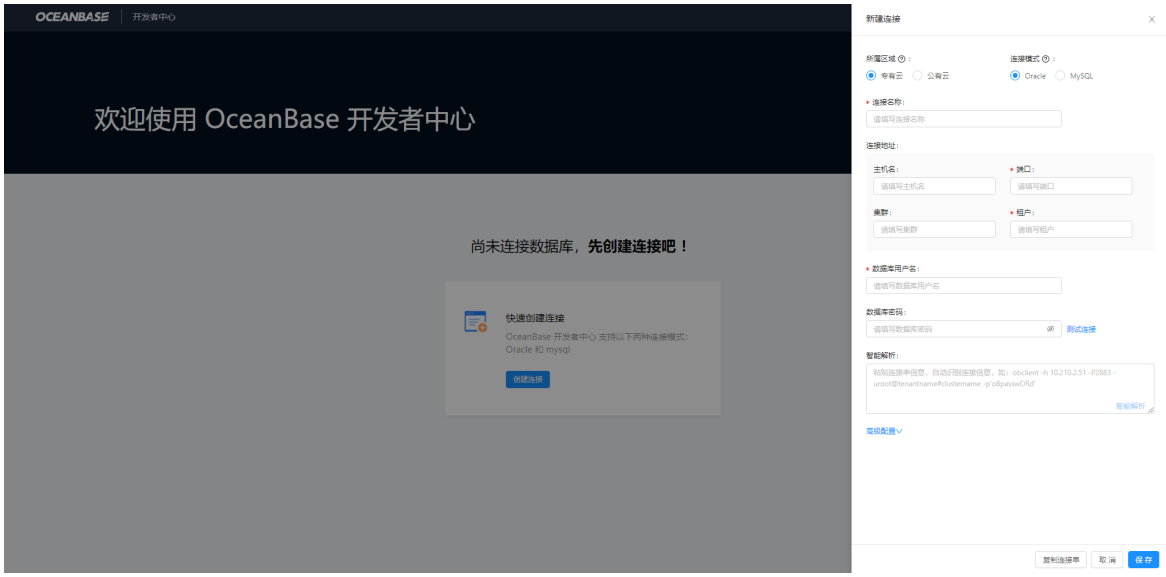
操作步骤：

1. 进入 ODC 后，点击页面中的 **创建连接** 按钮。
2. 在创建连接页面需要填写或选择以下信息，如果您拥有 OceanBase 数据库连接串，那么以下信息均可从连接串中直接获得（mysql -h **主机名** -P **端口号** -u **数据库用户名@租户名#集群名** -D **默认数据库** -p '**数据库密码**'）：

信息项	说明
所属区域	选择连接示例所在的区域 **公有云**（阿里云上的数据库实例） 或 **专有云**（在自己环境中搭建的数据库实例）。
连接模式	选择 **MySQL** 或 **Oracle**。
连接名称	指定该连接的名称。

信息项	说明
连接地址	○ **主机名**：在租户信息页面中获得主机名信息。 ○ **端口号**：在租户信息页面中获得端口号信息。 ○ **集群**：要连接的数据库所在集群的名称。 ○ **租户**：要连接的数据库所在租户的名称。
默认数据库	连接 MySQL 租户中的数据库时，需填写该连接要使用的默认数据库，连接 Oracle 租户中的数据库时则无需填写该项。
数据库用户名	在租户中创建的账号（MySQL 模式下该账户必须有访问默认数据库的权限）。
数据库密码	在租户中创建的账号的密码。 点击数据库密码框后的 **测试连接** 按钮，测试是否能够连接到目标数据库。如果配置信息有误，会有异常提示，例如用户或密码不对、网络异常等。用户只能够连接本租户下的数据库，否则会报出测试失败的错误。
SQL 查询超时时间	单击页面下方 **高级配置** 标签弹出该项。支持自定义 SQL 超时时间，默认 60 s，超过指定的时间时查询自动终止。
查询 sys 租户视图	单击页面下方 **高级配置** 标签弹出该项。 ○ **账户**：拥有 sys 租户视图查看权限的账号。 ○ **密码**：账号对应的密码。 输入账户和密码信息后，单击密码框后的 **测试连接** 按钮测试账户信息是否正确。ODC V2.3.0 后使用数据导入导出和查看分区信息功能时需要在此项配置 root 账号。

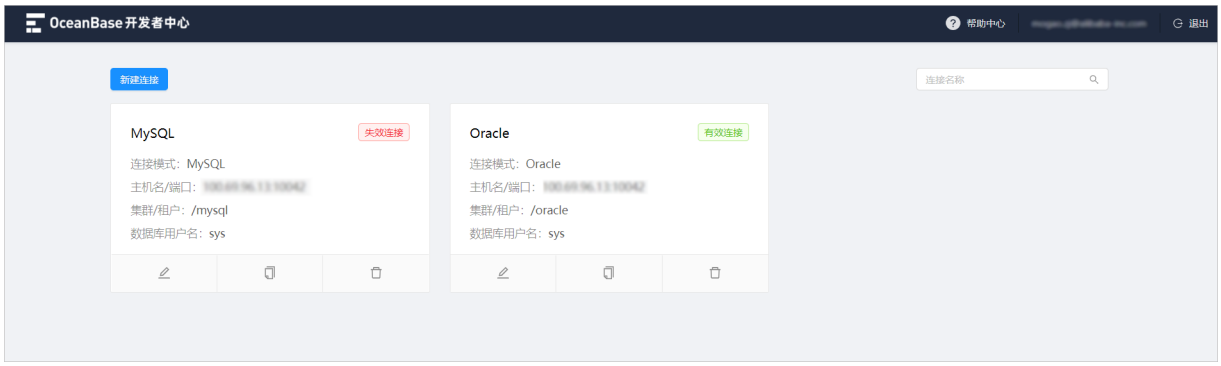
3. 单击页面下方的 **保存** 按钮，即可将当前的配置信息持久保存在 ODC 连接列表中。再次登录 ODC 时您可以在连接配置列表中直接选择该连接，保存连接配置时不要求测试连接一定成功。



4. 在页签下方，ODC 还提供了 **复制连接串** 按钮。单击该按钮，ODC 会将您当前编辑的连接信息拷贝为连接串格式以方便您获得当前配置的连接信息。
5. 当您不需要保存当前的连接配置时，可单击页签下方的 **取消** 按钮，ODC 会弹出弹窗跟您确认是否退出当前新建连接窗口。

4.2. 管理连接

用户创建并保存的连接配置会直接展示在 OceanBase 开发者中心（OceanBase Developer Center，ODC）首页。每个连接配置卡片上会直接展示该连接的名称、连接模式、主机名/端口号、集群/租户和数据库用户名等信息。在卡片的右上角会展示该连接的状态（失效连接或有效连接）以方便您直接获得连接的状态。单击对应的连接配置卡片即可进入目标数据库连接。

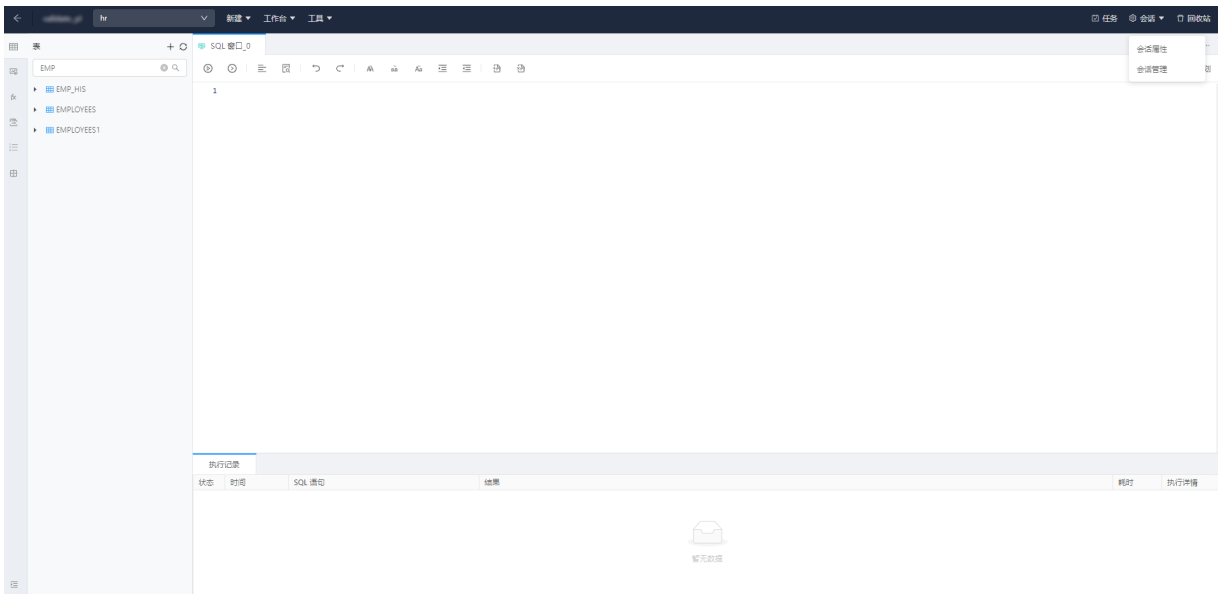


每个连接配置卡片上依次有 **编辑**、**复制** 和 **删除** 等操作按钮：

- **编辑**：对当前保存的连接配置信息进行修改。例如更改数据库账户密码后，需要在连接配置中更新密码并再次保存以防止连接失效。
- **复制**：对当前的配置进行拷贝。这样您在已有的信息基础上，只需做简单的修改即可创建一个新的连接，方便您更便捷的创建信息相似的连接配置。
- **删除**：删除当前的连接配置。例如要连接的数据库不存在了或者后续不需要继续访问该数据库时，可直接删除当前的连接配置。

4.3. 管理会话

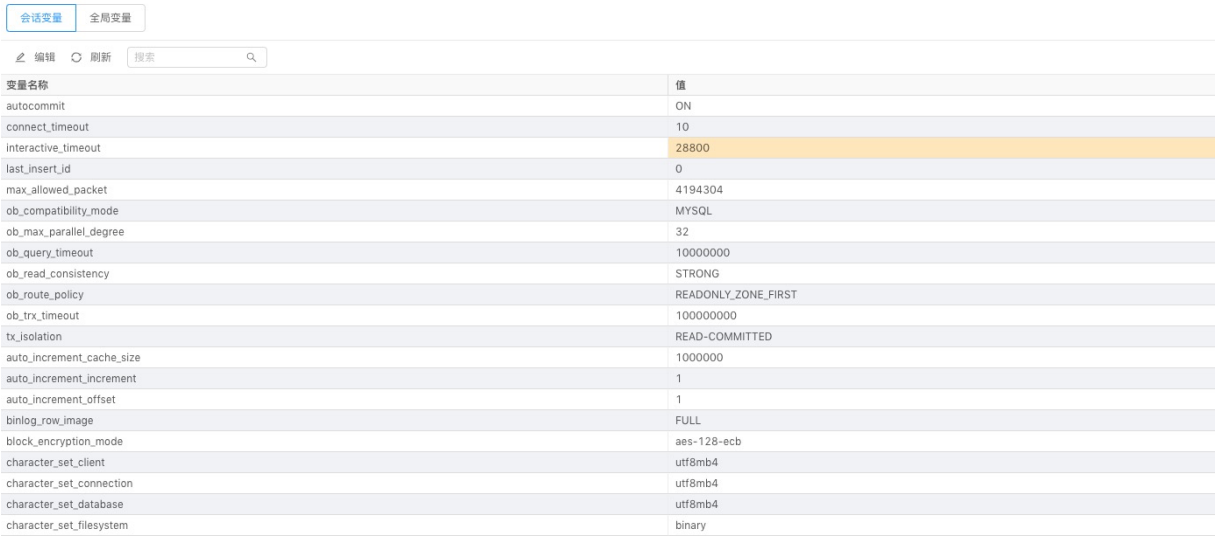
进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击目标连接配置卡片即可进入对应的数据库对象管理页面。点击数据库对象管理页面右上角的 **会话** 按钮，可选择进入 **会话属性** 或 **会话管理** 页面。



会话属性

会话属性即数据库变量，OceanBase 中有会话变量和全局变量：

- **会话变量**：当客户端连接到数据库后，数据库会复制全局变量以自动生成会话变量。会话变量的修改只对当前会话生效。
- **全局变量**：数据库实例共享全局变量。这意味着不同用户共享这些全局变量，且数据库会保存您对全局变量做出的更改，断开连接并再次进入数据库时，更改依旧有效。



变量名称	值
autocommit	ON
connect_timeout	10
interactive_timeout	28800
last_insert_id	0
max_allowed_packet	4194304
ob_compatibility_mode	MYSQL
ob_max_parallel_degree	32
ob_query_timeout	10000000
ob_read_consistency	STRONG
ob_route_policy	READONLY_ZONE_FIRST
ob_trx_timeout	100000000
tx_isolation	READ-COMMITTED
auto_increment_cache_size	1000000
auto_increment_increment	1
auto_increment_offset	1
binlog_row_image	FULL
block_encryption_mode	aes-128-ecb
character_set_client	utf8mb4
character_set_connection	utf8mb4
character_set_database	utf8mb4
character_set_filesystem	binary

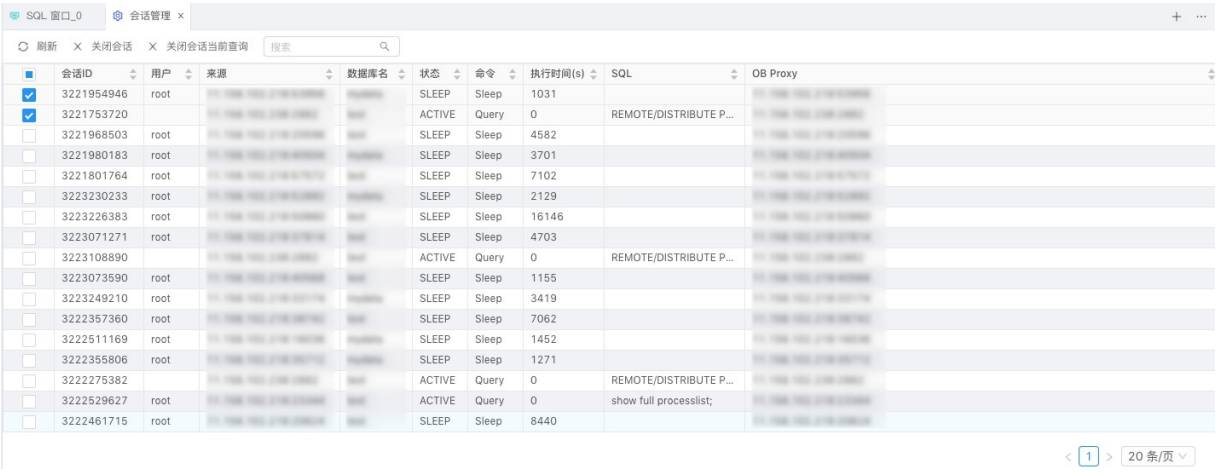
ODC 提供了可视化界面使您可以清晰直观的查看与修改当前数据库支持的变量。对于变量值的修改：

- 如果变量的值是字符或数字类型，可以在变量编辑页面中直接输入修改值。
- 如果变量的值是枚举类型，ODC 会在变量值的编辑框中列举出该变量支持的值的集合，您可以在不用准确记忆变量名和值的情况下通过可视化界面完成修改，这降低了用户记忆变量的成本并提高了变量修改的效率。ODC 会使用橙色标记修改过的数据库变量，以方便您更直观的管理变量。

会话管理

与当前数据库的连接被称为一个会话，在会话管理页面您可以查看连接到当前数据库的所有会话和以下会话信息：

会话信息	说明
会话 ID	展示当前会话的 ID，是会话的唯一标识。
用户	会话登录使用的数据库用户名。
来源	发起会话的地址。
数据库名	当前会话访问的数据库名称。
状态	展示当期会话的状态，有 **SLEEP**（休眠）和 **ACTIVE**（活跃）两种状态。
命令	会话当前执行的命令类型。
执行时间（s）	会话当前执行 SQL 所花的时间。
SQL	会话当前执行 SQL 的具体内容。
OBProxy	会话访问的代理地址。



会话管理页面的导航栏中提供了以下功能键供您管理会话：

- **刷新**：刷新会话列表。
- **关闭会话**：终止当前会话。

- **关闭会话当前查询**：终止当前会话正在进行的 SQL 执行进程。

4.4. 使用工作台

4.4.1. 概述

进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击目标连接配置卡片即可进入对应的数据库管理页面，数据库管理页面由左边的对象导航栏和右边工作台组成。工作台是数据库开发者编辑 SQL 或 PL 语句的工作区。ODC 在工作台提供以下功能：

- SQL 窗口：数据库开发者编辑 SQL 和 PL 语句的工作区。
- 匿名块窗口：数据库开发者编辑匿名块脚本的工作区。
- 已保存的脚本：数据库开发者可以在这里打开之前在 ODC 中保存过的脚本。

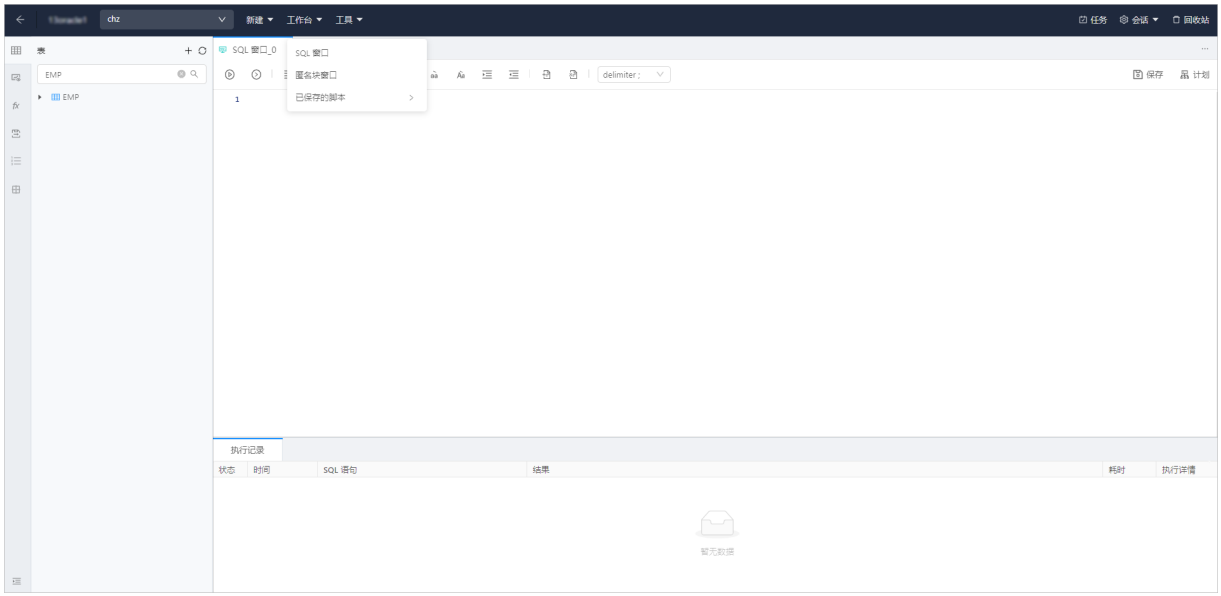
> **注意**
>
> 只有 Oracle 模式下支持创建匿名块窗口。

本章节将分篇介绍工作台提供的功能：

- SQL 窗口
- 匿名块窗口
- 已保存的脚本

4.4.2. SQL 窗口

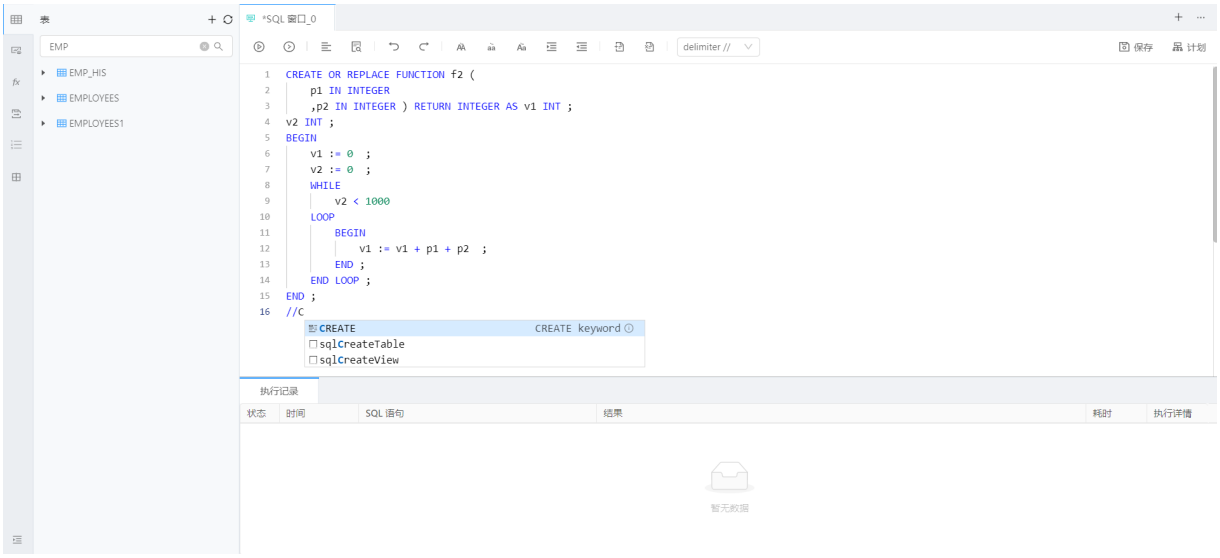
进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）的数据库管理页面后，点击上方导航栏中的 **工作台** 标签，在弹出下拉菜单中您可以选择单击 **SQL 窗口** 来新建 SQL 窗口。SQL 窗口中提供了对 SQL 及 PL 脚本进行编辑的区域和 **执行记录** 与 **结果** 两个页签。



SQL 编辑区域

在 SQL 窗口的编辑区域您可以体验到较多的细节，如窗口中会对数据库的关键字进行不同颜色的标识，其次还有输入自动提示、格式化等功能以提升您编辑 SQL 语句的效率。

> **说明**
>
> SQL 窗口中也支持编辑 PL 语句，首先需要在工具栏中选择 delimiter 以完成界定符的设置，然后再开始编写 PL 语句，PL 语句编写完毕后使用选择的界定符作为结尾。



除了编辑 SQL 语句时可以体验到上述细节，编辑区的工具栏中还提供了以下功能键：

功能	说明
运行	点击该按钮运行当前 SQL 窗口中所有的 SQL 语句。
运行当前语句	击该按钮运行当前您选中的所有SQL语句或当前光标所在行的 SQL 语句。
格式化	点击该按钮对选中的 SQL 语句或当前 SQL 窗口中的所有 SQL 语句进行统一的格式化（缩进、换行和高亮关键字等操作）。
计划	查看选中的或当前光标所在的 SQL 语句在执行前系统预估的执行计划（ `EXPLAINPLAN` 操作的结果），展示的执行数据可能与实际执行语句后的数据略有不用，您可以使用该功能评估 SQL 语句。
查找/替换	在查找框内输入内容以对脚本进行检索，查找后可在替换框内输入内容以对查找内容进行替换。

功能	说明
撤销	将脚本回退到上一步操作执行的结果。
重做	执行 **撤销** 操作后，对脚本重新执行撤销前的操作。
大写	将您在脚本中选中的语句全部转换为大写形式。
小写	将您在脚本中选中的语句全部转换为小写形式。
首字母大写	将您在脚本中选中的语句全部转换为首字母大写形式。
添加缩进	对您在脚本中选中的语句进行缩进操作。
删除缩进	删除您在脚本中选中的语句中所包含的缩进。
添加注释	将您在脚本中选中的语句转换为注释。
删除注释	将您在脚本中选中的注释转换为SQL语句。
保存	点击该按钮保存当前窗口中的脚本。之后您再次进入工作台可以直接打开保存好的脚本。每个用户只能看到自己保存的 SQL 脚本，且脚本名不允许重复。打开已保存的脚本后，您可以继续对该脚本进行编辑。

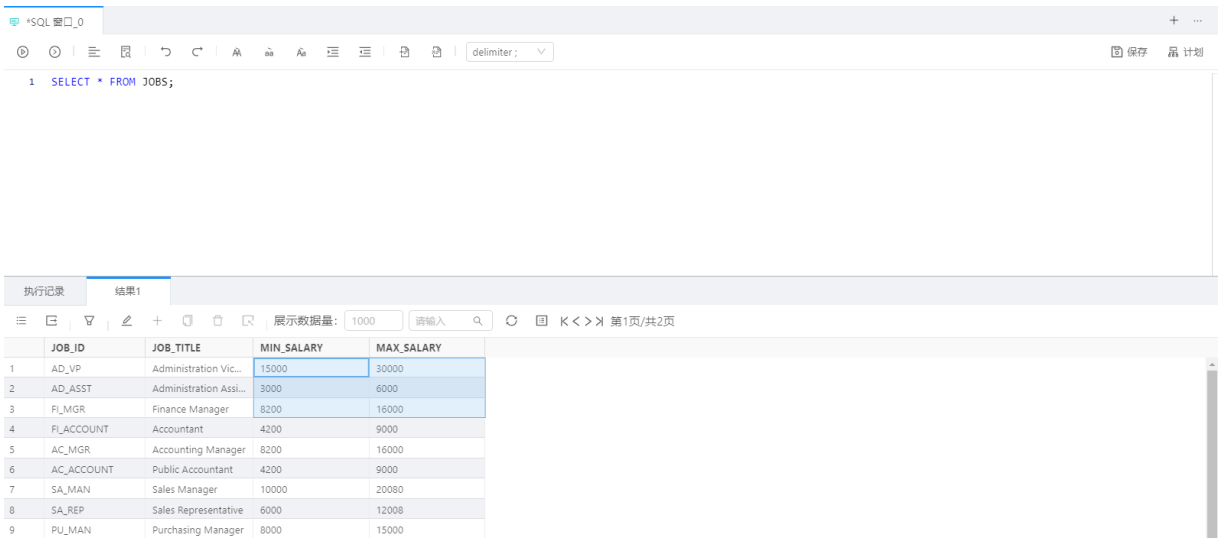
执行记录页签

在执行记录页签中您可以查看当前连接进程中每次执行 SQL 语句的记录，包括 **状态**、**时间**、**SQL 语句**、**结果描述**、**耗时** 和 **执行详情**（即执行计划）等信息。

结果页签

您可以在该页签查看当前 SQL 语句的执行结果。返回的结果集支持通过鼠标选中目标范围后直接拷贝至外部文件，或者您也可以通过工具栏中的 **导出数据** 工具将结果集中的数据导出到本地文件。当单元格中的数据过长导致无法全部显示时，您可以将鼠标箭头放置在目标单元格上来预览单元格中的全部数据。

右键结果页签的标签名，在弹出的列表中单击 **固定** 可让当前结果页签保持展示状态，当执行一条新的查询时当前结果不会被覆盖而是生成一个新的结果展示页签，这样您可以方便地对比不同的查询结果。单击 **解除固定** 以取消目标结果页签的固定状态。



结果页签的导航栏中还提供了以下工具：

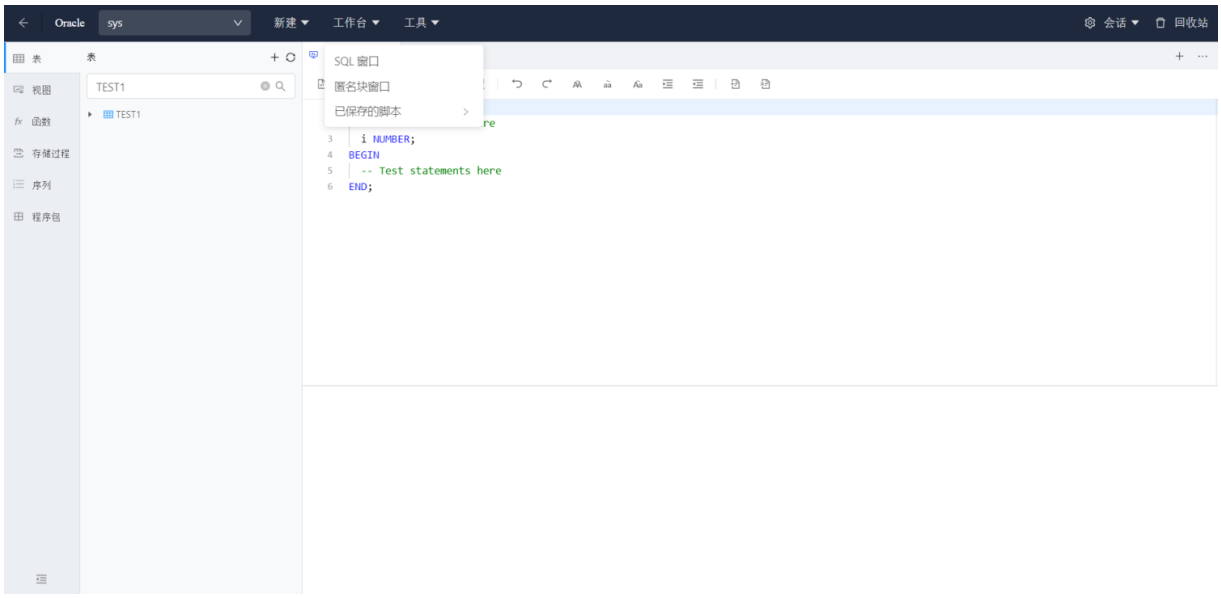
功能	说明
列模式	以表格的形式展示一行选中的数据。在 **列模式** 页面可以进行上一行、下一行的切换展示，对于列数量比较多场景，**列模式** 比较方便行数据的查看。
导出数据	将查询结果导出为 CSV 或 SQL 文件，您需要选择导出的 **文件类型**、**字符集**（gbk 或 utf8）和 **行数限制** 等信息。
列管理	筛选在页面中展示的列。
编辑	单击 **编辑** 按钮开启对当前展示的结果集的编辑操作，**编辑** 按钮后提供了 **添加一行**、**复制当前行**、**删除** 和 **提交** 等操作按钮。您可以双击数据对其直接进行修改，或是通过上述操作按钮进行快捷操作。编辑结束后点击 **提交** 按钮会在弹窗中展示对应操作的 SQL 语句，点击弹窗右下角的 **执行** 按钮对结果集执行编辑操作，不点击 **执行** 则编辑无效。再次单击 **编辑** 按钮以关闭编辑模式。
展示数据量	可指定结果集展示的行数和在结果集中搜索指定值。
刷新	在编辑和指定展示数据量后可以刷新结果集以展示最新的数据。

功能	说明
执行计划	查看当前 SQL 语句执行后的实际资源消耗及执行计划，可快速判断 SQL 语句的优劣。

4.4.3. 匿名块窗口

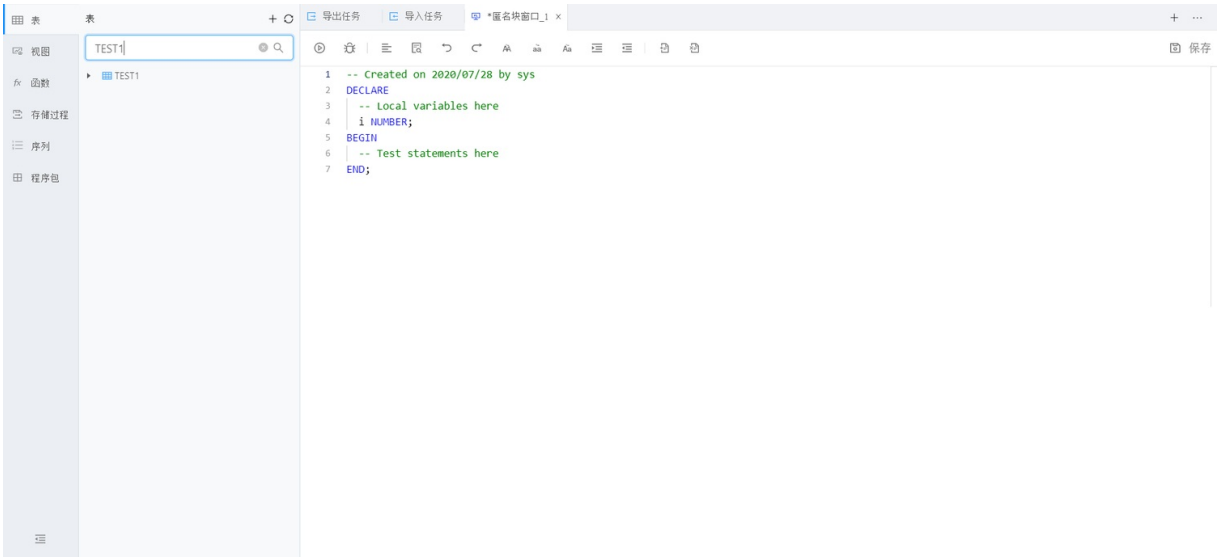
进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）的数据库管理页面后，点击上方导航栏中的 **工作台** 标签，在弹出下拉菜单中您可以选择点击 **匿名块窗口** 来新建窗口。

匿名块窗口中提供了 **PL 编辑区域**、**PL 运行结果和 DBMS 输出页签** 和 **调试页面**。



PL 编辑区域

在匿名块窗口的编辑区域您可以体验到较多的细节，如当您新建一个匿名块窗口时会自动生成引导代码方便您编写脚本，同时窗口中会对数据库的关键字进行不同颜色的标识，其次还有格式化等功能以提升您编辑 PL 语句的效率。



除了编辑 PL 语句时可以体验到上述细节以外，编辑区域的工具栏中还提供了以下功能键：

功能	说明
运行	单击该按钮运行代码区中的语句。
调试	单击该按钮进入调试页面，并对代码区内的匿名块对象进行调试。
保存	点击该按钮保存当前窗口中的脚本。之后您再次进入工作台可以直接打开保存好的脚本。每个用户只能看到自己保存的脚本，且脚本名不允许重复。打开已保存的脚本后，您可以继续对该脚本进行编辑。
格式化	点击该按钮对选中的 SQL 语句或当前 SQL 窗口中的所有 SQL 语句进行统一的格式化（缩进、换行和高亮关键字等操作）。
查找/替换	在查找框内输入内容以对脚本进行检索，查找后可在替换框内输入内容以对查找内容进行替换。
撤销	将脚本回退到上一步操作执行的结果。
重做	执行撤销操作后，对脚本重新执行撤销前的操作。
大写	将您在脚本中选中的语句全部转换为大写形式。

功能	说明
小写	将您在脚本中选中的语句全部转换为小写形式。
首字母大写	将您在脚本中选中的语句全部转换为首字母大写形式。
添加缩进	对您在脚本中选中的语句进行缩进操作。
删除缩进	删除您在脚本中选中的语句中所包含的缩进。
添加注释	对脚本中选中的语句添加注释。
删除注释	对脚本中选中的语句删除注释。

PL 运行结果和 DBMS 输出页签

在匿名块编辑区域的工具栏单击 **运行** 后，会运行编辑区域中的匿名块脚本，在编辑区域下方会展示 **运行结果** 页签和 **DBMS 输出** 页签：

- **运行结果**：展示脚本运行的结果，若是运行无误则显示 **运行成功** 的字样提示，若是运行有误则展示错误提示信息。
- **DBMS 输出**：若子程序中包含 PL 输出语句则展示其调用后的输出结果（例如 `dbms_output.put_line` ）。

调试页面

在匿名块的编辑区域单击 **调试** 按钮后可进入调试页面。进入调试页面后您可以对当前窗口中的匿名块进行调试。

在调试模式下，编辑区的工具栏中提供了以下功能键：

功能	说明
自动调试	批量运行至最近断点，若无断点直接运行至结尾语句
跳入	单步执行，若当前行是调用已定义的存储程序或函数，则会进入被调用的子程序。
跳出	在子程序内，执行跳出操作后，返回到上层调用位置的下一行；主程序内运行跳出，效果和 **自动调试** 相同。
跳过	单步执行，不会进入子程序。

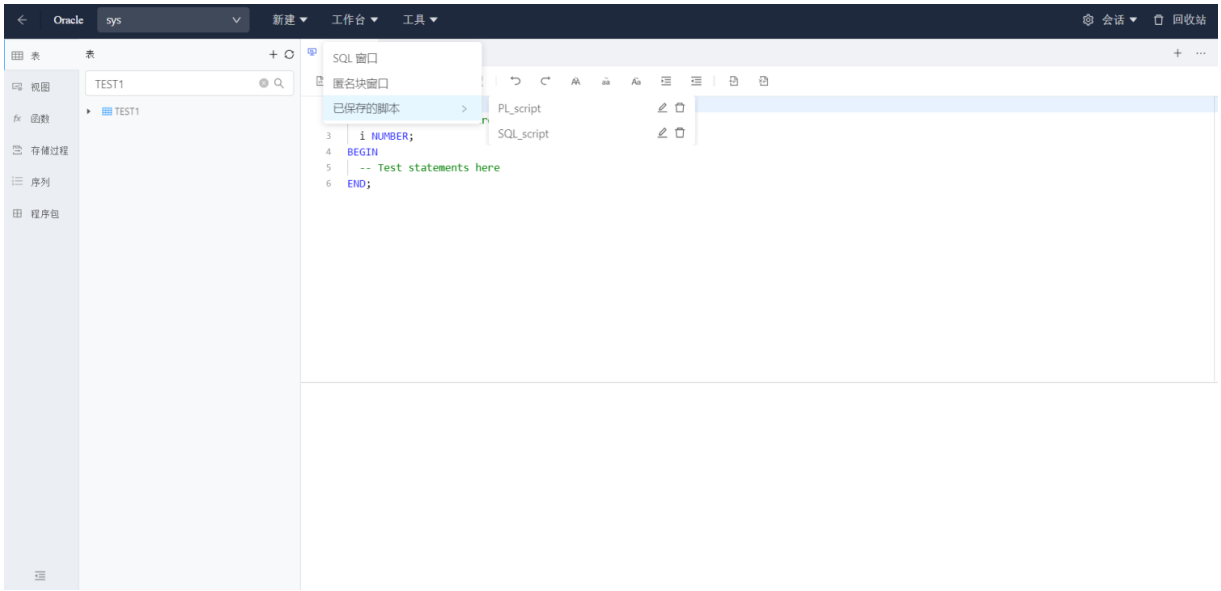
功能	说明
终止调试	继续运行直到结束，会跳过设置的断点。
重新调试	对当前调试对象发起新一轮的调试，与 **调试** 的区别是不用新建调试连接。
退出调试	关闭调试连接，并退出调试交互窗口。

在调试模式下除了编辑区还提供了以下页签：

- **参数**：参数页签展示该调试编辑所有的参数定义信息（参数名、模式和类型）及对应的值，其中对应的值会根据调试过程作出相应的变化。
- **堆栈和变量**：展示当前栈上的变量与其对应的值。
- **DBMS 输出**：若子程序中包含 PL 输出语句则展示其调用后的输出结果（例如 `dbms_output.put_line` ）。
- **断点**：在编辑区，单击行号设置断点（再次单击取消断点）。在断点页签可以查看到您设置的所有断点，列表默认按照断点执行顺序进行排序，列表支持批量选择。每一行断点信息的操作栏中提供了 **取消**（取消该断点）和 **查看**（在编辑区光标会跳转至该断点所处的位置）两个操作。
- **调试日志**：调试运行结束后展示错误信息（如果调试存在错误）。

4.4.4. 已保存的脚本

进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）的数据库管理页面后，点击上方导航栏中的 **工作台** 标签，在弹出的下拉菜单中您可以选择点击 **已保存的脚本** 以进入您之前在 SQL 窗口和匿名块窗口中保存的脚本。



在下拉菜单展示的脚本列表中单击脚本名以继续编辑或运行该脚本，若您的脚本是 SQL 脚本则自动生成 SQL 窗口以展示脚本，若是匿名块脚本则自动生成匿名块窗口以展示脚本。在脚本列表中，脚本名后还提供了以下两个功能键：

- **编辑**：单击编辑按钮，在弹出的页面中可以修改脚本的名称和脚本内容。单击页面右下角的 **确认** 按钮以使修改生效。
- **删除**：单击删除按钮以删除该脚本。

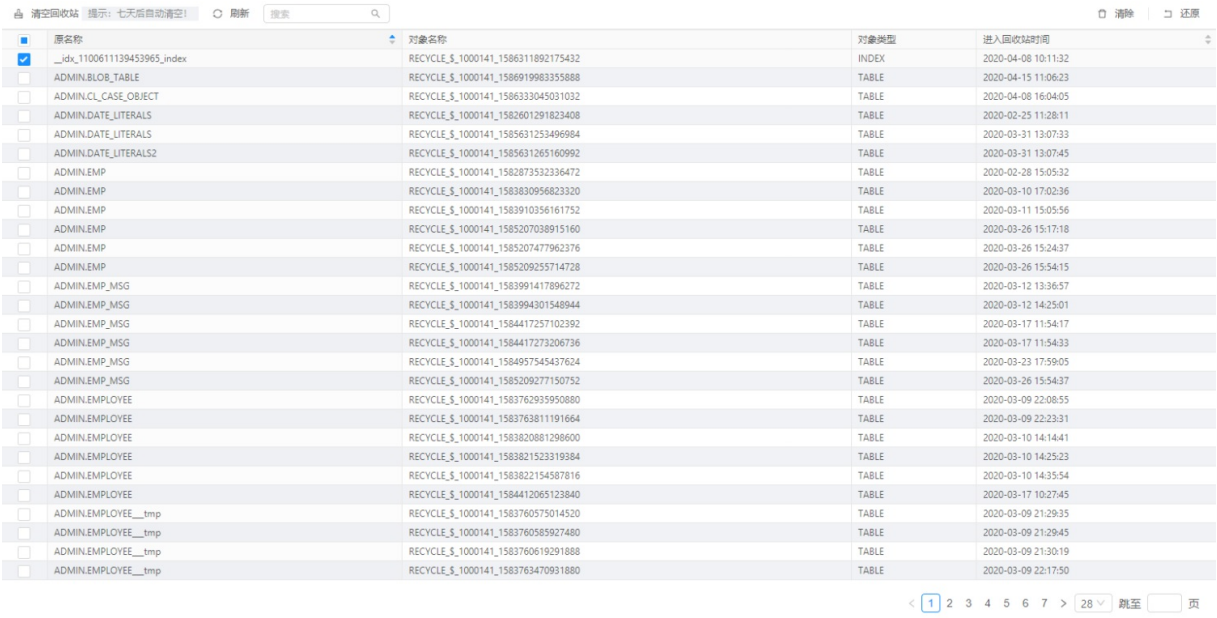
4.5. 使用回收站

进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击目标连接配置卡片进入对应的数据库对象管理页面后，点击右上角的 **回收站** 标签即可进入回收站页面。

回收站的功能是短暂保存被删除掉的数据库对象，它可以帮助您轻松查询和还原被删除的数据库对象。原理上，回收站是一个数据字典表，被用来放置用户删除掉的数据库对象信息。这里的删除指的是操作 `DROP`，被 `DELETE` 操作删除的数据不会进入回收站。所以被 `DROP` 操作删除的对象并没有被数据库彻底删除，它们仍然会占用空间，您可以进行 `PURGE` 操作或在 ODC 中清空回收站以彻底释放空间。

如下图所示，回收站页面展示以下信息：

信息	说明
原名称	展示被删除的对象的名称（用户创建对象时自定义的名称）。
对象名称	系统中为该对象命名的名称。与原名称不同，对象名称由系统指定不会出现重复名称，可以作为对象的唯一标识。
对象类型	展示被删除的对象的类型，对象类型有 **TABLE（表）**、**VIEW（视图）**、**FUNCTION（函数）**、**PROCEDURE（存储程序）** 和 **SEQUENCE（序列）** 5种。
送入回收站时间	展示该对象被删除的时间戳，方便您更具体的回溯被删除的对象。



回收站页面的导航栏中还提供了以下功能键：

- **清空回收站**：清空回收站中当前所有的被删除对象并释放空间。
- **刷新**：刷新回收站中被删除对象的列表。
- **清除**：将当前选中的对象从回收站中立即删除并释放存储空间。
- **还原**：将当前选中的对象从回收站中恢复到原来的位置。

4.6. 数据导出和导入

4.6.1. 概述

为了让数据库开发者更方便的维护数据库内的数据，OceanBase 开发者中心（OceanBase Developer Center，ODC）提供整库导入导出和单表导出功能。通过整库导入导出用户可以批量选择数据库中的表以导入或导出其中的数据，通过单表导出，用户可以导出目标表中的数据，目前 ODC 支持导入导出 SQL 文本和 ODC 格式的文件。

创建导入或导出任务后，ODC 提供导入导出任务页签功能。页签中会显示当前的导入或导出任务列表，您还可以查看目标任务的的任务详情和日志等信息。

> 注意
>
> - OceanBase V2.2.30 及之后版本支持字符编码 UTF8、GBK、GB18030 和 UTF16。OceanBase 2.2.30 版本之前仅支持字符编码 UTF8。
> - 在 ODC V2.2.1 及之前的低版本中使用导入导出功能时必须通过 OBProxy 连接至目标实例，直连时 ODC 不支持导入导出功能；
> - ODC V2.3.0 版本后，导入导出功能需要在创建建连接的高级配置中设置 root 用户，否则会提示无法使用导入导出功能，因为是从该版本开始，导入导出功能查询视图信息时依赖 root 账号。

本章节将分篇介绍以下导入导出的相关功能：

- 导出导入格式
- 批量导出与导入

- 单表导出与导入
- 任务管理

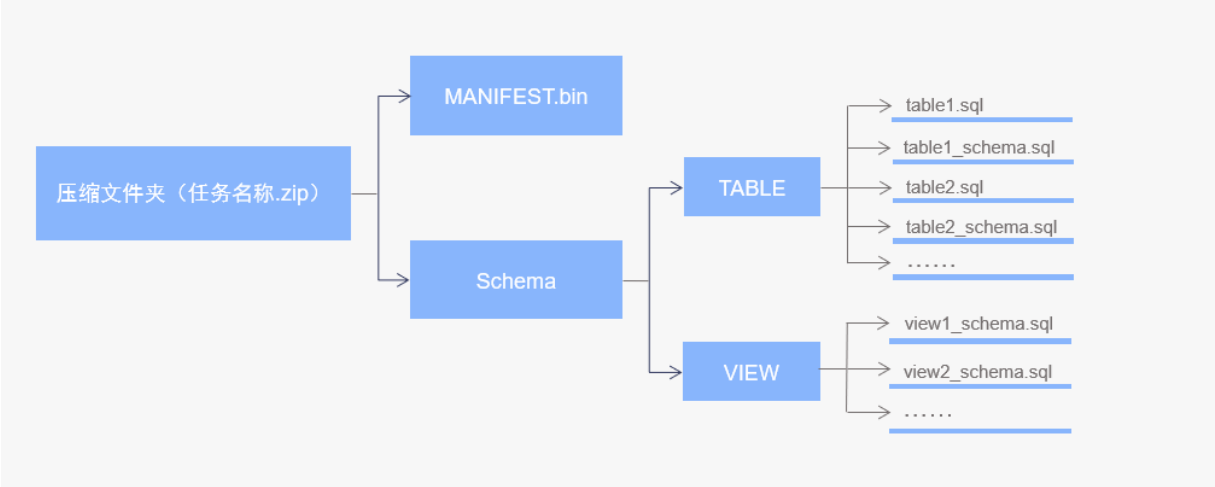
4.6.2. 导出导入格式

OceanBase 开发者中心（OceanBase Developer Center，ODC）数据导出与导入支持导出的内容有导出结构和数据、仅导出数据和仅导出结构等三种模式。其中结构是指被导出对象的结构定义（DDL）文件，数据是对象中实际保存的数据文件。根据您选择的导出导入格式不同，其中结构和数据文件的类型也不同。

目前 ODC 支持的导出格式有 ODC 格式和 SQL 格式，支持的导入格式有 ODC 格式、SQL 格式、SQL 文件（批量导入）和 CSV 文件（单表导入）。其中 SQL 格式和 ODC 格式是 OceanBase 导入导出工具特有的格式。

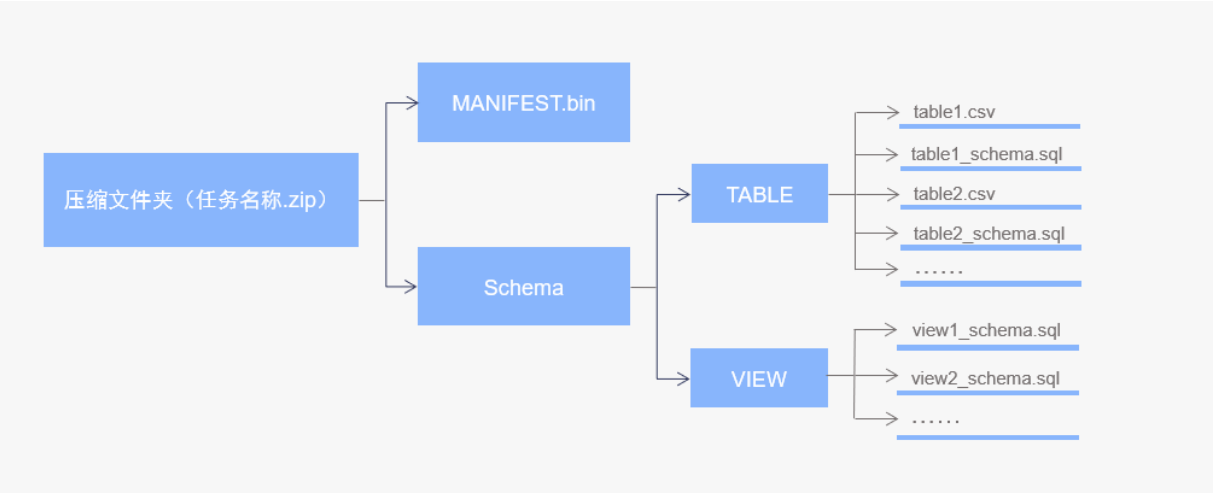
SQL 格式

SQL 格式的文件是以 .zip 结尾的压缩文件，其中包含元数据文件 MANIFEST.bin 和 Schema 下导入导出对象的结构定义文件和数据文件，结构定义文件是以 `对象名_schema.sql` 命名的 SQL 文本，数据文件是以 `对象名.sql` 命名的 SQL 文本，其中对象名与数据库实际对象名保持一致。



ODC 格式

ODC 格式的文件也是以 .zip 结尾的压缩文件，其中包含元数据文件 MANIFEST.bin 和 Schema 下导出对象的结构定义文件和数据文件，结构定义文件是以 `对象名_schema.sql` 命名的 SQL 文本，数据文件是以 `对象名.csv` 命名的 CSV 文本，其中对象名与数据库实际对象名保持一致。



SQL 文件

数据导入支持的 SQL 文件是以 .sql 结尾包含可直接在数据库中执行的 `SELECT` 、 `INSERT` 、 `UPDATE` 、 `DELETE` 、 `CREATE` 和 `DROP` 语句的文件。

CSV 文件

数据导入支持的 CSV 文件是 .csv 或 .txt 结尾的支持自定义分割符的数据文件。

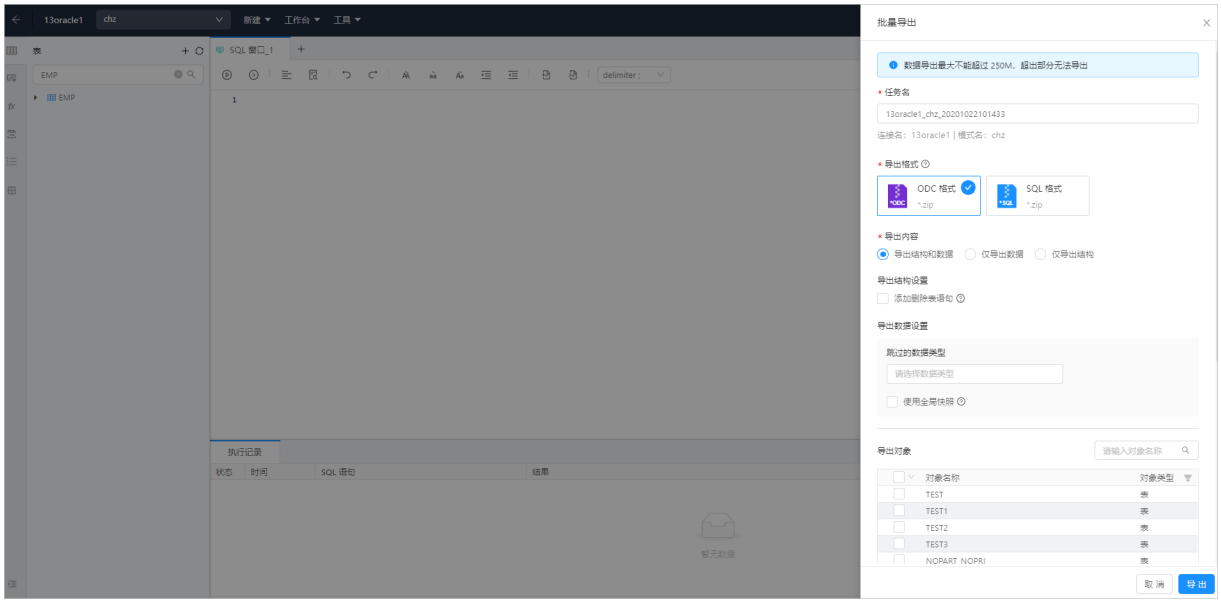
4.6.3. 批量导出与导入

进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击目标连接配置卡片进入对应的数据库对象管理页面，点击上方导航栏中的 **工具** 标签，在弹出的下拉菜单中您可以单击 **导出** 或 **导入** 标签进入整库导出与导入页面。

批量导出

在导航栏中单击 **工具** 标签，在弹出的下拉菜单中单击 **导出** 以弹出 **批量导出** 页签。在 **批量导出** 页签，您可以选择将指定表的数据导出为 SQL 格式文件或是 ODC 格式文件。

> 注意
>
> - ODC 对导出的数据大小有限制，最大支持导出 250 M 的数据（压缩前），超出部分无法导出。
> - 导出文件的编码为 UTF-8，暂不支持修改。



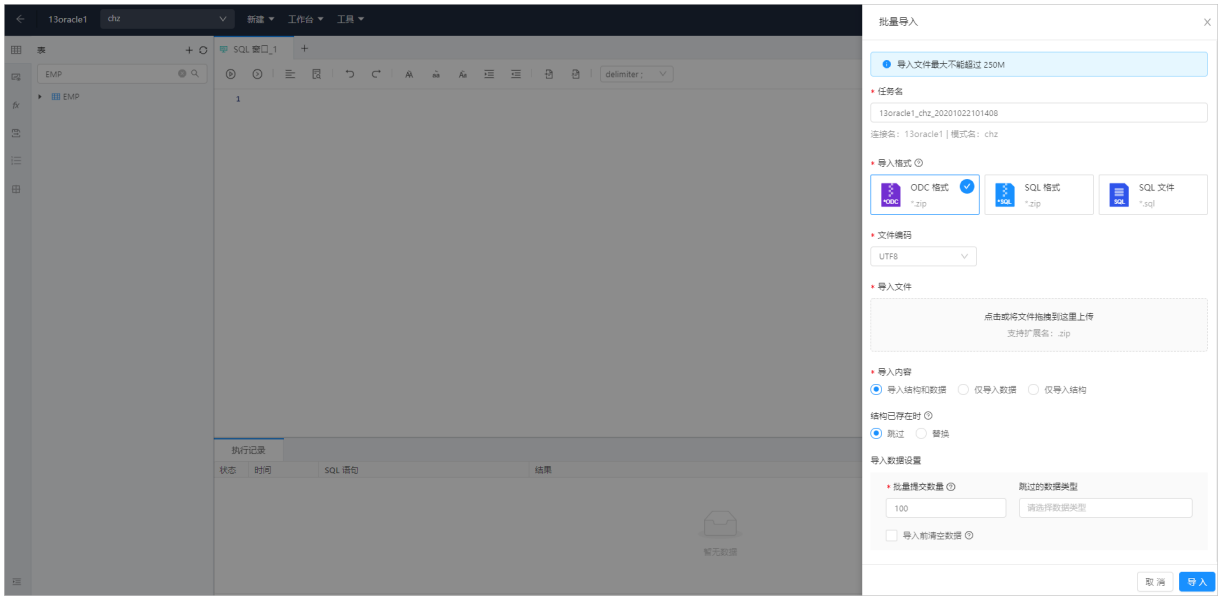
操作步骤：

1. 进入批量导出页签后，ODC 会自动生成默认的任务名（连接名_模式名_日期），您也可以对任务名进行修改以自定义导出任务名称。
2. 选择导出格式：ODC 目前支持将数据导出为 **ODC 格式** 和 **SQL 格式** 文件。
3. 选择导出内容：ODC 目前支持 **导出结构和内容**、**仅导出数据** 和 **仅导出结构** 三种方式。其中导出结构指是否要导出目标对象的定义语句。
4. 设置导出结构：在 **导出结构设置** 中勾选是否要 **添加删除表语句**。勾选后在导出的结构文件中每条 `CREATE` 语句前会添加对应的 `DROP` 语句，勾选后当您导入该结构文件时会删除数据库中原有的对象以实现数据的覆盖。
5. 导出数据设置：
   - 跳过的数据类型：指定导出数据时要跳过数据类型。MySQL 和 Oracle 下可跳过的类型不同，支持多选。
   - 使用全局快照：勾选该项后ODC 将导出全局中指定表的最新快照点中的数据，以保证全局数据的统一性。
   - 批量提交数量：选择导出数据为 SQL 格式文件时，可指定批量提交数量，即在导出文件中导出指定数量数据时添加一句 `COMMIT` 命令。
6. 选择导出对象：在批量导出页签的 **导出列表** 中勾选您想要导出的对象。选择好所有要导出的对象后，单击右下角的 **导出** 按钮以生成导出文件。
7. 在选择好所有要导出的对象后，单击右下角的 **导出** 按钮以生成导出文件。
8. 在 任务列表 中您可以将生成的文件保存到本地并在本地中查看导出的数据文件。

批量导入

在导航栏中单击 **工具** 标签，在弹出的下拉菜单中单击 **导入** 以进入 **批量导入** 页面。在 **批量导入** 页面，您可以通过导入 ODC 格式文件、SQL 格式文件或 SQL 文件向表中插入数据。

> **注意**
>
> ODC 对导入上传的文件大小有限制，最大支持导入 250 M（压缩后）的文件，超出部分无法上传。

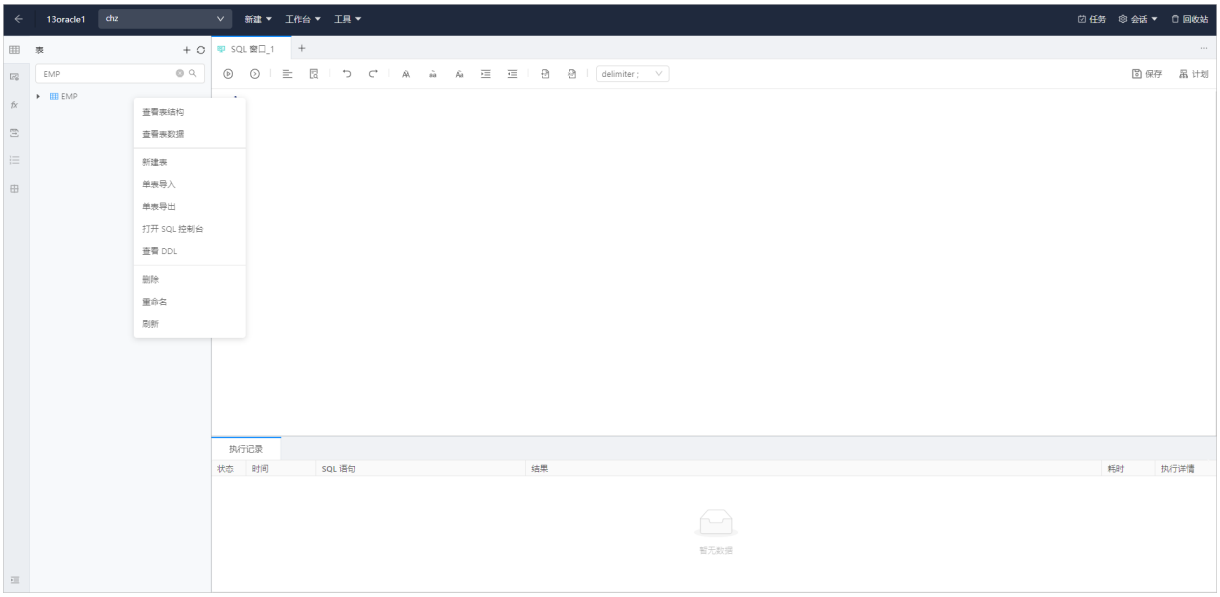


操作步骤：

1. 进入批量导入页签后，ODC 会自动生成默认的任务名（连接名_模式名_日期），您也可以对任务名进行修改以自定义导入任务名称。
2. 选择导入格式：ODC 目前支持导入 **ODC 格式**、**SQL 格式** 和 **SQL 文件** 等文件类型。
3. 指定文件编码：选择导出数据的文件编码格式，默认为 UTF-8 格式，支持下拉选择。
4. 上传导入文件：单击文件池进入文件资源管理器选择要导入的文件或者直接将导入文件拖入文件池中以完成文件上传。上传的文件格式需要与您选择的导入文件格式相同，ODC 格式和 SQL 格式支持上传 .zip 结尾的文件。SQL 文件支持上传 .sql 文件。具体格式信息，请参见 **导出导入格式**。
5. 选择导入内容：ODC 目前支持 **导入结构和数据**、**仅导入数据** 和 **仅导入结构** 三种方式。
6. 选择结构已存在时的操作：选择当导出的对象已经存在时是否要跳过该对象还是替换该对象。跳过时则保留原对象的结构，替换时删除原对象并根据导入内容重建。
7. 导入数据设置：
   - 批量提交数量：指定每导入多少行数据时，数据库自动提交一次。
   - 跳过的数据类型：当导入 ODC 格式的文件时可以指定导入时要跳过的数据类型，MySQL 和 Oracle 下可跳过的类型不同，支持多选。
   - 导入前清空数据：当导入的对象已存在时是否清空该对象之前的数据。当在步骤 6 选择替换已存在的结构时，该项不可用且默认被勾选。
8. 指定好上述信息后，单击右下角的 **导入** 按钮以开始导入数据。

4.6.4. 单表导出与导入

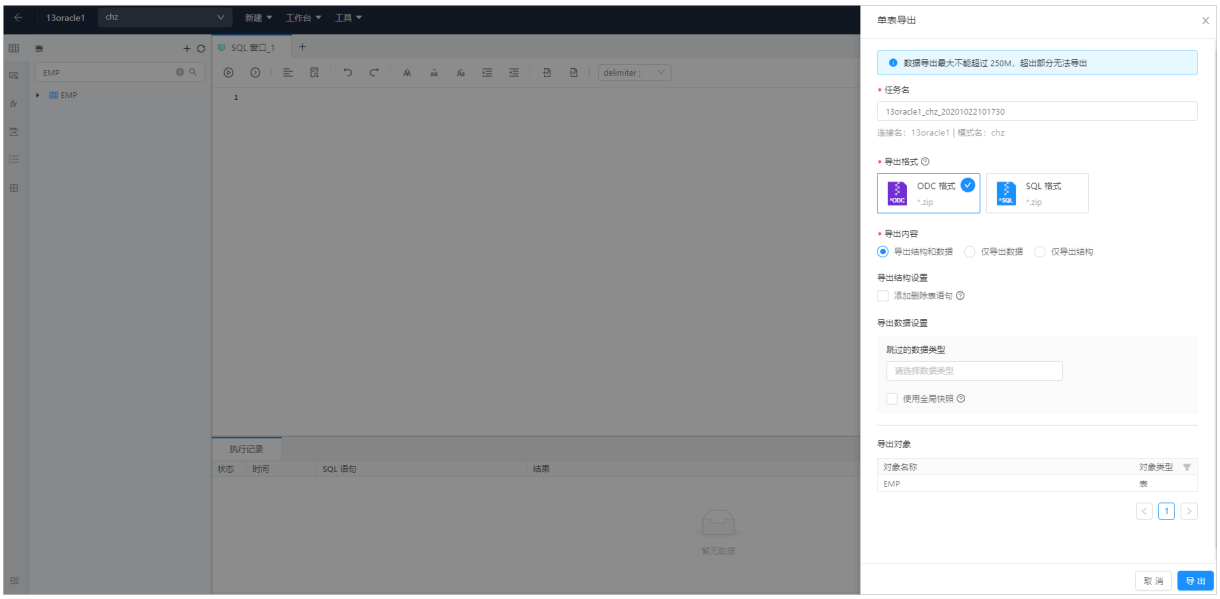
进入 OceanBase 开发者中心（OceanBase Developer Center，ODC）后在数据库对象管理页面的左导航栏中单击 表 标签以查看表列表，在表列表中右键表名，在弹出的列表中您可以选择使用 **单表导出** 或 **单表导入** 功能以导出或导入当前选中的表中的数据。



单表导出

单表导出的操作页面如下所示，与 **批量导出** 的操作基本一致：

> **注意**
>
> - ODC 对导出的数据大小有限制，最大支持导出 250 M 的数据（压缩前），超出部分无法导出。
> - 导出文件的编码为 UTF-8，暂不支持修改。



操作步骤：

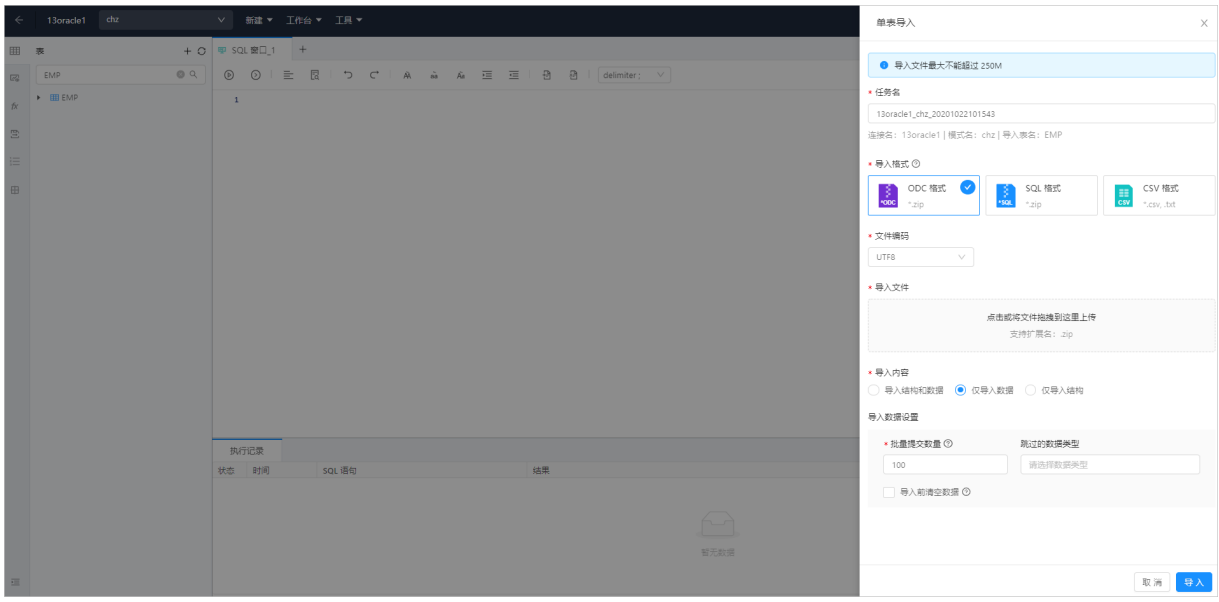
1. 进入单表导出页签后，ODC 会自动生成默认的任务名（连接名_模式名_日期），您也可以对任务名进行修改以自定义导出任务名称。
2. 选择导出格式：ODC 目前支持将数据导出为 **ODC 格式** 和 **SQL 格式** 文件。
3. 选择导出内容：ODC 目前支持 **导出结构和内容**、**仅导出数据** 和 **仅导出结构** 三种方式。其中导出结构指是否要导出目标对象的定义语句。

4. 设置导出结构：在 **导出结构设置** 中勾选是否要 **添加删除表语句。**勾选后在导出的结构文件中每条 `CREATE` 语句前会添加对应的 `DROP` 语句，勾选后当您导入该结构文件时会删除数据库中原有的对象以实现数据的覆盖。
5. 导出数据设置：
   - 跳过的数据类型：指定导出数据时要跳过数据类型，MySQL 和 Oracle 下可跳过的类型不同，支持多选。
   - 使用全局快照：勾选该项后 ODC 将导出全局中指定表的最新快照点中的数据，以保证全局数据的统一性。
   - 批量提交数量：选择导出数据为 SQL 格式文件时，可指定批量提交数量，即在导出文件中导出指定数量数据时添加一句 `COMMIT` 命令。
6. 单击右下角的 **导出** 按钮以生成导出文件。
7. 在 **任务列表** 中您可以将生成的文件下载到本地并查看导出的数据文件。

单表导入

单表导出的操作页面如下所示，与 **批量导入** 的操作基本一致：

> **注意**
>
> ODC 对导入上传的文件大小有限制，最大支持导入 250 M（压缩后）的文件，超出部分无法上传。



操作步骤：

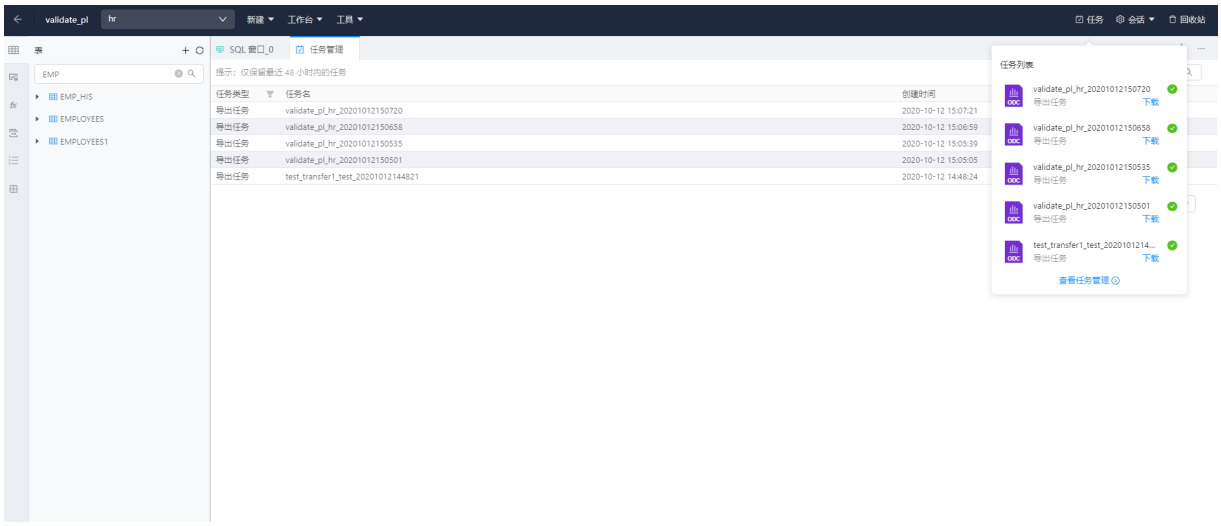
1. 进入批量导入页签后，ODC 会自动生成默认的任务名（连接名_模式名_日期），您也可以对任务名进行修改以自定义导入任务名称。
2. 选择导入格式：ODC 目前支持导入 **ODC 格式**、**SQL 格式** 和 **CSV文件** 等文件类型。
3. 指定文件编码：选择导出数据的文件编码格式，默认为 UTF-8 格式，支持下拉选择。
4. 上传导入文件：单击文件池进入文件资源管理器选择要导入的文件或者直接将导入文件拖入文件池中以完成文件上传。上传的文件格式需要与您选择的导入文件格式相同，ODC 格式和 SQL 格式支持上传 .zip 结尾的文件。CSV 文件支持上传 .csv 或 .text 文件。具体格式信息，请参见 **导出导入格式**。

5. 选择导入内容：ODC 目前支持 **导入结构和数据**、**仅导入数据** 和 **仅导入结构** 三种方式。
6. 选择结构已存在时的操作：选择当导出的对象已经存在时是否要跳过该对象还是替换该对象。跳过时则保留原对象的结构，替换时删除原对象并根据导入内容重建。
7. 导入数据设置：
   - 批量提交数量：指定每导入多少行数据时，数据库自动提交一次。
   - 跳过的数据类型：当导入 ODC 格式文件和 CSV 文件时可以指定导入时要跳过的数据类型，MySQL 和 Oracle 下可跳过的类型不同，支持多选。
   - 导入前清空数据：当导入的对象已存在时是否清空该对象之前的数据。当在步骤 6 选择替换已存在的结构时，该项不可用且默认被勾选。
8. 指定好上述信息后，单击右下角的 **导入** 按钮以开始导入数据。

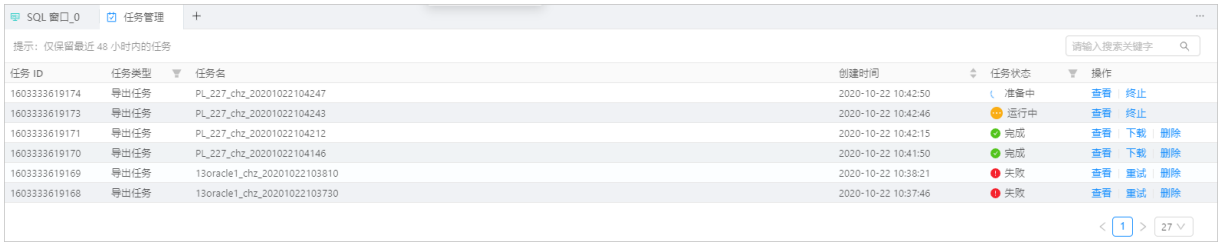
4.6.5. 任务管理

在OceanBase 开发者中心（OceanBase Developer Center，ODC）创建导入或导出任务后，ODC 会自动跳转至任务管理页面并展示导入或导出任务列表。您也可以通过单击上方导航栏中的 **任务** 按钮查看历史任务和进入任务管理页面。

> **说明**
>
> 最大支持 3 个导入导出任务并行运行，后续任务在队列中等待运行。



任务操作



进入任务管理页面后，任务列表会展示最近 48 小时内的任务。列表中会展示以下信息并且支持筛选和排序：

- 任务类型：展示该任务是导出任务或导入任务。
- 任务名：展示导出导入数据时 ODC 生成的默认名称或是您自定义的任务名。
- 创建时间：展示该任务生成的具体日期和时间。
- 任务状态：展示任务的当前状态，有 **准备中**、**运行中**、**已终止**、**完成** 和 **失败** 等五种状态。不同状态下提供的管理操作不同。
- 操作：提供了 **查看**、**下载**、**重试**、**终止** 和 **删除** 等任务管理操作。

操作	说明
查看	单击该按钮进入任务详情页面以查看目标任务的 任务信息 和 任务日志。
下载	任务成功时，单击该按钮下载导入导出文件至本地。
重试	任务失败时，单击该按钮重新发起该任务。
终止	任务状态为 准备中 或 运行中 时，单击该按钮终止任务。
删除	任务已终止、完成或失败时，单击该按钮从任务列表中删除目标任务。

任务信息

在任务详情页签的右上角选择 **任务信息** 会展示任务基本信息、导入导出文件信息和导入导出对象信息。

- 任务基本信息：展示 **任务名**、**连接名**、**模式名**、**任务类型**、**创建时间**、**重试次数** 和 **重试时间** 等信息。
- 导入导出文件信息：展示了创建导入导出任务时您选择的 **文件格式**、**文件编码**、**导入文件**、**导入/导出内容**、**导入对象大小**、**结构已存在时**、**导出结构设置** 和 **导出数据设置** 等信息。
- 导入导出对象信息：展示了该任务中您选择的对象在导出或导入数据时的完成情况，包括

  对象名称、**对象类型**、**结构处理状态**、**数据总记录数**、**数据已处理数** 和 **数据处理状态** 等信息。若您选择的是导入 CSV 文件时，还会显示 **字段映射** 信息。

任务日志

在任务详情页签的右上角选择 **任务日志** 以查看任务的告警日志和普通日志。

- 告警日志：告警日志展示任务的 **ERROR** 和 **WARN** 日志。当任务失败时，您看以查看告警日志以获得错误信息。
- 普通日志：普通日志展示任务的 **INFO** 日志。

4.7. 数据库对象

4.7.1. 表对象

4.7.1.1. 概述

表由行（Row）和列（Column）组成的。每列称为一个字段，每列的标题称为字段名。行包括了若干列信息项，一行数据称为一个或一条记录。一张数据库表由一条或多条记录组成，没有记录的表称为空表。每个表中通常都有一个主关键字，用于唯一地确定一条记录。

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击表标签可以查看表列表。在表列表中双击表名进入表管理页面，您可以在表管理页面的数据页签查看和修改表的数据，或在属性页签查看表的基本信息、列、分区、索引、约束和 DDL 等属性信息。

> 注意
>
> 分区信息查看依赖 root 账号，所以您需要在创建连接的高级配置中设置 root 用户。

本章节将分篇介绍以下 ODC 中表对象的相关操作：

- 新建表
- 表列表
- 表属性管理
- 表数据管理

4.7.1.2. 新建表

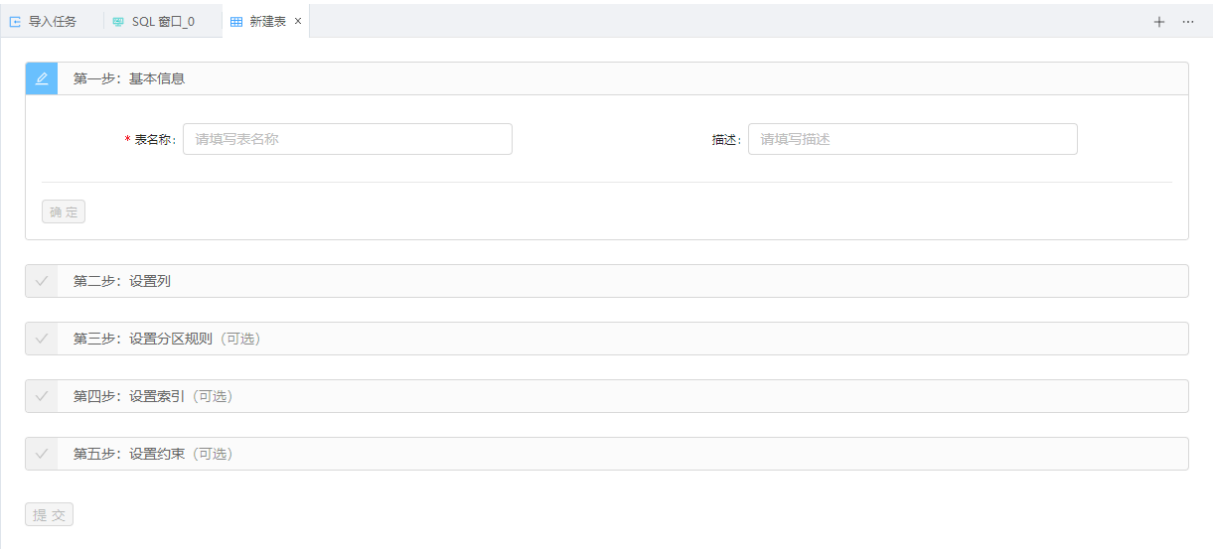
在 OceanBase 开发者中心（OceanBase Developer Center，ODC）单击连接名进入连接后，在左导航栏中单击 **表** 标签可以查看表列表。在表列表的右上角，单击 **+** 创建表，或是在上方导航栏中单击 **新建** 以创建您需要的对象。

如下图所示，创建表有以下 6 个步骤：

1. **设置基本信息**。
2. **设置列**。
3. **设置分区规则（可选）**。
4. **设置索引（可选）**。
5. **设置约束（可选）**。
6. 点击 **提交** 按钮，等待表生成。

设置基本信息

指定 **表名称** 和表的 **描述**（可选）。在 MySQL 模式下还需要指定 **默认字符集** 和 **默认排序规则**。



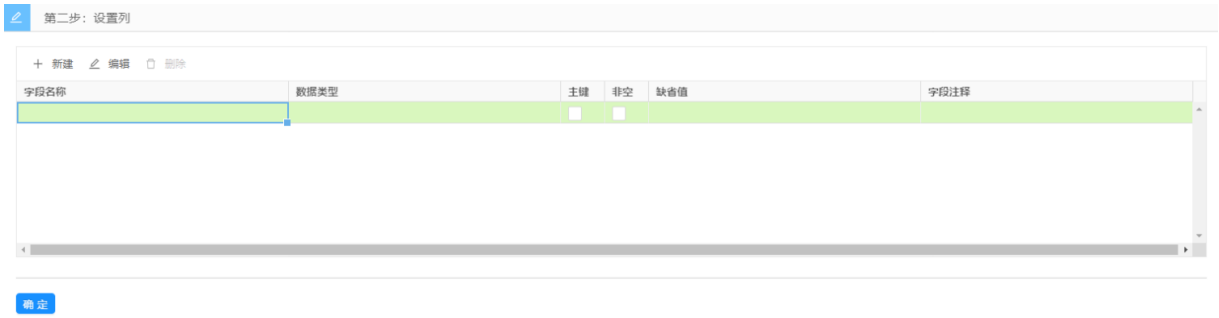
设置列

设置列是创建表的第二步，如下图所示，添加一列时您需要指定以下信息：

信息项	说明
字段名称	指定字段（列）的名称。
数据类型	指定字段（列）的数据类型。
主键	是否指定该字段（列）为主键。
非空	是否限制该字段（列）的值不能为空。
缺省值	该字段（列）的默认值。
字段注释	对该字段的说明。

设置列页面还提供了三个功能键：

- **+新建**：添加一下新字段。
- **编辑**：对选定的字段进行编辑或直接双击单元格进行编辑。
- **删除**：删除当前选中的字段。



设置分区规则

设置分区规则是创建表的第三步。OceanBase 中 MySQL 模式支持六种分区方法：key、Hash、Range、Range_colums、List 和 List_columns。Oracle 模式支持三种分区方法：List 、Range 和 Hash。

由于在 MySQL 模式和 Oracle 模式下分区的定义不同，所以以下信息在不同模式下的意义略有不同，请根据您选择的分区方法按其定义指定以下信息：

信息项	说明
分区方法	指定分区方法。MySQL 模式和 Oracle 模式下支持的分区方法不同。
字段	选择的字段将成为分区键。
表达式	根据表达式的返回值进行分区（Oracle 模式不支持分区表达式）。
分区	根据您选择的 **分区方法**，您可能需要指定 **分区名称**、**分区数量**、**区间上限值** 或 **值枚举** 等信息。您可以添加多个分区，并拖动它们进行排序。



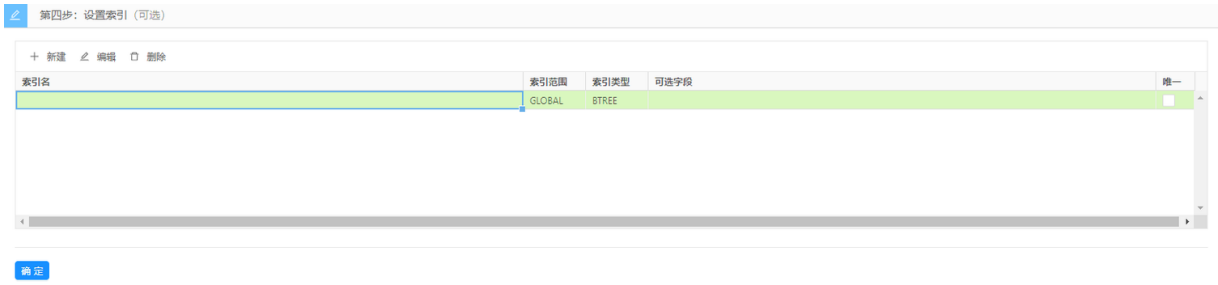
设置索引

设置索引是创建表的第四步，如下图所示，您需要指定以下信息：

信息项	说明
索引名	为改索引指定的名字。
索引范围	默认值是 GLOBAL（全局索引），只有设置了分区的表才可以指定索引范围是否为 LOCAL（局部索引）。
索引类型	索引类型目前只支持 BTree。
可选字段	在哪些列上建索引，要注意索引列的顺序。
唯一	是否唯一索引，即通过索引的方式保证唯一性约束。

设置索引页面还提供了三个功能键：

- **+新建**：添加一下新索引。
- **编辑**：对选定的索引进行编辑或直接双击单元格进行编辑。
- **删除**：删除当前选中的索引。



设置约束

设置约束是创建表的第五步，ODC 中支持设置的表级约束有以下 4 种：

- **主键约束**：定义一个主键来唯一标识表中的每一行数据。主键约束可以是一个字段或是一组字段，一张表中只能设置一个主键约束，且设置完成后不支持编辑。
- **唯一约束**：保证在一个字段或者一组字段里的数据在表中是唯一的，一张表中可以设置多个唯一约束。
- **外键约束**：在两个表的数据之间建立连接，它可以是一列或者多列，主要作用是保持关联表之间数据的一致性、完整性。设置完成后不支持新增和编辑。
- **检查约束**：在编辑数据库数据时，按照设置的检查规则进行校验，校验通过才允许数据修改操作。**检查约束** 仅在 Oracle 模式下支持。

由于 OceanBase 数据库中 MySQL 模式和 Oracle 模式支持的约束不同，而不同约束所需的信息也略有不同，所以请根据您选择的约束方法按页面上的要求指定所需信息：

信息项	说明
约束名称	指定约束的名称。
列信息	选择一个字段或是一组字段作为约束。
是否启用	指定创建的约束是否生效。有些情况下，您可能需要约束暂不生效，例如导入大量数据时，通过禁用约束可提高效率。
可延迟状态	支持设置三种延迟状态 **立即验证**、**不可延迟** 和 **延迟验证**。
关联schema（Oracle 模式下）	使用 **外键约束** 时需指定关联表（父表）所在的 Schema。
关联数据库（MySQL 模式下）	使用 **外键约束** 时需指定关联表（父表）所在的数据库。
关联表	使用 **外键约束** 时需指定关联表（父表）。
关联字段	使用 **外键约束** 时需指定关联表（父表）。
删除	指定当关联表（父表）中的数据被删除时，当前表（子表）对应的动作。可以指定 CASCADE、NOACTION、RESTRICT 和 SETNULL 等 4 种对应的动作。
更新	指定当关联表（父表）中的数据被更新时，当前表（子表）对应的动作。可以指定 CASCADE、NOACTION、RESTRICT 和 SETNULL 等 4 种对应的动作。
检查条件	指定执行 **检查约束** 时校验数据的检查规则。



设置约束页面还提供了三个功能键：

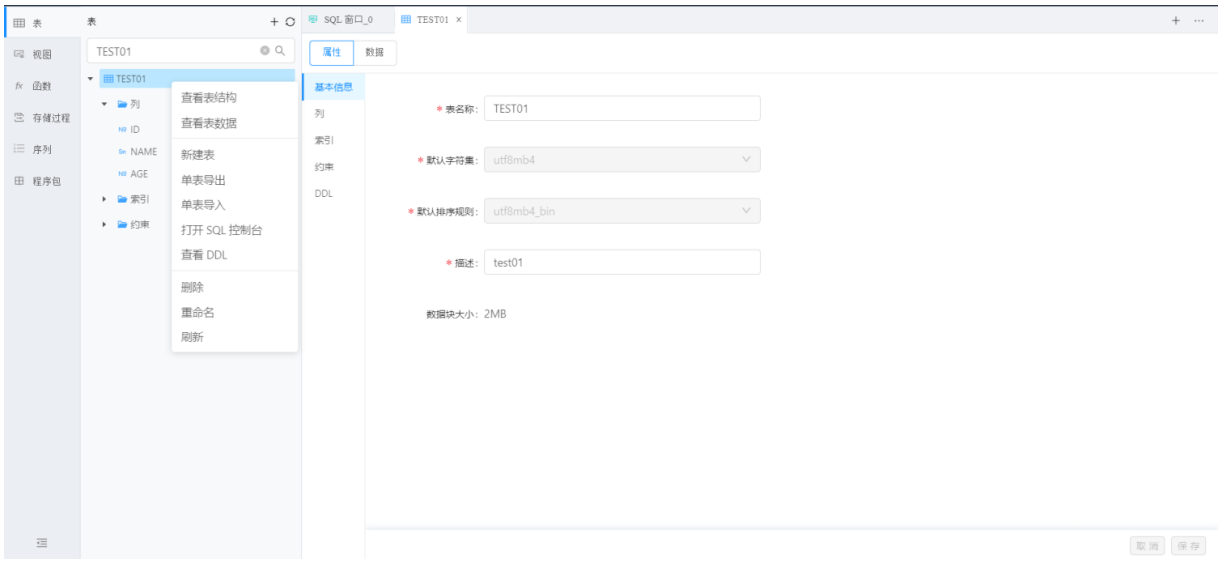
- **+新建**：添加一下新约束。
- **编辑**：对选定的约束进行编辑或直接双击单元格进行编辑。
- **删除**：删除当前选中的约束。

4.7.1.3. 表列表

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **表** 标签可以查看表列表。表列表中展示了当前连接的数据库中存在的表对象，您可以在表列表中查看该对象的 结构树 或右键目标对象以进行一些 管理操作。

结构树

如下图所示，在表列表中单击目标表名前的下弹标志会在弹出的下拉列表中展示其结构树，对象表的结构树最深有三层，右键结构树中不同层级中的项会提供对该项的一些操作键，操作键的具体信息请查看 管理操作 中的 **管理操作键表**：



管理操作

右键架构树中的目标项，在弹出的右键管理操作列表中 ODC 提供了一些操作键供您快速管理目标对象。

下述为管理操作键表：

操作键	说明
查看表结构	进入 表属性 页签，全量查看表的基本信息、列、索引、约束和 DDL 等信息。
查看表数据	进入 表数据 页签，查看表中的数据。
新建表	进入新建表页面，根据步骤创建新的表对象。

操作键	说明
单表导入	向当前表中导入数据，具体信息请查看本手册文档 单表导入。
单表导出	导出当前表中的数据，具体信息请查看本手册文档 单表导出。
打开 SQL 控制台	新建一个 SQL 窗口。
查看 DDL	进入表属性页签，查看定义当前表的 SQL 语句。
删除	删除当前表对象。
重命名	重命名当前表对象。
刷新	对当前表对象进行一些管理操作后，刷新结构树以展示最新信息。
查看列/索引/约束	进入表属性页签的列/索引/约束信息部分，查看表对象中列/索引/约束的具体信息。
新建列/索引/约束	弹出新建列/索引/约束页面。
删除列/索引/约束	删除目标列/索引/约。
编辑列/索引/约束	弹出编辑列/索引/约束页面以更改目标列/索引/约束的信息。

4.7.1.4. 表属性管理

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **表** 标签可以查看表列表。在表列表中双击表名进入表管理页面，点击上方导航栏中的 **属性** 按钮进入属性页签。在属性页签中，您可以查看当前表中以下属性的信息：

- **基本信息**
- **列**
- **索引**
- **约束**

- 分区
- DDL

基本信息

在 **属性** 页签的左导航栏中点击 **基本信息** 标签查看当前表的基本信息：

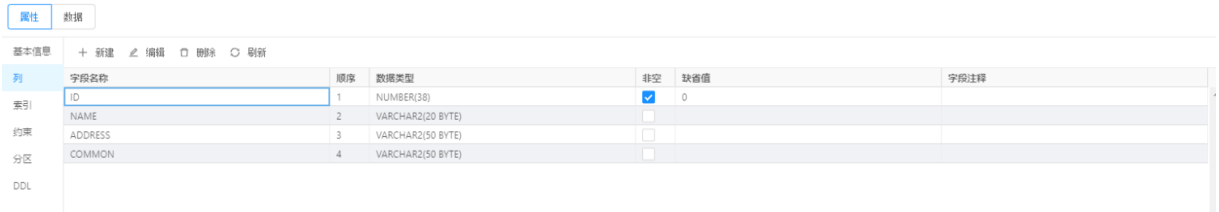
- **表名称**：显示当前表的名字，同时您可以在此修改表的名字。
- **默认字符集**：当前表所用的字符集。
- **默认排序规则**：当前表所用的排序规则。
- **描述**：显示创建表时您添加的描述，同时您可以在此修改对表的描述。
- **数据库的大小**：显示当前表的数据的大小。



列信息

在属性页签的左导航栏中点击 **列** 标签查看当前表中所包含的列，并且导航栏中为您提供了以下功能键：

- **新建**：在表中新增一个字段（列）。
- **编辑**：编辑当前选中的字段，在弹出的编辑窗口中您可以修改 **数据类型**、**主键**、**非空值**、
- **缺省值** 和 **字段注释** 等信息（**字段名称** 不可修改）。或双击单元格直接对其内容进行修改。
- **删除**：删除当前选中的字段。



索引信息

在属性页签的左导航栏中点击 **索引** 标签查看当前表中包含的索引，并且导航栏中为您提供了以下功能键：

- **新建**：在表中新增一个索引。
- **删除**：删除当前选中的索引。

> **说明**
>
> 目前暂不支持修改表中已经创建好的索引。



约束信息

在属性页签左导航栏点击的 **约束** 标签查看当前表中包含的约束信息，并且导航栏中为您提供了以下功能键：

- **新建**：在表中新增一个约束。
- **编辑**：编辑当前选中约束。
- **删除**：删除当前选中的约束。
- **刷新**：刷新约束列表。

> **说明**
>
> 不同约束方法支持的功能键不同，当导航栏中功能键为灰色或不存在时，表明该约束方法暂不支持该操作。



分区信息

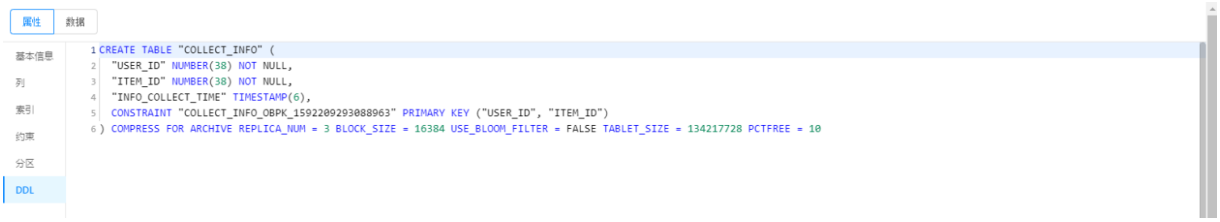
在属性页签的左导航栏中点击 **分区** 标签查看当前表中所包含的 **分区名称**、**顺序**、**分区方法**、**表达式**（Oracle 模式下展示分区键）、**区间上限值** 或 **值枚举** 等信息（不同分区方法展示的分区信息不同）。且导航栏中提供以下功能键：

- **新建**：即从未定义的区间范围划分出一个新分区。
- **编辑**：编辑选中的分区的分区信息。
- **分裂**：对已定义的分区进行拆分，使分区的粒度变小。
- **删除**：删除选中的分区，分区的数据也随之销毁。
- **刷新**：刷新分区列表。



DDL信息

您可以在 **属性** 页签中点击左导航栏中的 **DDL** 标签查看当前表的 DDL 语句，即查看定义当前表的 SQL 语句，不支持编辑。



4.7.1.5. 表数据管理

在OceanBase开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **表** 标签可以查看表列表。在表列表中双击表名进入表管理页面，点击上方导航栏中的 **数据** 按钮进入数据页签，在该页签可对当前表中的数据进行查看、新增、修改和删除操作。



	ID	name	birthday	score	city
1	702	ming	2001-01-01	10	shanghai
2	223	anna	2003-03-01	91	chengdu
3	001	tom	2003-03-01	82	chengdu
4	002	kimi	2004-04-01	73	hangzhou
5	003	sue	2003-03-01	64	beijing
6	004	liming	2006-06-01	51	beijing
7	005	zhangsan	2007-07-01	22	shanghai
8	006	lisi	2008-08-01	16	chengdu
9	007	wanger	2009-09-01	100	chengdu
10	008	sansan	2010-10-01	100	hangzhou
11	009	aya	2011-01-01	100	beijing
12	010	ben	2012-02-01	99	beijing
13	NULL	NULL	NULL	NULL	NULL

数据页签的导航栏中提供了以下功能键供您管理表中的数据：

- **添加行**：点击该功能键将在表中插入一个空行。双击单元格以插入数据，ODC 提供了校验和工具来帮助您更快捷方便的插入数据。例如，当前单元格在日期类型的列中，ODC 会为您提供一个日历窗口来帮助您直接选择要插入的日期值。
- **复制当前行**：在表中复制当前选中的行。
- **删除行**：在表中删除当前选中的行。
- **导出数据**：将表中数据导出，可以选择导出的文件类型（CSV 或 SQL）、字符集（GBK 或 UTF8）和行数。
- **列管理**：筛选在页面中展示的列。
- **展示数据量**：限制在页面中展示的行数。

ODC 中用颜色标识您对表中数据的操作，新增的数据用绿色标识，删除的数据用红色标识，修改后的数据用橙色标识。

4.7.2. 视图对象

4.7.2.1. 概述

视图是基于一个或多个表（或视图）而创建的虚拟表，视图基于的表称为基表。视图本身不包含数据，通过它可以对基表里面的数据进行查询和修改。所以视图的定义存在数据字典中，但是与此定义相关的数据并没有再存一份于数据库中。视图看上去非常像数据库的物理表，对它的操作同任何其它的表一样。当通过视图修改数据时，实际上是在改变基表中的数据，相反地，基表数据的改变也会反映在由基表产生的视图中。

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击视图标签可以查看视图列表。在视图列表中双击视图名进入视图管理页面，您可以在视图管理页面的数据页签查看视图的数据，或在属性页签查看视图的基本信息、列和 DDL 等属性信息。

本章节将分篇介绍以下 ODC 中视图对象的相关操作：

- 新建视图
- 管理视图

4.7.2.2. 新建视图

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **视图** 标签可以查看视图列表。在视图列表的右上角，点击 + 创建视图，或是在 ODC 上方导航栏中点击 **新建** 以创建您需要的对象。新建视图需要输入以下信息：

信息项	说明
视图名称	指定该视图的名称。
检查项	检查项是对写入数据的约束检查，OceanBase 暂时不支持，默认值为 NONE。
字段	设置视图中的字段。

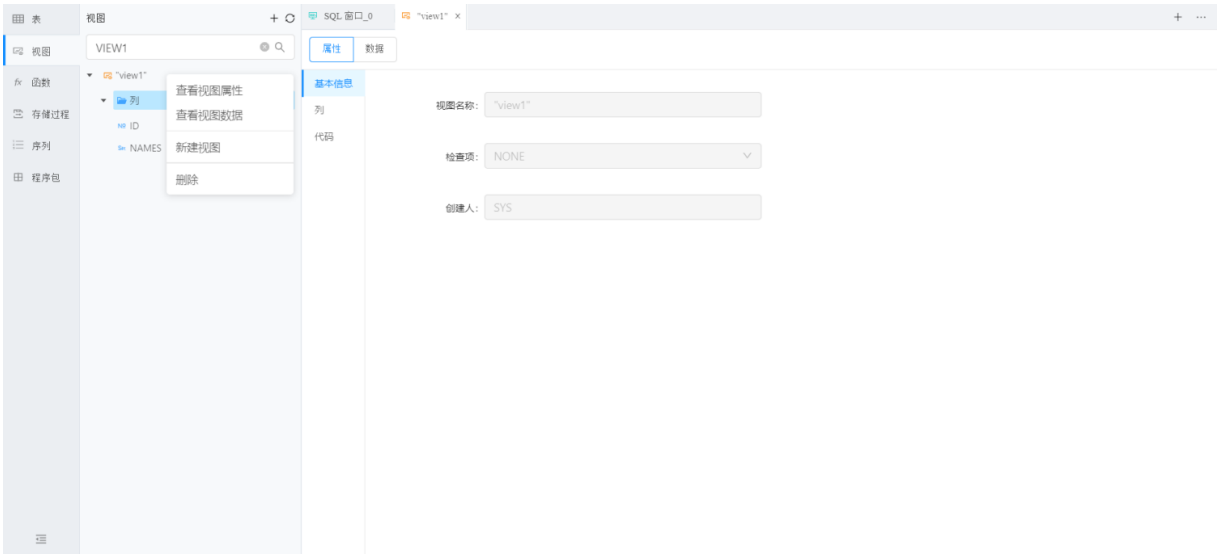


4.7.2.3. 管理视图

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **视图** 标签可以查看视图列表，在视图列表中展示了当前连接的数据库中存在的视图对象，您可以在视图列表中查看该对象的 **结构树** 或右键目标对象以进行一些 **管理操作**。或者在视图列表中双击目标视图名进入视图管理页面，您可以在视图管理页面的 **属性和数据页签**。

结构树

如下图所示，在视图列表中单击目标视图名前的下弹标志会在弹出的下拉列表中展示其结构树，对象视图的结构树最深有三层，右键结构树中不同层级中的项会提供对该项的一些操作键，操作键的具体信息请查看 **管理操作** 中的 **管理操作键表**：



管理操作

右键结构树中的目标项，在弹出的右键管理操作列表中 ODC 提供了一些操作键供您快速管理目标对象。

下述为管理操作键表：

操作键	说明
查看视图属性	进入 **属性** 页签，全量查看视图的基本信息、列和代码 等信息。
查看视图数据	进入 **数据** 页签，查看视图中的数据。
新建视图	进入新建视图页面，根据步骤创建新的视图对象。
删除	删除当前视图对象。
刷新	对当前视图对象进行一些管理操作后，刷新结构树以展示最新信息。
查看列	进入视图属性页签的列信息部分，查看视图对象中列的具体信息。

属性和数据页签

- 属性页签：
  - **基本信息**：查看 **视图名称**、**检查项** 和 **创建人** 等信息。
  - **列**：查看视图对应基表的 **字段名称**、**数据类型** 和 **字段注释** 等信息。
  - **DDL**：当前视图的定义脚本。
- 数据页签：展示视图中包含的字段名（列）。



4.7.3. 函数对象

4.7.3.1. 概述

函数即定义了一组 SQL 脚本的集合，可以在SQL语句中对函数进行调用以返回一个指定的结果。

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击函数标签可以查看函数列表。在函数列表中双击函数名进入函数管理页面，您可以在函数管理页面查看函数的基本信息、参数和 DDL 等信息。

本章节将分篇介绍以下 ODC 中函数对象的相关操作：

- 新建函数
- 管理函数

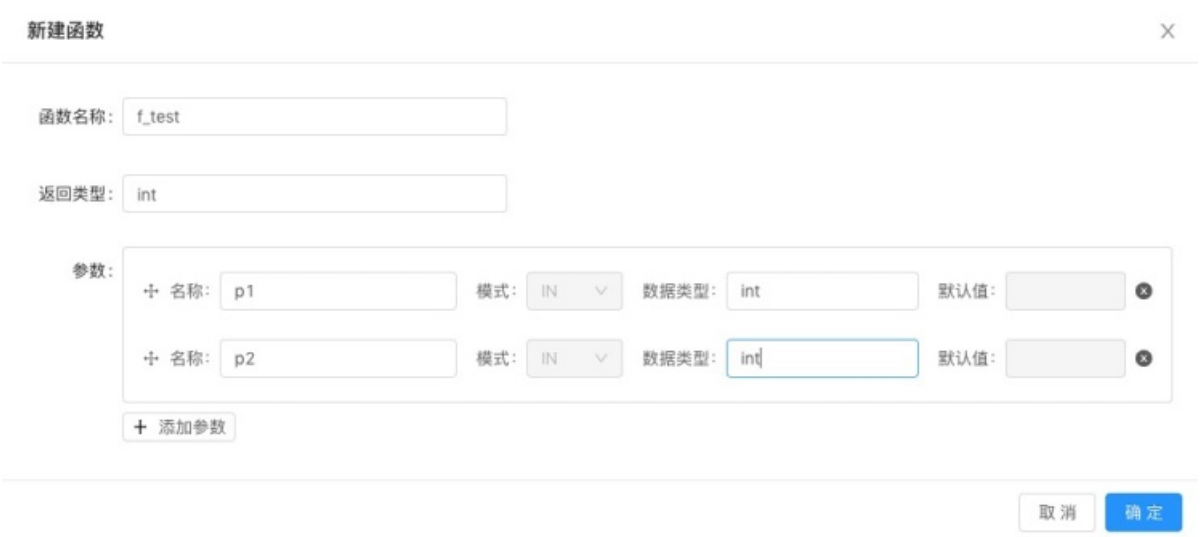
4.7.3.2. 新建函数

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 函数 标签可以查看函数列表。在函数列表的右上角，点击 + 创建函数，或是在 ODC 上方导航栏中点击 新建 以创建您需要的对象。创建函数需要输入以下信息：

信息项	说明
函数名称	指定函数的名称。
返回类型	函数返回值的数据类型。
参数	参数即调用函数时传入的信息，您需要指定参数的 **名称**、**模式**、**数据类型** 和 **默认值**（MySQL 模式下不支持设置 **默认值** 和 **模式**）。可拖动参数以调整顺序。

参数 中需指定的 **模式** 指参数类型。MySQL 模式下不支持设定参数模式，Oracle 模式下支持三种参数模式 IN（输入）、OUT（输出）、INOUT（输入输出）：

- **IN**：输入参数，调用函数时将参数的值传入函数供执行函数时使用。
- **OUT**：输出参数，调用函数时，函数会忽略输出参数本身的值并传入一个空值，在函数体中可以修改输出参数并将修改结果返回给输出参数所代表的实参。
- **INOUT**：输入输出参数，即同时具备输入参数和输出参数的功能。

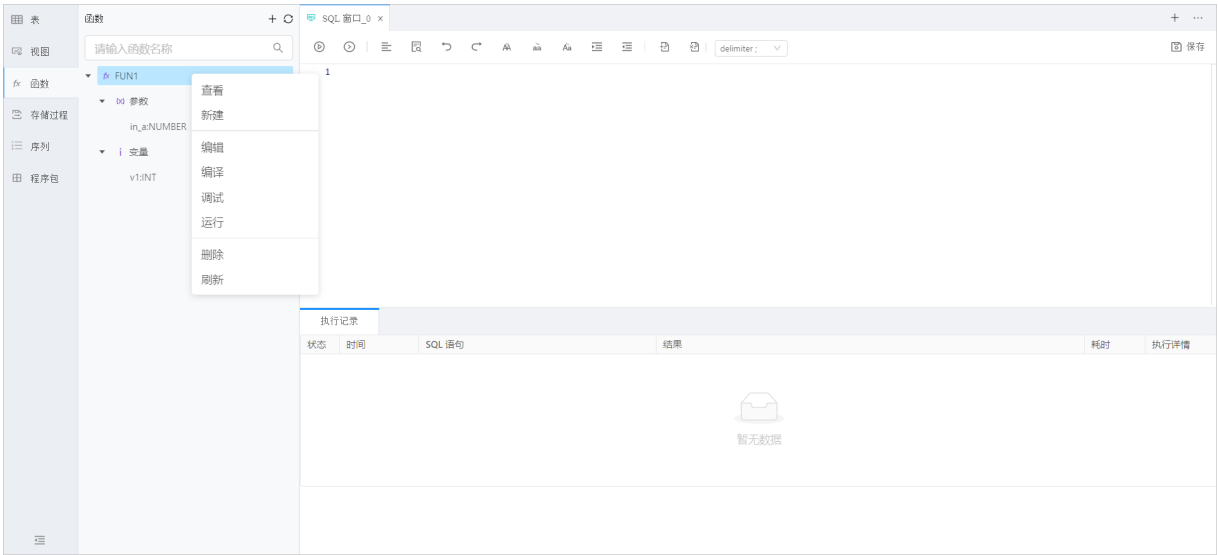


4.7.3.3. 管理函数

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **函数** 标签可以查看函数列表。在函数列表中展示了当前连接的数据库中存在的函数对象，您可以在函数列表中查看该对象的 **结构树** 或右键目标对象以进行一些 **管理操作**。或者，在函数列表中双击目标函数名进入 **函数管理页面**。

结构树

如下图所示，在函数列表中单击目标函数名前的下弹标志会在弹出的下拉列表中展示其结构树，对象函数的结构树最深有三层，右键结构树中不同层级中的项会提供对该项的一些操作键，操作键的具体信息请查看 **管理操作** 中的 **管理操作键表**：



管理操作

右键结构树中的目标项，在弹出的右键管理操作列表中 ODC 提供了一些操作键供您快速管理目标对象。

下述为管理操作键表：

操作键	说明
查看	进入 **函数管理页面**，全量查看当前函数的基本信息、参数和代码等信息。
新建	进入新建函数页面，根据步骤创建新的函数对象。
编辑	在 **函数管理页面** 中展示定义当前函数的语句，您可以在窗口中继续编辑该函数。
编译	编译目标函数。
调试	进入 **调试页面** 以对当前函数进行调试。PL 对象调试仅支持连接 OBServer V2.2.70、V3.0.00 及以后版本。
运行	运行当前函数。
删除	删除当前函数。
刷新	对当前函数对象进行一些管理操作后，刷新结构树以展示最新信息。

函数管理页面

在函数列表中双击目标函数名进入函数管理页面。您可以在函数管理页面查看以下信息：

- **基本信息**：展示 **函数名称**、**返回类型** 和 **创建人** 等信息。
- **参数**：展示参数 **名称**、**顺序**、**模式**、**数据类型**、**默认值** 等信息。
- **代码**：展示定义函数的脚本，还提供了 **编辑**（进入函数编辑页面）和 **查找** 两个功能键。



函数编辑页面

在函数列表中右键目标项，在提供的操作键中单击 **编辑** 可进入函数编辑页面，在编辑页面的代码区会展示该函数的定义语句， 您可以继续进行编辑。同时，工具栏提供了以下功能键：

功能	说明
确认修改	单击以跟新您本次修改。

功能	说明
编译	编译当前页面中的语句。
运行	单击该按钮运行代码区中的语句。
调试	单击该按钮进入调试页面，并对代码区内的 PL 对象进行调试。
格式化	点击该按钮对选中的 SQL 语句或当前 SQL 窗口中的所有 SQL 语句进行统一的格式化（缩进、换行和高亮关键字等操作）。
查找/替换	在查找框内输入内容以对脚本进行检索，查找后可在替换框内输入内容以对查找内容进行替换。
撤销	将脚本回退到上一步操作执行的结果。
重做	执行撤销操作后，对脚本重新执行撤销前的操作。
大写	将您在脚本中选中的语句全部转换为大写形式。
小写	将您在脚本中选中的语句全部转换为小写形式。
首字母大写	将您在脚本中选中的语句全部转换为首字母大写形式。
添加缩进	对您在脚本中选中的语句进行缩进操作。
删除缩进	删除您在脚本中选中的语句中所包含的缩进。
添加注释	对脚本中选中的语句添加注释。
删除注释	对脚本中选中的语句删除注释。

4.7.4. 存储过程对象

4.7.4.1. 概述

存储过程（Stored Procedure）是在数据库系统中，一组为了完成特定功能的 SQL 语句集，它存储在数据库中，一次编译后永久有效。

函数和存储过程的定义类似，都是 SQL 脚本的集合，但是二者主要区别如下：

- 函数一般只返回一个变量，而存储过程没有返回值。
- 函数是可以嵌入在 SQL 语句中使用，比如，可以在 `SELECT` 语句中调用函数，但是存储过程一般是作为一个独立的部分被执行。
- 函数限制比较多，而存储过程的限制相对就比较少。一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 存储过程 标签可以查看存储过程列表。在存储过程列表中双击存储过程名进入存储过程管理页面，您可以在存储过程管理页面查看存储过程的基本信息、参数和 DDL 等信息。

本章节将分篇介绍以下 ODC 中存储过程对象的相关操作：

- 新建存储过程
- 管理存储过程

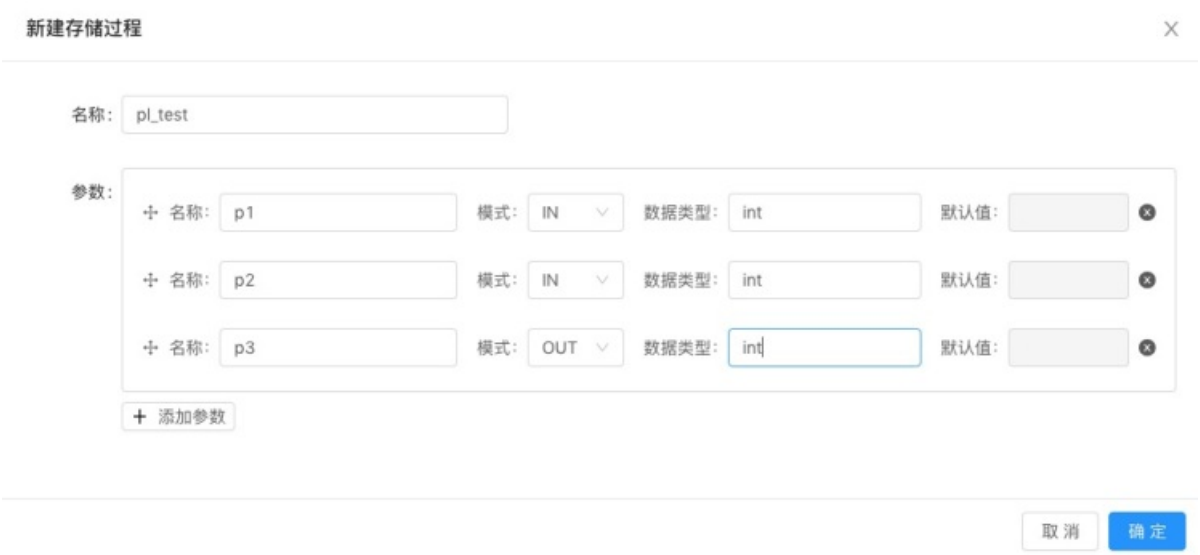
4.7.4.2. 新建存储过程

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **存储过程** 标签可以查看存储过程列表。在存储过程列表的右上角，点击 **+** 创建存储过程，或是在上方导航栏中点击 **新建** 以创建您需要的对象。创建存储过程需要输入以下信息：

信息项	说明
名称	指定存储过程的名称。
参数	参数即调用存储过程时传入的信息，您需要指定参数的 **名称**、**模式**、**数据类型** 和 **默认值**（MySQL 模式下不支持设置 **默认值**）。可拖动参数以调整顺序。

参数 需指定的 **模式** 指参数类型。有 IN（输入）、OUT（输出）、INOUT（输入输出）三种类型：

- **IN**：输入参数，调用存储过程时将参数的值传入存储过程供执行存储过程时使用。
- **OUT**：输出参数，调用存储过程时，存储过程会忽略输出参数本身的值并传入一个空值，执行结束后，输出参数会被赋予存储过程对它修改的值。一般用输出参数获取存储过程的执行结果数据。
- **INOUT**：输入输出参数，即同时具备输入参数和输出参数的功能。

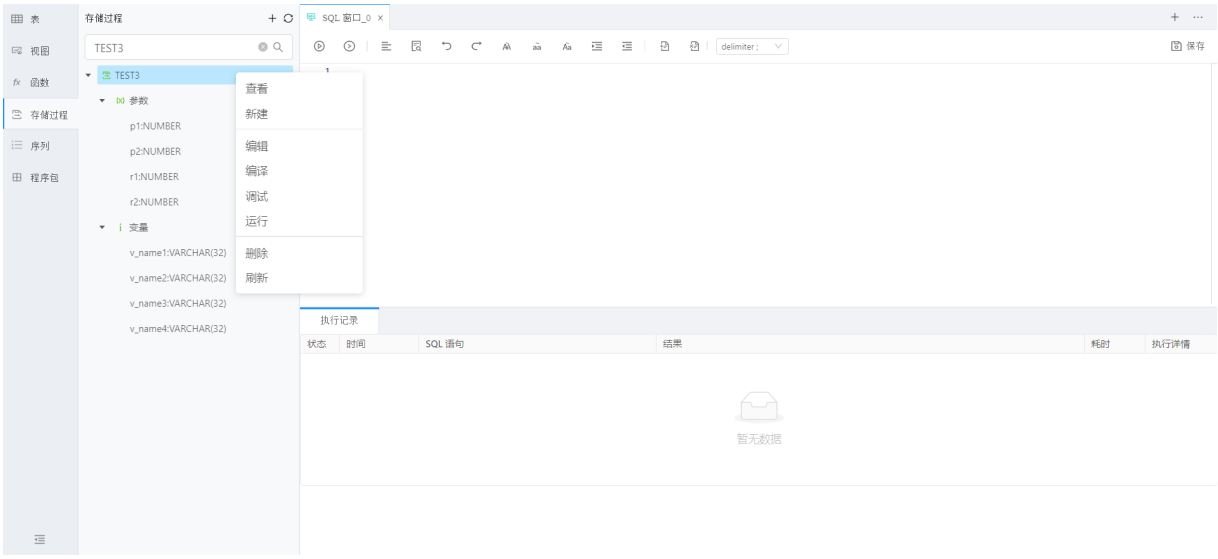


4.7.4.3. 管理存储过程

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **存储过程** 标签可以查看存储过程列表。在存储过程列表中展示了当前连接的数据库中存在的存储过程对象，您可以在存储过程列表中查看该对象的 **结构树** 或右键目标对象以进行一些 **管理操作**。或者，在存储过程列表中双击目标存储过程名进入 **存储过程管理页面**。

结构树

如下图所示，在存储过程列表中单击目标存储过程名前的下弹标志会在弹出的下拉列表中展示其结构树，对象存储过程的结构树最深有三层，右键结构树中不同层级中的项会提供对该项的一些操作键，操作键的具体信息请查看 **管理操作** 中的 **管理操作键表**：



管理操作

右键结构树中的目标项，在弹出的右键管理操作列表中 ODC 提供了一些操作键供您快速管理目标对象。

下述为管理操作键表：

操作键	说明
查看	进入 **存储过程管理页面**，全量查看当前存储过程的基本信息、参数和代码等信息。
新建	进入新建存储过程页面，根据步骤创建新的存储过程对象。
编辑	在 **存储过程编辑页面** 中展示定义当前存储过程的语句，您可以在窗口中继续编辑该存储过程。
编译	编译目标存储过程。
调试	进入 **调试页面** 以对当前存储过程进行调试。PL 对象调试仅支持连接 OBServer V2.2.70、V3.0.00 及以后版本。
运行	运行当前存储过程。
删除	删除当前存储过程。
刷新	对当前存储过程对象进行一些管理操作后，刷新结构树以展示最新信息。

存储过程管理页面

您可以在存储过程管理页面查看以下信息：

- **基本信息**：展示存储过程 **名称** 和 **创建人** 等信息。
- **参数**：展示参数 **名称**、**顺序**、**模式**、**数据类型**、**默认值** 等信息。
- **代码**：展示定义存储过程的脚本，还提供了 **编辑**（进入存储过程编辑页面）和 **查找** 两个功能键。



存储过程编辑页面

在存储过程列表中右键目标项，在提供的操作键中单击 **编辑** 可进入存储过程编辑页面，在编辑页面的代码区会展示该存储过程的定义语句， 您可以继续进行编辑。同时，工具栏提供了以下功能键：

功能	说明
确认修改	单击以跟新您本次修改。
编译	编译当前页面中的语句。
运行	单击该按钮运行代码区中的语句。
调试	单击该按钮进入调试页面，并对代码区内的 PL 对象进行调试。
格式化	点击该按钮对选中的 SQL 语句或当前 SQL 窗口中的所有 SQL 语句进行统一的格式化（缩进、换行和高亮关键字等操作）。
查找/替换	在查找框内输入内容以对脚本进行检索，查找后可在替换框内输入内容以对查找内容进行替换。
撤销	将脚本回退到上一步操作执行的结果。
重做	执行撤销操作后，对脚本重新执行撤销前的操作。
大写	将您在脚本中选中的语句全部转换为大写形式。
小写	将您在脚本中选中的语句全部转换为小写形式。
首字母大写	将您在脚本中选中的语句全部转换为首字母大写形式。
添加缩进	对您在脚本中选中的语句进行缩进操作。
删除缩进	删除您在脚本中选中的语句中所包含的缩进。
添加注释	对脚本中选中的语句添加注释。
删除注释	对脚本中选中的语句删除注释。

4.7.5. 序列对象

4.7.5.1. 概述

序列（Sequence）是指在数据库中按照一定规则自增的一种数字序列。因为自增所以不会出现重复的值，所以通常用来做为表的主键或唯一键。

> 注意
>
> 只有 Oracle 模式下支持创建序列对象。

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入一个 Oracle 模式的连接后，在左导航栏中点击序列标签可以查看序列列表。在序列列表中双击序列名进入序列管理页面，您可以在序列管理页面查看序列的基本信息、参数和 DDL 等信息。

本章节将分篇介绍以下 ODC 中序列对象的相关操作：

- 新建序列
- 管理序列

4.7.5.2. 新建序列

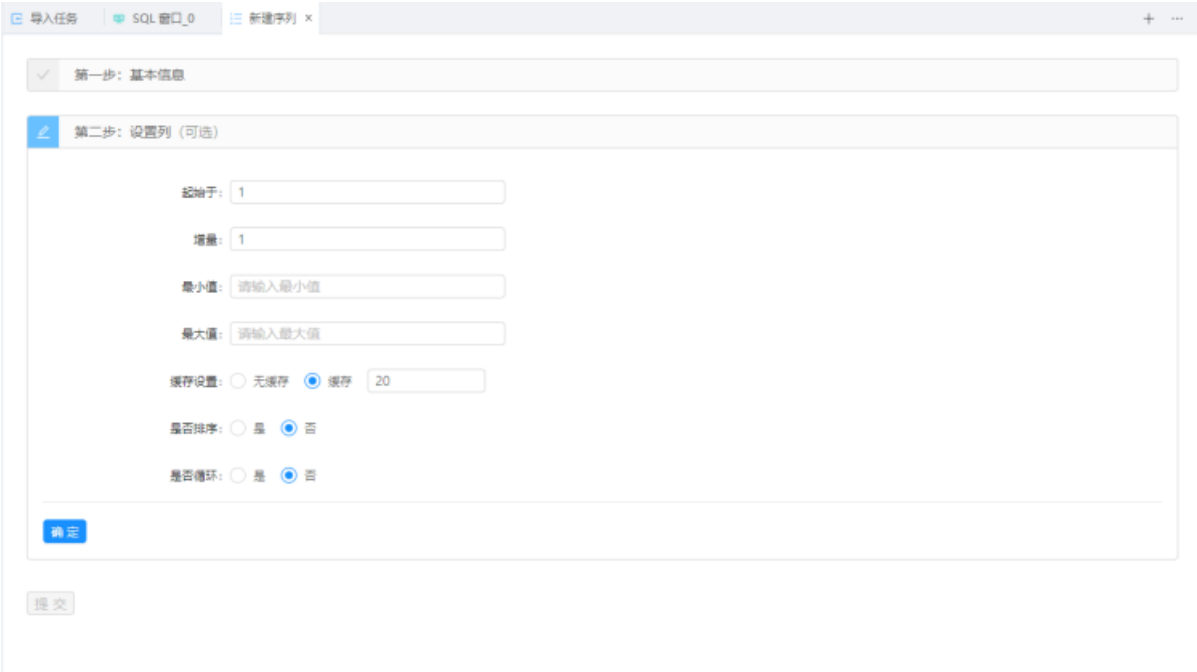
在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入一个 Oracle 模式的连接后，在左导航栏中点击 **序列** 标签可以查看序列列表。在序列列表的右上角，点击 **+** 创建序列，或是在 ODC 上方导航栏中点击 **新建** 以创建您需要的对象。创建序列有以下两个步骤：

1. 设置 **基本信息**：需要指定 **序列名称** 和 **当前用户**（根据当前连接账号自动填充）。
2. **设置列**（可选）。在该步骤需要指定以下信息：

信息项	说明
起始于	指定序列的起始值，要保证不小于属性 **最小值**。
增量	指定序列值自增时的步长，可以为负数。
最小值	指定序列可获取的最小值，数据库支持的最小值为-1026。
最大值	指定序列可获取的最大值，数据库支持的最大值为1027。
缓存设置	指定序列缓存的大小，合理设置缓存值可以提高序列取值的执行性能，也可设置 **无缓存**。
是否排序	指定序列的取值是否有序，有序取值场景下，序列的取值性能不如无序。

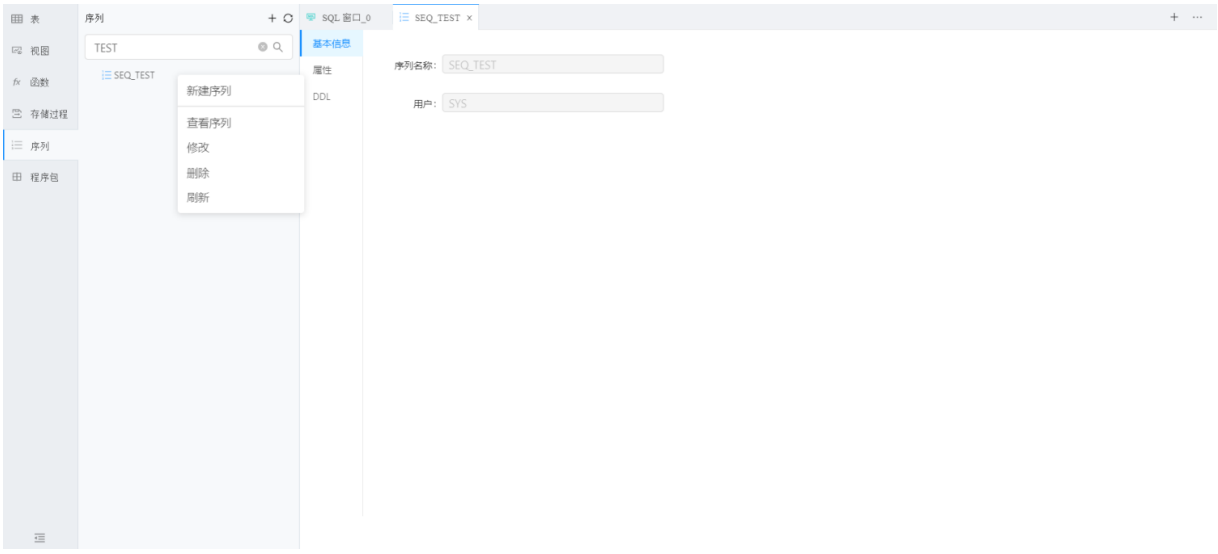
信息项	说明
是否循环	指定序列取值超过指定的 **最大值** 后，是否重新从最小值开始取值。

点击设置页面上的 **提交** 按钮后，根据您指定的信息会预生成一个 SQL 脚本，检查 SQL 脚本无误后点击页面右下角的 **执行** 按钮开始执行脚本并生成序列。



4.7.5.3. 管理序列

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入一个 Oracle 模式的连接后，在左导航栏中点击 **序列** 标签可以查看序列列表。您可以在序列列表中右键目标对象以进行一些 **管理操作**。或者，在序列列表中双击目标序列名进入 **序列管理页面**。



管理操作

在序列列表中右键目标对象，在弹出的右键管理操作列表中 ODC 提供了一些操作键供您快速管理目标对象。

下述为管理操作键表：

操作键	说明
新建序列	进入新建序列页面，根据步骤创建新的序列对象。
查看序列	进入 **序列管理页面**，全量查看当前序列的基本信息、属性和 DDL 等信息。
修改	进入 **序列管理页面** 的属性部分修改当前序列的信息。
删除	删除当前序列。
刷新	对当前序列对象进行一些管理操作后，刷新列表以展示最新信息。

序列管理页面

您可以在序列管理页面查看以下信息：

- **基本信息**：展示 **序列名称** 和 **用户名** 等信息。
- **属性**：展示属性 **下一个缓冲值**、**增量**、**最小值**、**最大值**、**缓存设置**、**是否排序** 和 **是否循环** 等信息。其中 **下一个缓冲值** 展示的是序列视图中 `last_number` 的值，该值仅做展示不允许修改。
- **DDL**：展示定义序列的脚本。



4.7.6. 程序包对象

4.7.6.1. 概述

程序包（Package）是由一组相关的函数，过程，变量，游标等 PL 程序设计元素的组合。程序包类似于 Java 中的类，变量类似于类中的成员变量，函数和过程类似 Java 类中的方法。其主要作用是把相同或相似的东西归类，方便维护和管理，提高开发效率，提高性能。

> 注意
>
> 只有 Oracle 模式下支持创建程序包对象。

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入一个 Oracle 模式的连接后，在左导航栏中点击程序包标签可以查看程序包列表。在程序包列表中双击程序包名进入程序包管理页面，您可以在程序包管理页面查看包头和包体的的基本信息和代码等信息。

本章节将分篇介绍以下 ODC 中程序包对象的相关操作：

- 新建程序包
- 管理程序包

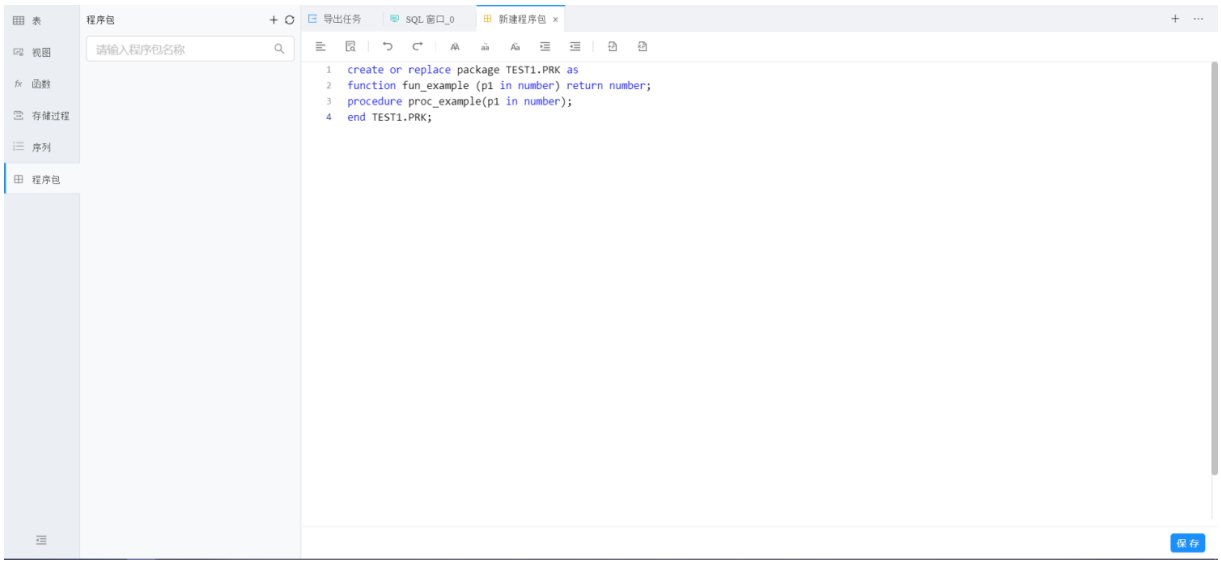
4.7.6.2. 新建程序包

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **程序包** 标签可以查看程序包列表。在程序包列表的右上角，点击 **+** 创建程序包，或是在上方导航栏中单击 **新建** 以创建您需要的对象。新建程序包需要分别创建包头和创建包体。

创建包头

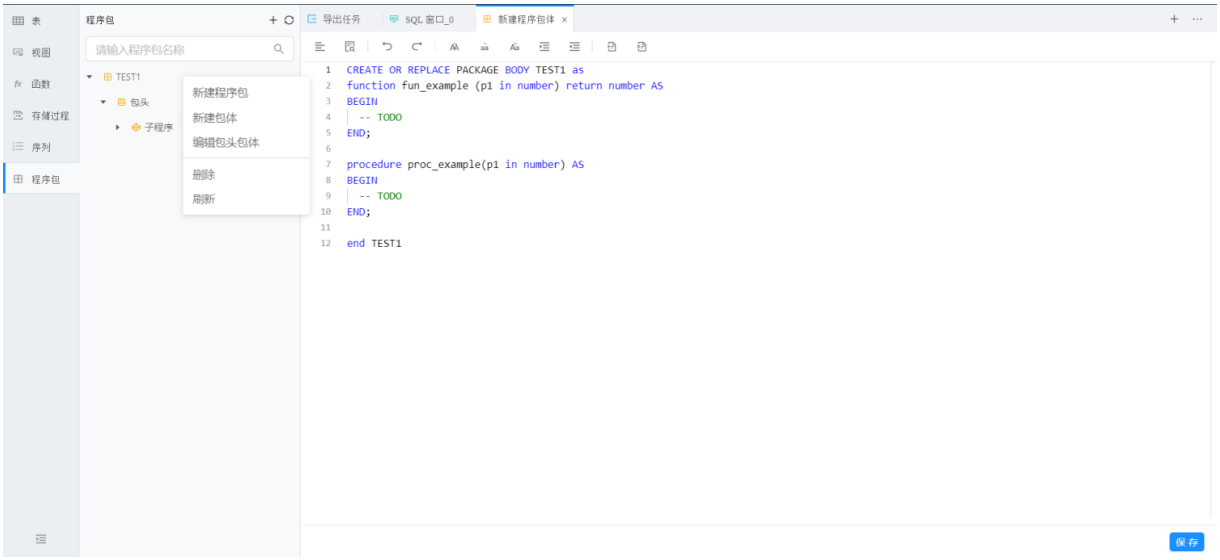
在程序包列表点击右上角的 **+** 或是在 ODC 导航栏中单击 **新建** 以进入新建程序包页面指定包头。

您需要在新建程序包页面中指定程序包名称，然后单击右下角的 **确认** 按钮进入包头定义页面。如下图所示，ODC 会为您自动生成模板语句，在包头定义页面的导航栏中，ODC 提供了功能键 **格式化** 方便您规范语句格式。完成包头定义语句后单击页面右下角的 **保存** 按钮生成包头：



创建包体

创建包头后，新建的程序包就会出现在程序包列表中，在列表中找到刚刚创建好的程序包，右键目标程序包名，单击 **新建包体** 以进入包体定义页面。如下图所示，和包头定义页面相似，ODC 会为您自动生成模板语句并在导航栏中提供功能键 **格式化** 方便您规范语句格式。完成包体定义语句后单击页面右下角的 **保存** 按钮生成包体：

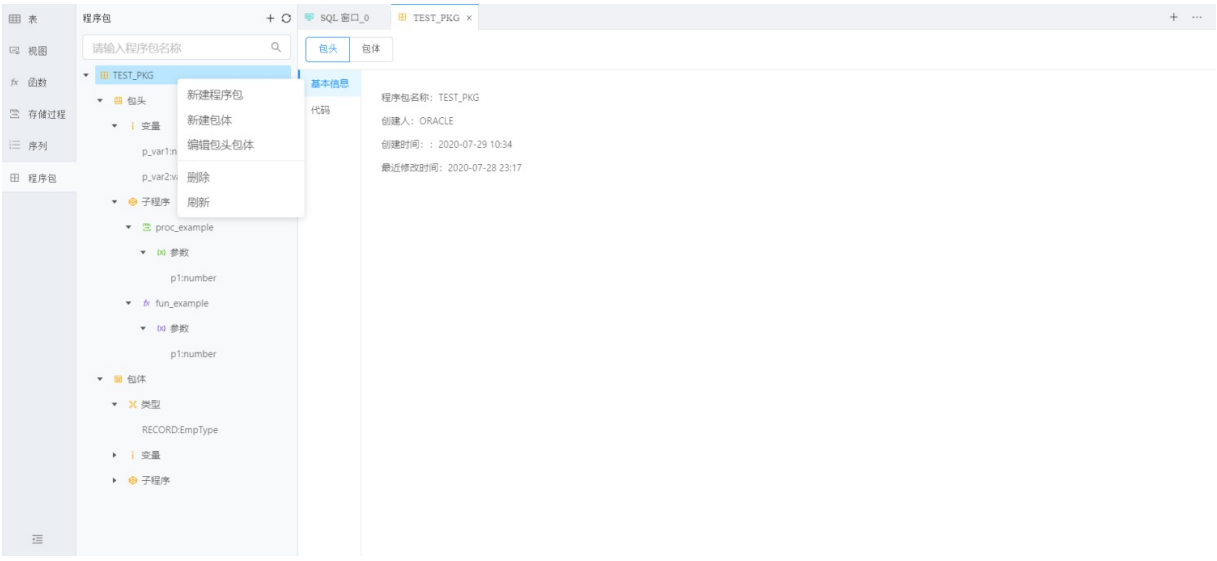


4.7.6.3. 管理程序包

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）点击连接名进入连接后，在左导航栏中点击 **程序包** 标签可以查看程序包列表。在程序包列表中展示了当前连接的数据库中存在的程序包对象，您可以在程序包列表中查看该对象的 **结构树** 或右键目标对象以进行一些 **管理操作**。或者，在程序包列表中双击目标程序包名进入程序包管理页面，程序包管理页面提供了 **包头页签** 和 **包体页签**。

结构树

如下图所示，在程序包列表中单击目标程序包名前的下弹标志会在弹出的下拉列表中展示其结构树，对象程序包的结构树最深有六层，右键结构树中不同层级中的项会提供对该项的一些操作键，操作键的具体信息请查看 **管理操作** 中的 **管理操作键表**：



管理操作

右键结构树中的目标项，在弹出的右键管理操作列表中 ODC 提供了一些操作键供您快速管理目标对象。

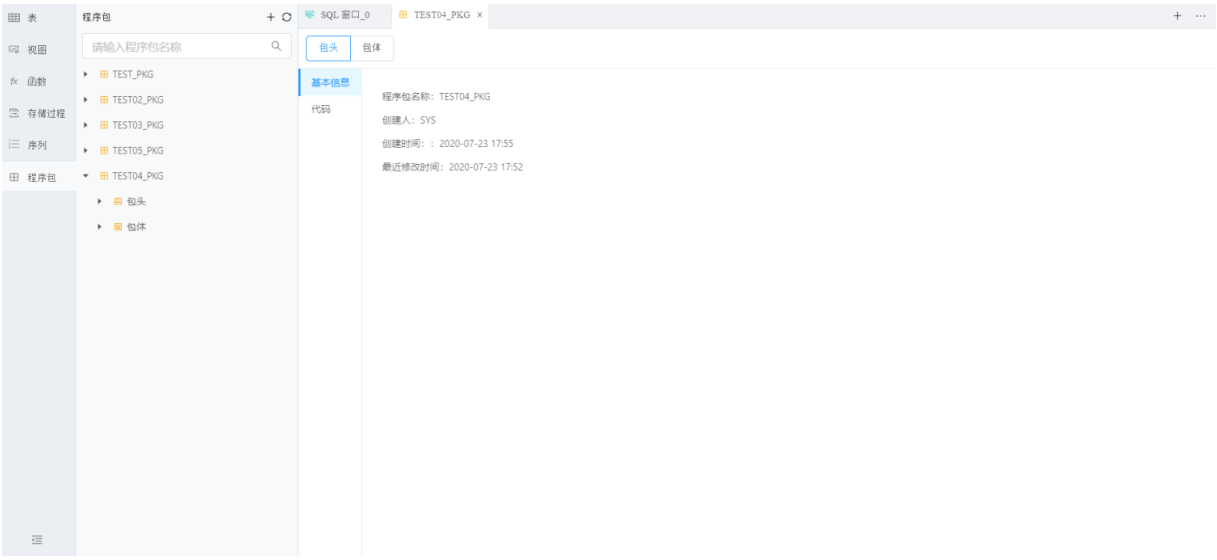
下述为管理操作键表：

操作键	说明
新建程序包	进入新建程序包页面，根据步骤创建新的程序包包头。
新建包体	进入包体定义页面以新建包体。
编辑包头包体	同时打开 **包头/包体编辑页面**，您可以在窗口中继续编辑包头和包体。
删除程序包	删除当前程序包。
刷新	对当前程序包对象进行一些管理操作后，刷新结构树以展示最新信息。
查看	进入 **包头页签** 或是 **包体页签**，全量查看当前程序包的基本信息和代码等信息。
编辑	在 **包头/包体编辑页面** 中展示定义包头/包体的语句，您可以在窗口中继续编辑包头/包体。
编译	编译当前程序包包体。
运行	运行程序包体子程序中的存储过程或函数。
删除	删除当前程序包包体。

包头页签

如下图所示，在程序包管理页面，单击导航栏中的 **包头** 标签会展示以下信息：

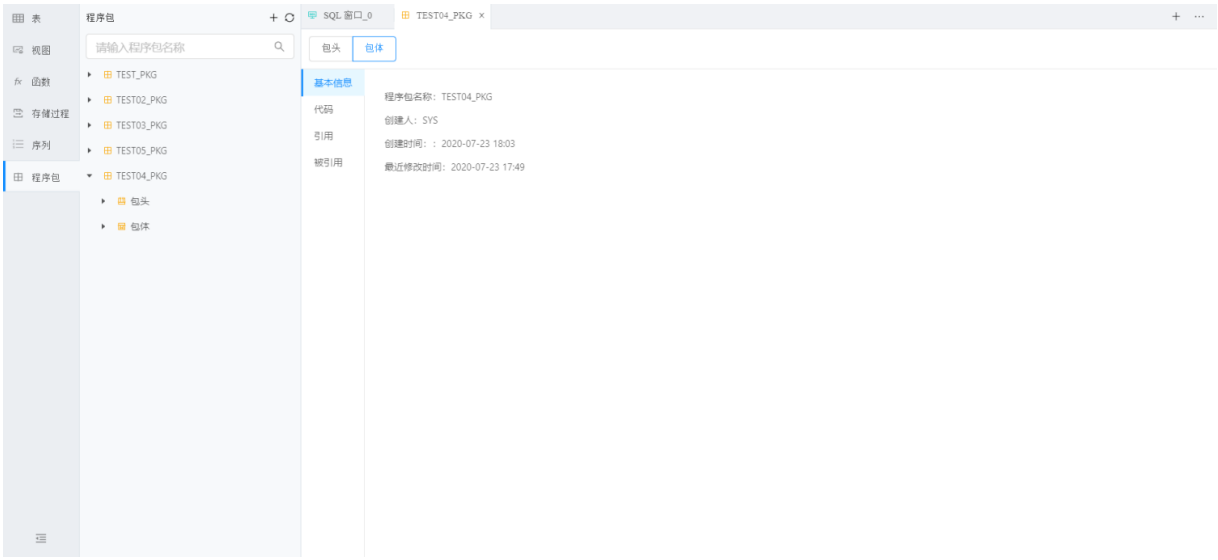
- **基本信息**：展示 **程序包名称**、**创建人**、**创建时间** 和 **最近修改时间** 等程序包信息。
- **代码**：展示包头的定义代码，还提供了 **编辑**（进入包头编辑页面）和 **查找** 两个功能键。



包体页签

如下图所示，在程序包管理页面，单击导航栏中的 **包体** 标签会展示以下信息：

- **基本信息**：和包头页签相同，展示 **程序包名称**、**创建人**、**创建时间** 和 **最近修改时间** 等程序包信息。
- **代码**：展示包体的定义代码，还提供了 **编辑**（进入包体编辑页面）和 **查找** 两个功能键。



包头/包体编辑页面

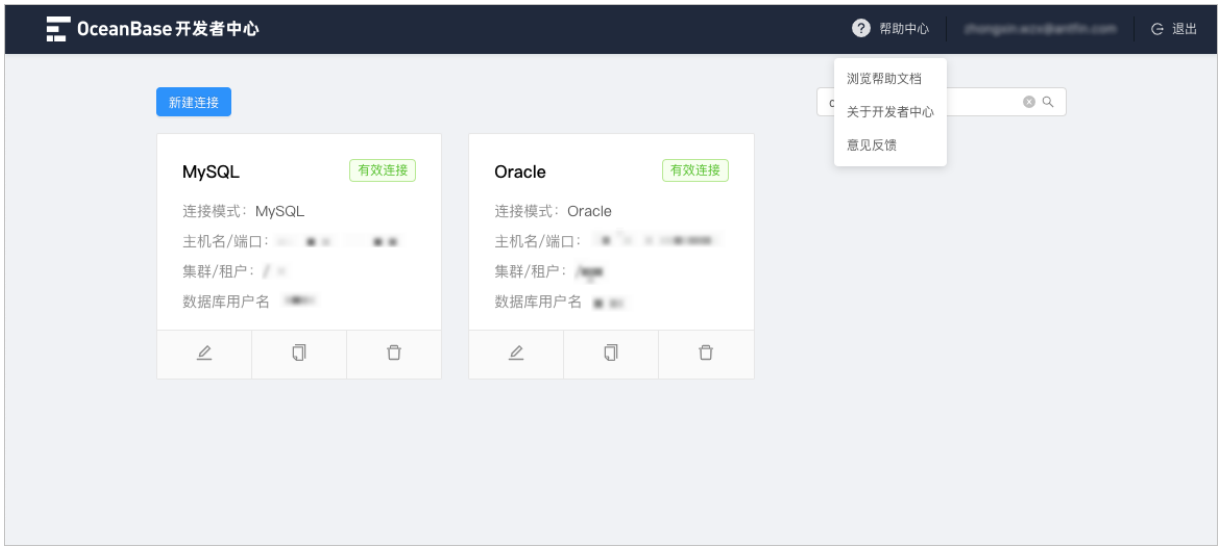
在程序包列表中右键目标项，在提供的操作键中单击 **编辑** 可进入程序包编辑页面，在编辑页面的代码区会展示该程序包的定义语句， 您可以继续进行编辑。同时，工具栏提供了以下功能键：

功能	说明
确认修改	单击以跟新您本次修改。
编译	编译当前页面中的语句。

功能	说明
格式化	点击该按钮对选中的 SQL 语句或当前 SQL 窗口中的所有 SQL 语句进行统一的格式化（缩进、换行和高亮关键字等操作）。
查找/替换	在查找框内输入内容以对脚本进行检索，查找后可在替换框内输入内容以对查找内容进行替换。
撤销	将脚本回退到上一步操作执行的结果。
重做	执行撤销操作后，对脚本重新执行撤销前的操作。
大写	将您在脚本中选中的语句全部转换为大写形式。
小写	将您在脚本中选中的语句全部转换为小写形式。
首字母大写	将您在脚本中选中的语句全部转换为首字母大写形式。
添加缩进	对您在脚本中选中的语句进行缩进操作。
删除缩进	删除您在脚本中选中的语句中所包含的缩进。
添加注释	对脚本中选中的语句添加注释。
删除注释	对脚本中选中的语句删除注释。

4.8. 帮助中心

在 OceanBase 开发者中心（OceanBase Developer Center，ODC）首页右上角的帮助中心，您可以浏览帮助文档、下载产品手册并在 **关于开发者中心** 和 **意见反馈** 页签获得产品信息与服务邮箱。

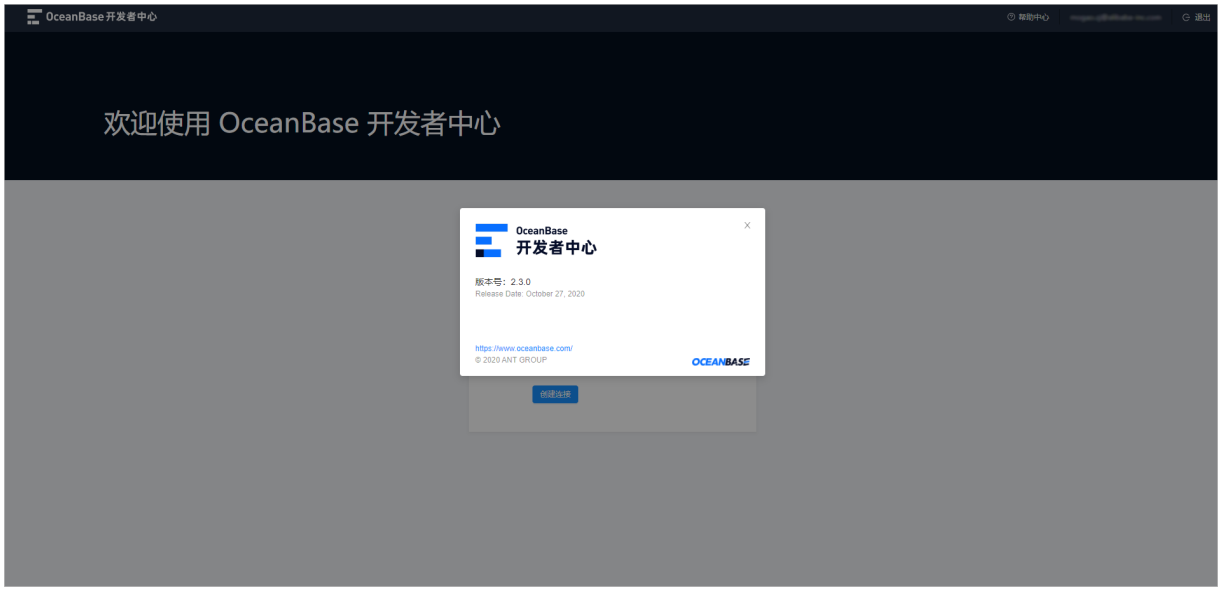


浏览帮助文档

单击帮助中心下拉列表中的 **浏览帮助文档** 标签，Web 版 ODC 会在浏览器的新页签中展示对应版本的帮助文档，客户端版 ODC 会弹出文档浏览窗。若您需要查看最新或历史版本的产品文档，可在 OceanBase 文档中心 进行浏览。

关于开发者中心

单击帮助中心下拉列表中的 **关于开发者中心** 标签，在弹出的窗口中您可以获得产品名称、版本号、发版日期、官网地址和版权等信息。



意见反馈

单击帮助中心下拉列表中的 **意见反馈** 标签，在弹出的窗口中您可以获得我们的支持邮箱（odcsupport@service.alipay.com），通过邮箱您可以与我们取得联系并反馈您的宝贵意见。

5.SQL 参考（MySQL 模式）

5.1. 基本元素

5.1.1. 数据类型

OceanBase支持如下数据类型：

- 数值类型
- 日期时间类型
- 字符类型
- 大对象类型

数值类型

数值类型可以划分为三类：

- 整数类型：`BOOL` / `BOOLEAN` 、`TINYINT` 、`SMALLINT` 、`MEDIUMINT` 、`INT` / `INTEGER` 、`BIGINT` 。
- 定点类型：`DECIMAL` / `NUMERIC` 。
- 浮点类型：`FLOAT` 、`DOUBLE` 。
- Bit-Value类型：`BIT`

所有数据类型都可以同时使用 `UNSIGNED` 关键字声明为无符号类型，这会导致值域发生变化。

数值类型在定义时可以指定precision和scale，不同类型中precision和scale的含义可能有所不同，详情请见各类型详细说明。

整数类型

整数类型为定长、精确数值类型，值域取决于类型长度，以及是否为无符号，precision只表示最小显示宽度，详见“ZEROFILL属性”节。以下为相关信息：

类型	长度(字节)	值域(有符号)	值域(无符号)
`BOOL` / `BOOLEAN` / `TINYINT`	1	[-2 ^ 7 , 2 ^ 7 - 1]	[0, 2 ^ 8 - 1]
`SMALLINT`	2	[-2 ^ 15, 2 ^ 15 - 1]	[0, 2 ^ 16 - 1]
`MEDIUMINT`	3	[-2 ^ 23, 2 ^ 23 - 1]	[0, 2 ^ 24 - 1]

类型	长度(字节)	值域(有符号)	值域(无符号)
INT / INTEGER	4	[-2 ^ 31, 2 ^ 31 - 1]	[0, 2 ^ 32 - 1]
BIGINT	8	[-2 ^ 63, 2 ^ 63 - 1]	[0, 2 ^ 64 - 1]

BOOL 等价于 BOOLEAN，这两个类型又等价于 TINYINT(1)。

INT 等价于 INTEGER。

定点类型

定点类型为变长、精确数值类型，值域和精度取决于precision和scale，以及是否为无符号。precision和scale分别表示十进制下的总最大有效位数、小数部分最大有效位数，整数部分最大有效位数等于precision - scale，其中precision和scale的最大值分别为65、30，默认值分别为10、0。例如：

- DECIMAL(5, 2)，整数部分和小数部分最大有效位数分别为3、2，所以值域为[-999.99, 999.99]。
- 如果同时定义为 UNSIGNED，则值域为[0, 999.99]。

DECIMAL 等价于 NUMERIC。

浮点类型

浮点类型为定长、非精确数值类型，值域和精度取决于类型长度、precision和scale，以及是否为无符号。precision和scale分别表示十进制下的总最大有效位数、小数部分最大有效位数，整数部分最大有效位数等于precision - scale，其中precision和scale的最大值分别为53、30。

浮点类型的精度只是IEEE标准中规定的理论值，实际情况可能因硬件或操作系统限制略有不同。

以下为不指定precision和scale时的默认信息：

类型	长度(字节)	值域(有符号)	值域(无符号)	精度
FLOAT	4	[-2 ^ 128, 2 ^ 128]	[0, 2 ^ 128]	7位
DOUBLE	8	[-2 ^ 1024, 2 ^ 1024]	[0, 2 ^ 1024]	15位

如果指定precision和scale，则值域确定方法与定点类型相同。

ZEROFILL属性

数值类型在定义时可以通过ZEROFILL关键字指定最小显示宽度，同时将该类型隐式定义为UNSIGNED。在数据实际显示宽度不足最小显示宽度时，通过先在小数部分补零到scale上限、再在整数部分补零到precision上限的方式，将显示宽度补足到最小显示宽度。例如：

- INT(5) ZEROFILL：当数据值为 123 时，将显示为 00123。

- `DECIMAL(10, 5) ZEROFILL`：当数据值为 `123.456` 时，将显示为 `00123.45600`。

BIT-Value类型

BIT数据类型用于存储bit values。一个BIT(M)能够存储M-bit的值，M的范围是1~64。

bit value通过b'value'的形式指定，**value**是用0和1来指定的，例如,b'111'表示7，b'10000000'表示128。

当向BIT(M)列插入值时，如果插入值的长度小于**M**，则会在左侧填充0。例如：将b'101'插入到BIT(6)时，相当于插入了b'000101'。

日期时间类型

以下为相关信息：

类型	格式	下界	上界	含义
`DATETIME`	YYYY-MM-DD HH:MM:SS[.fraction]	0000-01-01 00:00:00.000000	9999-12-31 23:59:59.999999	日期时间(不考虑时区)
`TIMESTAMP`	YYYY-MM-DD HH:MM:SS[.fraction]	0000-01-01 00:00:00.000000	9999-12-31 23:59:59.999999	日期时间(考虑时区)
`DATE`	YYYY-MM-DD	0000-01-01	9999-12-31	日期
`TIME`	HH:MM:SS[.fraction]	-838:59:59.000000	838:59:59.000000	时间
`YEAR`	YYYY	1901	2155	年份

`DATETIME`、`TIMESTAMP`、`TIME`的值域和精度取决于scale。该属性表示小数部分最大有效位数，最大值为6，默认值为0。

字符类型

以下为普通字符类型相关信息：

类型	长度	定义长度上限(字符)	字符集
`VARCHAR`	变长	262144 / 256K	`UTF8MB4`
`VARBINARY`	变长	1048576 / 1M	`BINARY`

类型	长度	定义长度上限(字符)	字符集
CHAR	定长	256	UTF8MB4
BINARY	定长	256	BINARY
enum	变长	最多定义65535个元素，每个元素长度最多255个字符	UTF8MB4
set	变长	最多定义64个元素，每个元素长度最多255个字符	UTF8MB4

大对象类型

以下为大对象类型相关信息：

类型	长度	存储长度上限(字节)	字符集
TINYTEXT	变长	256	UTF8MB4
TINYBLOB	变长	256	BINARY
TEXT	变长	65536 / 64K	UTF8MB4
BLOB	变长	65536 / 64K	BINARY
MEDIUMTEXT	变长	16777216 / 16M	UTF8MB4
MEDIUMBLOB	变长	16777216 / 16M	BINARY
LONGTEXT	变长	50331648 / 48M	UTF8MB4
LONGBLOB	变长	50331648 / 48M	BINARY

5.1.2. 表达式

表达式是广义概念，通常有若干个输入参数，并且返回一个输出结果。其中输入参数可能来自于常量或单行数据，也可能来自于多行数据。表达式可以组合，一个表达式的输入可以为另一个表达式的输出。

根据表达式的来源和形式，可分为如下几类：

- 列引用
- 常量
- 运算符
- 函数

例如：

```
SELECT ABS(a + 1)
 FROM t1
WHERE a > 0;
```

- a为列引用。
- 0、1为常量。
- `>`、`+`为运算符，以0、1、a表达式为输入。
- `ABS`为函数，以`+`表达式为输入。

5.1.3. 类型转换

OceanBase支持显式类型转换和隐式类型转换。

显示类型转换通过`CAST`函数实现。

隐式类型转换发生在这样的场景中：一个操作需要一个指定类型的参数，但语句的实际参数值并不是指定类型，这时OceanBase会将实际参数值转为指定类型后，再进行后续操作。

5.1.4. 字符集

目前只支持如下字符集：

- `UTF8MB4`：变长编码，字符最大长度4字节。
- `BINARY`：定长编码，字符固定长度1字节。

`UT8`/`UTF8MB3`是`UTF8MB4`的子集，变长编码，字符最大长度3字节。为支持无缝迁移，OceanBase在语法上将`UTF8`视为`UTF8MB4`的同义词。

5.1.5. 排序规则

目前只支持如下排序规则：

- `UTF8MB4`的`UTF8MB4_GENERAL_CI`。

- `UTF8MB4` 的 `UTF8MB4_BIN` 。
- `BINARY` 的 `BINARY` 。

5.1.6. 数据比较规则

OceanBase允许任意两或多个任意类型的数据进行比较，比较结果可能为：

- 非0 / TRUE
- 0 / FALSE。
- NULL。

如果参与比较的数据类型不一致，OceanBase会根据相关规则确定一个用于比较的数据类型，逻辑上所有参与比较的数据都需要先转换为该数据类型才能参与比较操作。

如果比较类型为文本类型，则还需要确定一个用于比较的排序规则。

5.1.7. 字面量

文本

文本是使用单引号 `'` 或双引号 `"` 引起来的字符序列，如果打开 `ANSI_QUOTES` 模式，那只有单引号 `'` 可以用于引用字符串。

数值

十进制数值可以分为精确数值（整数和定点数值）和浮点数值。数值可以使用小数点 `.` 作为十进制分隔符，也可以在前面加一个负号 `-` 来表示负值。

十六进制数值只支持整数数值，以前缀 `0X` 开始，允许出现字母 `A` 到 `F` ，所有字母不区分大小写。

日期时间

日期时间字面量有文本或数值两种形式。

- 文本格式可使用全量分隔符： `'2015-07-21 12:34:56.789'` ，或完全不使用分隔符： `'20150721'` 。
- 数值格式只允许使用小数点 `.` 作为秒和微秒部分的分隔符： `20150721123456.789` 。
- 在需要使用分隔符时，除了秒和微秒部分之间只能使用小数点 `.` 外，其它分隔符建议只使用 `-` 、 `/` 、 `:` 等常用分隔符。

转义字符

转义字符是在字符串中，某些序列前添加反斜线 `\` ，用于表示特殊含义。转义字符对大小写敏感。例如 `\b` 表示退格，但 `\B` 表示 `B` 。

以下为所有转义符列表：

转义符	含义
\b	退格符。
\f	换页符。
\n	换行符。
\r	回车符。
\t	tab字符。
\\	反斜线字符。
\'	单引号。
\"	双引号。
_	下划线。
\%	百分号。
\0	NULL。
\Z	ASCII 26，控制符Ctrl+Z。

5.1.8. 注释

SQL语句

在普通SQL语句中，OceanBase支持如下3种注释方法：

- 从 # 到行尾
- 从 __ 到行尾

- 从 /* 到 */

数据库对象

在DDL语句中通过 COMMENT 子句可以位数据库对象指定注释。例如：

```
create table t(pk INT PRIMARY KEY COMMENT '主键');
```

5.2. 运算符

5.2.1. 算术运算符

以下为所有算术运算符列表：

运算符	操作数	含义
+	一元/二元	一元表示正数，二元表示加法。
-	一元/二元	一元表示负数，二元表示减法。
*	二元	乘法。
/	二元	普通除法。
DIV	二元	整数除法，返回商数。
MOD 或 %	二元	整数除法，返回余数。

整数除法遵循如下原则：

- 商数不论正负，统一向0取整。
- 余数正负与被除数相同。

例如：

```
OceanBase (root@oceanbase)> SELECT (-7) DIV (3.6), (-7) MOD (3.6);
+----------------+----------------+
| (-7) DIV (3.6) | (-7) MOD (3.6) |
+----------------+----------------+
|             -1 |           -3.4 |
+----------------+----------------+
1 row in set (0.01 sec)

OceanBase (root@oceanbase)> SELECT (-7) DIV (-3.4), (-7) % (-3.4);
+-----------------+---------------+
| (-7) DIV (-3.4) | (-7) % (-3.4) |
+-----------------+---------------+
|               2 |          -0.2 |
+-----------------+---------------+
1 row in set (0.02 sec)
```

5.2.2. 位运算符

位运算使用 `BIGINT UNSIGNED` 计算，不需考虑符号位。

以下为所有位运算符列表：

运算符	操作数	含义
&	二元	按位与。
|	二元	按位或。
~	一元	按位取反。
^	二元	按位异或。
<<	二元	逻辑左移。
>>	二元	逻辑右移。

5.2.3. 比较运算符

以下为所有比较运算符列表：

运算符	操作数	含义	NULL参与运算
=	二元	等于。	结果为NULL。
<> / !=	二元	不等于。	结果为NULL。
>	二元	大于。	结果为NULL。
>=	二元	大于等于。	结果为NULL。
<	二元	小于。	结果为NULL。
<=	二元	小于等于。	结果为NULL。
[NOT] IN	二元	是否在集合中。	详见说明。
[NOT] BETWEEN AND	三元	是否在区间内。	详见说明。
IS [NOT] TRUE	一元	是否等于TRUE。	结果为TRUE或FALSE。
IS [NOT] FALSE	一元	是否等于FALSE。	结果为TRUE或FALSE。
IS [NOT] NULL	一元	是否等于NULL。	结果为TRUE或FALSE。
<=>	二元	安全等于。	结果为TRUE或FALSE。

以下为部分运算符对NULL的特殊处理：

- value [NOT] IN () ：
  - value为NULL时，结果为NULL。
  - value不为NULL、且集合包含NULL时，如果集合中有非NULL值与value相等，则结果为TRUE，否则为NULL。
- value [NOT] BETWEEN lower AND upper ：

- value为NULL、或lower和upper都为NULL时，结果为NULL。
- value不为NULL、且只有lower/upper为NULL时，如果只通过value和lower/upper就能确定运算结果，则直接使用这个结果，否则返回NULL。

例如：

```
OceanBase (root@oceanbase)> SELECT 1 IN (1, NULL), 1 IN (2, NULL);
+---------------+---------------+
| 1 IN (1, NULL) | 1 IN (2, NULL) |
+---------------+---------------+
|             1 |          NULL |
+---------------+---------------+
1 row in set (0.01 sec)

OceanBase (root@oceanbase)> SELECT 1 BETWEEN 0 AND NULL, 1 BETWEEN 2 AND NULL;
+---------------------+---------------------+
| 1 BETWEEN 0 AND NULL | 1 BETWEEN 2 AND NULL |
+---------------------+---------------------+
|              NULL |                   0 |
+---------------------+---------------------+
1 row in set (0.01 sec)
```

5.2.4. 逻辑运算符

以下为所有逻辑运算符列表：

运算符	操作数	含义
AND / &&	二元	逻辑与
OR / ||	二元	逻辑或
NOT / !	一元	逻辑非

5.2.5. 日期时间运算符

以下为所有日期时间运算符列表：

运算符	操作数	含义
+	二元	同 DATE_ADD 函数
-	二元	同 DATE_SUB 函数

例如：

```
OceanBase (root@oceanbase)> SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
+---------------------------------------------+
| '2008-12-31 23:59:59' + INTERVAL 1 SECOND |
+---------------------------------------------+
| 2009-01-01 00:00:00                         |
+---------------------------------------------+
1 row in set (0.01 sec)
```

5.2.6. 字符连接运算符

以下为所有字符连接运算符列表：

运算符	操作数	含义
||	二元	字符串连接

如前节所述，|| 也可作为逻辑或运算符，两个不同含义是通过SQL MODE控制的：

- SQL MODE包含 PIPES_AS_CONCAT 时，|| 为字符连接运算符。
- SQL MODE不包含 PIPES_AS_CONCAT 时，|| 为逻辑与运算符。

5.2.7. 层次查询运算符

以下为所有层次查询运算符列表：

运算符	操作数	含义
PRIOR	一元	用于指明column来自parent row
CONNECT_BY_ROOT	一元	用于指明顶层祖先

5.2.8. 排序规则运算符

以下为所有排序规则运算符列表：

运算符	操作数	含义
COLLATE	一元（后缀）	排序规则

例如：

```
SELECT last_name
 FROM employees
ORDER BY last_name COLLATE UTF8MB4_GENERAL_CI;
```

5.3. 函数

5.3.1. 函数

日期时间函数

日期时间函数主要用来显示有关日期和时间的信息。

CURDATE

声明

```
CURDATE()
```

说明

返回当前日期，不含时间部分。

例子

```
OceanBase (root@test)> SELECT CURDATE();
+------------+
| CURDATE() |
+------------+
| 2018-05-05 |
+------------+
1 row in set (0.00 sec)
```

CURRENT_DATE

声明

```
CURRENT_DATE()
CURRENT_DATE
```

说明

同 `CURDATE()` 。

CURRENT_TIME

声明

```
CURRENT_TIME([scale])
```

说明

返回当前时间，不含日期部分。

`scale` 有效值为0到6，表示微秒部分精度，默认值为0。

例子

```
OceanBase (root@test)> SELECT CURRENT_TIME(6);
+-----------------+
| CURRENT_TIME(6) |
+-----------------+
| 11:11:45.215311 |
+-----------------+
1 row in set (0.01 sec)
```

CURRENT_TIMESTAMP

声明

```
CURRENT_TIMESTAMP([scale])
```

说明

返回当前日期时间，考虑时区设置。

`scale` 有效值为0到6，表示微秒部分精度，默认值为0。

例子

```
OceanBase (root@test)> SELECT CURRENT_TIMESTAMP(6);
+----------------------------+
| CURRENT_TIMESTAMP(6)       |
+----------------------------+
| 2018-05-05 11:35:39.177764 |
+----------------------------+
1 row in set (0.01 sec)
```

CURTIME

声明

```
CURTIME()
```

说明

同 `CURRENT_TIME()` 。

DATE_ADD

声明

```
DATE_ADD(date, INTERVAL expr unit)
```

说明

日期时间的算术计算。

- `date` 表示日期时间基准，日期部分必选，时间部分可选。
- `expr` 表示时间间隔，可以为负值。
- `unit` 表示时间间隔单位。

以下为所有时间间隔单位列表：

单位	类型	含义	格式
`MICROSECOND`	独立	微秒	`MICROSECONDS`
`SECOND`	独立	秒	`SECONDS`
`MINUTE`	独立	分钟	`MINUTES`
`HOUR`	独立	小时	`HOURS`

单位	类型	含义	格式
DAY	独立	天	DAYS
WEEK	独立	周	WEEKS
MONTH	独立	月	MONTHS
QUARTER	独立	季度	QUARTERS
YEAR	独立	年	YEARS
SECOND_MICROSECOND	组合	秒到微秒	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	组合	分钟到微秒	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	组合	分钟到秒	'MINUTES:SECONDS'
HOUR_MICROSECOND	组合	小时到微秒	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	组合	小时到秒	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	组合	小时到分钟	'HOURS:MINUTES'

单位	类型	含义	格式
DAY_SECOND	组合	天到秒	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	组合	天到分钟	'DDAYSD HOURS:MINUTES'
DAY_HOUR	组合	天到小时	'DAYS HOURS'
YEAR_MONTH	组合	年到月	'YEARS-MONTHS'

例子

```
OceanBase (root@test)> SELECT
  -> DATE_ADD(NOW(), INTERVAL 5 DAY),
  -> DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND),
  -> DATE_ADD('2014-01-10', INTERVAL 5 SECOND),
  -> DATE_ADD('2014-01-10', INTERVAL 5 MINUTE),
  -> DATE_ADD('2014-01-10', INTERVAL 5 HOUR),
  -> DATE_ADD('2014-01-10', INTERVAL 5 DAY),
  -> DATE_ADD('2014-01-10', INTERVAL 5 WEEK),
  -> DATE_ADD('2014-01-10', INTERVAL 5 MONTH),
  -> DATE_ADD('2014-01-10', INTERVAL 5 QUARTER),
  -> DATE_ADD('2014-01-10', INTERVAL 5 YEAR),
  -> DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND),
  -> DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND),
  -> DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND),
  -> DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND),
  -> DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND),
  -> DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE),
  -> DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND),
  -> DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND),
  -> DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE),
  -> DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR),
  -> DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH)
  -> \G
*************************** 1. row ***************************
```

```
*************************** 1. row ***************************
                    DATE_ADD(NOW(), INTERVAL 5 DAY): 2018-05-10 14:54:52
             DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND): 2014-01-10 00:00:00.000005
               DATE_ADD('2014-01-10', INTERVAL 5 SECOND): 2014-01-10 00:00:05
               DATE_ADD('2014-01-10', INTERVAL 5 MINUTE): 2014-01-10 00:05:00
                 DATE_ADD('2014-01-10', INTERVAL 5 HOUR): 2014-01-10 05:00:00
                  DATE_ADD('2014-01-10', INTERVAL 5 DAY): 2014-01-15
                 DATE_ADD('2014-01-10', INTERVAL 5 WEEK): 2014-02-14
                DATE_ADD('2014-01-10', INTERVAL 5 MONTH): 2014-06-10
              DATE_ADD('2014-01-10', INTERVAL 5 QUARTER): 2015-04-10
                 DATE_ADD('2014-01-10', INTERVAL 5 YEAR): 2019-01-10
  DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND): 2014-01-10 00:00:05.000005
DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND): 2014-01-10 00:05:05.000005
       DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND): 2014-01-10 00:05:05
DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND): 2014-01-10 05:05:05.000005
       DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND): 2014-01-10 05:05:05
       DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE): 2014-01-10 05:05:00
DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND): 2014-01-11 05:05:05.000005
      DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND): 2014-01-11 05:05:05
       DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE): 2014-01-11 05:05:00
         DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR): 2014-01-11 05:00:00
         DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH): 2015-02-10
1 row in set (0.01 sec)
```

DATE_FORMAT

声明

```
DATE_FORMAT(date, format)
```

说明

将日期时间以指定格式输出。

- `date` 表示日期时间。
- `format` 表示输出格式。

以下为所有输出格式列表：

格式符	含义	格式
%a	星期的英文缩写。	Sun..Sat

格式符	含义	格式
%b	月份的英文缩写。	Jan..Dec
%c	月份的数字形式。	1..12
%D	日的英文缩写。	1st..31st
%d	日的数字形式。	01..31
%e	日的数字形式。	1.. 31
%f	微秒。	000000..999999
%H	小时。	00..23
%h	小时。	01..12
%I	小时。	01..12
%i	分钟。	00..59
%j	一年中的第几天。	001..366
%k	小时。	0..23
%l	小时。	0..12
%M	月份的英文名称。	January..December
%m	月份的数字形式。	01..12

格式符	含义	格式
%p	上午或下午。	AM/PM
%r	12小时制时间。	hh:mm:ss AM/PM
%S	秒。	00..59
%s	秒。	00..59
%T	24小时制时间。	hh:mm:ss
%U	一年中的第几周，周日为每周第一天。	00..53
%u	一年中的第几周，周一为每周第一天。	00..53
%V	一年中的第几周，周日为每周第一天(和 %X 同时使用)。	01..53
%v	一年中的第几周，周一为每周第一天(和 %x 同时使用)。	01..53
%W	星期的英文名称。	Sunday..Saturday
%w	一周中的第几天。	0=Sunday..6=Saturday
%X	一周所属的年份，周日为每周第一天(和 %V 同时使用)。	
%x	一周所属的年份，周一为每周第一天(和 %v 同时使用)。	

格式符	含义	格式
%Y	四位年份。	
%y	两位年份。	
%%	文字字符，输出一个 % 。	

例子

```
OceanBase (root@test)> SELECT
  -> DATE_FORMAT('2014-01-01', '%Y-%M-%d'),
  -> DATE_FORMAT('2014-01-01', '%X-%V'),
  -> DATE_FORMAT('2014-01-01', '%U')
  -> \G
*************************** 1. row ***************************
DATE_FORMAT('2014-01-01', '%Y-%M-%d'): 2014-January-01
  DATE_FORMAT('2014-01-01', '%X-%V'): 2013-52
    DATE_FORMAT('2014-01-01', '%U'): 00
1 row in set (0.01 sec)
```

DATE_SUB

声明

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

说明

日期时间的算术计算。

请参考 DATE_ADD() 。

DATEDIFF

声明

```
DATEDIFF(date1, date2)
```

说明

返回 date1 和 date2 之间的天数

计算中只用到参数的日期部分，忽略时间部分。

例子

```
OceanBase (root@test)> SELECT DATEDIFF('2015-06-19','1994-12-17');
+-------------------------------------+
| DATEDIFF('2015-06-19','1994-12-17') |
+-------------------------------------+
|                                7489 |
+-------------------------------------+
1 row in set (0.01 sec)
```

EXTRACT

声明

```
EXTRACT(unit FROM date)
```

说明

以整数类型返回 `date` 的指定部分值。如果指定多个部分，则将所有值按顺序拼接。

`unit` 请参考 `DATE_ADD()` 。当 `unit` 为 `WEEK` 时，请参考 `DATE_FORMAT()` 的 `%U` 说明。

例子

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

FROM_DAYS

声明

```
FROM_DAYS(N)
```

说明

返回指定天数 `N` 对应的 `DATE` 值。天数指距离 `0000-01-01` 的天数。

例子

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

FROM_UNIXTIME

声明

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

说明

- 不指定 `format` 参数时，返回 `DATETIME` 类型值，不考虑时区。
- 指定 `format` 参数时，返回指定格式的日期时间字符串。

`unix_timestamp` 指UNIX时间戳，即距离 `1970-01-01 00:00:00.000000` 的微秒数。

`format` 支持的格式已在 `DATE_FORMAT()` **说明** 中列出。

例子

```
OceanBase (root@test)> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x');
+-----------------------------------------------------------+
| FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x') |
+-----------------------------------------------------------+
| 2018 5th May 08:41:26 2018                                |
+-----------------------------------------------------------+
1 row in set (0.01 sec)
```

MONTH

声明

```
MONTH(date)
```

说明

返回 `date` 的月份信息。

例子

```
OceanBase (root@test)> SELECT MONTH('2008-02-03');
+---------------------+
| MONTH('2008-02-03') |
+---------------------+
|                   2 |
+---------------------+
1 row in set (0.01 sec)
```

NOW

声明

```
NOW([scale])
```

说明

同 `CURRENT_TIMESTAMP()` 。

PERIOD_DIFF

声明

```
PERIOD_DIFF(p1, p2)
```

说明

以月份位单位返回两个日期之间的间隔。日期只能包含年月信息，格式只能为 `YYYYMM` 或 `YYMM` 。

例子

```
OceanBase (root@test)> SELECT PERIOD_DIFF(200802, 200703);
+----------------------------+
| PERIOD_DIFF(200802,200703) |
+----------------------------+
|                         11 |
+----------------------------+
1 row in set (0.01 sec)
```

STR_TO_DATE

声明

```
STR_TO_DATE(str, format)
```

说明

使用 format 将 str 转换为 DATETIME 值、 DATE 值、或 TIME 值。返回值类型取决于 format 包含日期时间的哪些部分。

format 支持的格式已在 DATE_FORMAT() 说明 中列出。

例子

```
OceanBase (root@test)> SELECT STR_TO_DATE('2014-Jan-1st 5:5:5 pm', '%Y-%b-%D %r');
+-----------------------------------------------------+
| STR_TO_DATE('2014-Jan-1st 5:5:5 pm', '%Y-%b-%D %r') |
+-----------------------------------------------------+
| 2014-01-01 05:05:05                                 |
+-----------------------------------------------------+
1 row in set (0.01 sec)
```

TIME

声明

```
TIME(datetime
```

说明

以 TIME 类型返回 datetime 的时间信息。

例子

```
OceanBase (root@test)> SELECT TIME('2003-12-31 01:02:03');
+-----------------------------+
| TIME('2003-12-31 01:02:03') |
+-----------------------------+
| 01:02:03.000000             |
+-----------------------------+
1 row in set (0.01 sec)
```

TIME_TO_USEC

声明

```
TIME_TO_USEC(date)
```

说明

将 date 值转换为距离 1970-01-01 00:00:00.000000 的微秒数，考虑时区信息。

date 可以表示日期，也可以表示日期时间。

例子

```
OceanBase (root@test)> SELECT TIME_TO_USEC('2014-03-25'), TIME_TO_USEC(NOW());
+----------------------------+---------------------+
| TIME_TO_USEC('2014-03-25') | TIME_TO_USEC(NOW()) |
+----------------------------+---------------------+
|           1395676800000000 |    1525528100000000 |
+----------------------------+---------------------+
1 row in set (0.01 sec)
```

TIMEDIFF

声明

```
TIMEDIFF(date1, date2)
```

说明

以 `TIME` 类型返回两个日期时间的时间间隔。

例子

```
OceanBase (root@test)> SELECT
    -> TIMEDIFF('2015-06-06 12:12:12', '2014-06-05 11:11:11'),
    -> TIMEDIFF('2015-06-06 12:12:12', '2015-06-05 11:11:11')
    -> \G
*************************** 1. row ***************************
TIMEDIFF('2015-06-06 12:12:12', '2014-06-05 11:11:11'): 838:59:59
TIMEDIFF('2015-06-06 12:12:12', '2015-06-05 11:11:11'): 25:01:01
1 row in set (0.00 sec)
```

TIMESTAMPDIFF

声明

```
TIMESTAMPDIFF(unit, date1, date2)
```

说明

以 `unit` 为单位返回两个日期时间的间隔。 `unit` 只能为 `DATE_ADD()` 中的独立单位。

例子

```
OceanBase (root@test)> SELECT
  -> TIMESTAMPDIFF(SECOND, NOW(), '2011-01-01 11:11:11'),
  -> TIMESTAMPDIFF(DAY, '2011-01-01 11:11:11', NOW())
  -> \G
*************************** 1. row ***************************
TIMESTAMPDIFF(SECOND, NOW(), '2011-01-01 11:11:11'): -231677498
  TIMESTAMPDIFF(DAY, '2011-01-01 11:11:11', NOW()): 2681
1 row in set (0.00 sec)
```

TIMESTAMPADD

声明

```
TIMESTAMPADD(unit, interval_expr, date)
```

说明

日期时间的算术计算。

同 `DATE_ADD()` ，但 `unit` 只能为独立单位。

例子

```
OceanBase (root@test)> SELECT
  -> TIMESTAMPADD(DAY, -5, '2010-01-01 00:00:00'),
  -> DATE_ADD('2010-01-01 00:00:00', INTERVAL -5 DAY)
  -> \G
*************************** 1. row ***************************
  TIMESTAMPADD(DAY, -5, '2010-01-01 00:00:00'): 2009-12-27 00:00:00
DATE_ADD('2010-01-01 00:00:00', INTERVAL -5 DAY): 2009-12-27 00:00:00
1 row in set (0.01 sec)
```

TO_DAYS

声明

```
TO_DAYS(date)
```

说明

返回指定 `date` 值对应的天数。天数指距离 `0000-01-01` 的天数。

例子

```
OceanBase (root@test)> SELECT TO_DAYS('2015-11-04'), TO_DAYS('20151104');
+----------------------+--------------------+
| TO_DAYS('2015-11-04') | TO_DAYS('20151104') |
+----------------------+--------------------+
|               736271 |             736271 |
+----------------------+--------------------+
1 row in set (0.01 sec)
```

USEC_TO_TIME

声明

```
USEC_TO_TIME(usec)
```

说明

将 `usec` 值转换为 `TIMESTAMP` 类型值。

`usec` 表示距离 `1970-01-01 00:00:00.000000` 的微秒数，考虑时区信息。

例子

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

UNIX_TIMESTAMP

声明

```
UNIX_TIMESTAMP()
UNIX_TIMESTAMP(date)
```

说明

- 不指定 `date` 参数时，返回当前时间距离'1970-01-01 00:00:00'的秒数，考虑时区。
- 指定 `date` 参数时，返回指定时间距离'1970-01-01 00:00:00'的秒数，考虑时区。

例子

```
OceanBase (root@test)> SELECT UNIX_TIMESTAMP(), TIME_TO_USEC(NOW());
+------------------+--------------------+
| UNIX_TIMESTAMP() | TIME_TO_USEC(NOW()) |
+------------------+--------------------+
|     1525570561 |   1525570561000000 |
+------------------+--------------------+
1 row in set (0.01 sec)

OceanBase (root@test)> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
+---------------------------------------+
| UNIX_TIMESTAMP('1997-10-04 22:23:00') |
+---------------------------------------+
|                             875974980 |
+---------------------------------------+
1 row in set (0.01 sec)
```

UTC_TIMESTAMP

声明

```
UTC_TIMESTAMP()
```

说明

返回当前UTC时间。

例子

```
OceanBase (root@test)> SELECT UTC_TIMESTAMP();
+---------------------+
| UTC_TIMESTAMP()     |
+---------------------+
| 2018-05-06 01:38:32 |
+---------------------+
1 row in set (0.01 sec)
```

YEAR

声明

```
YEAR(date)
```

说明

返回 `date` 值的年份信息。

例子

```
OceanBase (root@test)> SELECT YEAR('1987-01-01');
+--------------------+
| YEAR('1987-01-01') |
+--------------------+
|               1987 |
+--------------------+
1 row in set (0.00 sec)
```

字符串函数

CONCAT

声明

```
CONCAT(str1, .., strN)
```

说明

把多个字符串连接成一个字符串。参数中包含 `NULL` 时，返回 `NULL` 。

例子

```
OceanBase (root@test)> SELECT
    -> CONCAT('test','OceanBase', '1.0'),
    -> CONCAT('test','OceanBase', NULL)
    -> \G
*************************** 1. row ***************************
CONCAT('test','OceanBase', '1.0'): testOceanBase1.0
 CONCAT('test','OceanBase', NULL): NULL
1 row in set (0.01 sec)
```

CONCAT_WS

声明

```
CONCAT_WS(separator, str1, .., strN)
```

说明

把多个字符串连接成一个字符串，相邻字符串间使用 `separator` 分隔。参数中包含 `NULL` 时，将忽略 `NULL` 。

例子

```
OceanBase (root@test)> SELECT
  -> CONCAT_WS('_', 'First', 'Second'),
  -> CONCAT_WS('_', 'First', NULL, 'Second')
  -> \G
*************************** 1. row ***************************
   CONCAT_WS('_', 'First', 'Second'): First_Second
CONCAT_WS('_', 'First', NULL, 'Second'): First_Second
1 row in set (0.00 sec)
```

SUBSTR

声明

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

说明

返回 `str` 的子字符串，起始位置为 `pos` ，长度为 `len` 。参数中包含 `NULL` 时，返回 `NULL` 。

- 不指定 `len` 时，返回的子字符串从 `pos` 位置开始到 `str` 结尾。
- `pos` 值为负数时，从 `str` 尾部向头部倒序确定起始位置。
- `len` 小于等于0，或 `pos` 指定的起始位置非法时，返回空字符串。

例子

```
OceanBase (root@test)> SELECT
  -> SUBSTR('abcdefg', 3),
  -> SUBSTR('abcdefg', 3, 2),
  -> SUBSTR('abcdefg', -3),
  -> SUBSTR('abcdefg', 3, -2),
  -> SUBSTR('abcdefg' from -4 for 2)
  -> \G
*************************** 1. row ***************************
      SUBSTR('abcdefg', 3): cdefg
   SUBSTR('abcdefg', 3, 2): cd
    SUBSTR('abcdefg', -3): efg
  SUBSTR('abcdefg', 3, -2):
SUBSTR('abcdefg' from -4 for 2): de
1 row in set (0.01 sec)
```

SUBSTRING

声明

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

说明

同 `SUBSTR` 。

TRIM

声明

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

说明

删除字符串所有前缀和/或后缀，默认为 `BOTH` 。参数中包含 `NULL` 时，返回 `NULL` 。

例子

```
OceanBase (root@test)> SELECT
  -> TRIM(' bar '),
  -> TRIM(LEADING 'x' FROM 'xxxbarxxx'),
  -> TRIM(BOTH 'x' FROM 'xxxbarxxx'),
  -> TRIM(TRAILING 'x' FROM 'xxxbarxxx')
  -> \G
*************************** 1. row ***************************
           TRIM(' bar '): bar
TRIM(LEADING 'x' FROM 'xxxbarxxx'): barxxx
  TRIM(BOTH 'x' FROM 'xxxbarxxx'): bar
TRIM(TRAILING 'x' FROM 'xxxbarxxx'): xxxbar
1 row in set (0.01 sec)
```

LTRIM

声明

```
LTRIM(str)
```

说明

删除字符串左侧的空格。

RTRIM

声明

```
RTRIM(str)
```

说明

删除字符串右侧的空格。

ASCII

声明

```
ASCII(str)
```

说明

返回字符串最左侧字符的ASCII码。

ORD

声明

```
ORD(str)
```

说明

返回字符串最左侧字符的字符码。单字节编码如果最左侧字符是多字节编码字符，返回值计算规则如下：

```
(1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

例子

```
OceanBase (root@test)> SELECT ORD('中国');
+---------------+
| ORD('中国') |
+---------------+
|    14989485 |
+---------------+
1 row in set (0.01 sec)
```

LENGTH

声明

```
LENGTH(str)
```

说明

返回 `str` 的字节长度。

例子

```
OceanBase (root@test)> SELECT LENGTH('中国'), LENGTH('hello');
+------------------+-----------------+
| LENGTH('中国') | LENGTH('hello') |
+------------------+-----------------+
|          6 |          5 |
+------------------+-----------------+
1 row in set (0.01 sec)
```

CHAR_LENGTH

声明

```
CHAR_LENGTH(str)
```

说明

返回字符串包含的字符数。

例子

```
OceanBase (root@test)> SELECT CHAR_LENGTH('中国'), CHAR_LENGTH('hello');
+----------------------+----------------------+
| CHAR_LENGTH('中国') | CHAR_LENGTH('hello') |
+----------------------+----------------------+
|              2 |              5 |
+----------------------+----------------------+
1 row in set (0.00 sec)
```

UPPER

声明

```
UPPER(str)
```

说明

将字符串中的小写字母转化为大写字母。

例子

```
OceanBase (root@test)> SELECT UPPER('OceanBase您好！');
+-----------------------------+
| UPPER('OceanBase您好！') |
+-----------------------------+
| OCEANBASE您好！        |
+-----------------------------+
1 row in set (0.01 sec)
```

LOWER

声明

```
LOWER(str)
```

说明

将字符串中的大写字母转化为小写字母。

例子

```
OceanBase (root@test)> SELECT LOWER('OceanBase您好！');
+-----------------------------+
| LOWER('OceanBase您好！') |
+-----------------------------+
| oceanbase您好！        |
+-----------------------------+
1 row in set (0.01 sec)
```

HEX

声明

```
HEX(str)
```

说明

将数字或字符串转化为十六进制字符串。

例子

```
OceanBase (root@test)> SELECT HEX(255), HEX('abc');
+----------+------------+
| HEX(255) | HEX('abc') |
+----------+------------+
| FF       | 616263     |
+----------+------------+
1 row in set (0.00 sec)
```

UNHEX

声明

```
UNHEX(str)
```

说明

将十六进制字符串转化为正常字符串。

例子

```
OceanBase (root@test)> SELECT UNHEX('4f6365616e42617365');
+-----------------------------+
| UNHEX('4f6365616e42617365') |
+-----------------------------+
| OceanBase                   |
+-----------------------------+
1 row in set (0.00 sec)
```

MD5

声明

```
MD5(str)
```

说明

返回字符串的MD5值。

例子

```
OceanBase (root@test)> SELECT MD5(1);
+----------------------------------+
| MD5(1)                           |
+----------------------------------+
| c4ca4238a0b923820dcc509a6f75849b |
+----------------------------------+
1 row in set (0.00 sec)
```

INT2IP

声明

```
INT2IP(int_value)
```

说明

将整数内码转换成IP地址。

例子

```
OceanBase (root@test)> SELECT
    -> INT2IP(16777216),
    -> HEX(16777216),
    -> INT2IP(1)
    -> \G
*************************** 1. row ***************************
INT2IP(16777216): 1.0.0.0
  HEX(16777216): 1000000
     INT2IP(1): 0.0.0.1
1 row in set (0.01 sec)
```

IP2INT

声明

```
IP2INT('ip_addr')
```

说明

将IP地址转换成整数内码。

例子

```
OceanBase (root@test)> SELECT
  -> IP2INT('0.0.0.1'),
  -> HEX(IP2INT('0.0.0.1')),
  -> HEX(IP2INT('1.0.0.0'))
  -> \G
*************************** 1. row ***************************
  IP2INT('0.0.0.1'): 1
HEX(IP2INT('0.0.0.1')): 1
HEX(IP2INT('1.0.0.0')): 1000000
1 row in set (0.01 sec)
```

LIKE

声明

```
str1 [NOT] LIKE str2 [ESCAPE str3]
```

说明

字符串通配符匹配。参数中包含 `NULL` 时，返回 `NULL` 。

通配符包括：

- `%` ：匹配任意长度字符串。
- `_` ：匹配单个字符。

ESCAPE用于定义转义符，如果str2中包含str3，在匹配时str3后的字符将做为普通字符处理。

例子

```
OceanBase (root@test)> SELECT 'ab%' LIKE 'abc%' ESCAPE 'c';
+-----------------------------+
| 'ab%' LIKE 'abc%' ESCAPE 'c' |
+-----------------------------+
|                           1 |
+-----------------------------+
1 row in set (0.01 sec)
```

REGEXP

声明

```
str [NOT] REGEXP | RLIKE pat
```

说明

正则匹配。参数中包含 NULL 时，返回 NULL 。

例子

```
OceanBase (root@test)> SELECT
  -> 1234 REGEXP 1,
  -> 'hello' RLIKE 'h%'
  -> \G
*************************** 1. row ***************************
  1234 REGEXP 1: 1
'hello' RLIKE 'h%': 0
1 row in set (0.01 sec)
```

REPEAT

声明

```
REPEAT(str, count)
```

说明

返回 str 重复 count 次组成的字符串。 count 小于等于0时，返回空字符串。参数中包含 NULL 时，返回 NULL 。

例子

```
OceanBase (root@test)> SELECT
  -> REPEAT('1', -1),
  -> REPEAT(null, null),
  -> REPEAT('Abc', 4)
  -> \G
*************************** 1. row ***************************
 REPEAT('1', -1):
REPEAT(null, null): NULL
 REPEAT('Abc', 4): AbcAbcAbcAbc
1 row in set (0.01 sec)
```

SPACE

声明

```
SPACE(N)
```

说明

返回包含 N 个空格的字符串。

SUBSTRING_INDEX

声明

```
SUBSTRING_INDEX(str, delim, count)
```

说明

在定界符 delim 以及 count 出现前，从字符串 str返回字符串。若 count为正值，则返回最终定界符(从左边开始)左边的一切内容。若 count为负值，则返回定界符（从右边开始）右边的一切内容。任意一个参数为NULL，返回NULL；若 str 或 delim 为空字符串，则返回空串；若 count= 0, 返回空串。

str, delim, count 参数支持数值与字符串的隐式转换。

例子

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

LOCATE

声明

```
LOCATE(substr,str) , LOCATE(substr,str,pos)
```

说明

第一个语法返回字符串 str 中子字符串 substr 的第一个出现位置。第二个语法返回字符串 str 中子字符串 substr 的第一个出现位置, 起始位置在 pos。如若 substr 不在 str 中，则返回值为0。

例子

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

POSITION

声明

```
POSITION(substr IN str)
```

说明

与LOCATE同义。

INSTR

声明

```
INSTR(str,substr)
```

说明

返回字符串 str 中子字符串的第一个出现位置。这和 LOCATE() 的双参数形式相同，除非参数的顺序被颠倒。

例子

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

REPLACE

声明

```
REPLACE(str, from_str, to_str)
```

说明

返回字符串 str 以及所有被字符 to_str 替代的字符串 from_str。

例子

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

FIELD

声明

```
FIELD(str,str1,str2,str3,…)
```

说明

返回参数 str 在 str1, str2, str3,... 列表中的索引位置（从1开始的位置）。在找不到 str 的情况下，返回值为 0。

如果所有的对于 FIELD() 的参数均为字符串，则所有参数均按照字符串进行比较。如果所有的参数均为数字，则按照数字进行比较。否则，参数均按照 double 类型进行比较。

如果 str 为 NULL，则返回值为 0，原因是 NULL 不能同任何值进行同等比较。FILED() 是 ELT() 的补数。

例子

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

ELT

声明

```
ELT(N, str1, str2, str3,⋯)
```

说明

若 N=1，则返回值为 str1, 若 N=2，则返回值为 str2，以此类推。若 N 小于 1 或大于参数的数目，则返回值为NULL。ELT() 是 FIELD() 的补数。

例子

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

INSERT

声明

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

说明

返回字符串 str1, 字符串中起始于 pos 位置，长度为 len 的子字符串将被 str2 取代。如果 pos 超过字符串长度，则返回值为原始字符串。 假如 len 的长度大于其它字符串的长度，则从位置 pos 开始替换。若任何一个参数为 null，则返回值为 NULL。这个函数支持多字节字元。

- *str1 和 str2 必须是字符串，pos 和 len 必须是整数。任意参数为 NULL，结果总为 NULL;*

- *str1 和 str2 中的文字符被当做字节流看待；*
- *pos 值为负数或者大于 str1 长度时，返回 str1；*
- 当 len 小于 0，或大于 str 的长度时，返回结果为 str1 从开头到 pos 位置的串，和 str2 的组合串。

例子

```
Oceanbase>select insert('Quadratic',-2,100,'What'), insert('Quadratic',7,3,'What'),
  -> insert('Quadratic',-1,3,'What'), insert('Quadratic',10,3,'What'), insert('Quadratic',5,-1,''),
  -> insert('Quadratic',7,-1,'What')\G;
* 1. row *
insert('Quadratic',-2,100,'What'): Quadratic
 insert('Quadratic',7,3,'What'): QuadraWhat
insert('Quadratic',-1,3,'What'): Quadratic
insert('Quadratic',10,3,'What'): Quadratic
   insert('Quadratic',5,-1,''): Quad
insert('Quadratic',7,-1,'What'): QuadraWhat
1 row in set (0.01 sec)
```

LPAD

声明

```
LPAD(str, len, padstr)
```

说明

用指定字符串padstr，在左侧填充字符串str到指定长度len。如果str长度大于len，则str被截断。

例子

```
OceanBase > SELECT LPAD('hi',4,'??');
+------------------+
| LPAD('hi',4,'??') |
+------------------+
| ??hi         |
+------------------+
1 row in set (0.01 sec)

OceanBase > SELECT LPAD('hi',1,'??');
+------------------+
| LPAD('hi',1,'??') |
+------------------+
| h            |
+------------------+
1 row in set (0.00 sec)
```

RPAD

声明

```
RPAD(str, len, padstr)
```

说明

用指定字符串padstr，在右侧填充字符串str到指定长度len。如果str长度大于len，则str被截断。

例子

```
OceanBase (root@test)> SELECT RPAD('hi',4,'??');
+------------------+
| RPAD('hi',4,'??') |
+------------------+
| hi??             |
+------------------+
1 row in set (0.00 sec)

OceanBase (root@test)> SELECT RPAD('hi',1,'??');
+------------------+
| RPAD('hi',1,'??') |
+------------------+
| h                |
+------------------+
1 row in set (0.00 sec)
```

UUID

声明

```
uuid()
```

说明

生成一个全局唯一ID。

例子

```
OceanBase (root@test)> select uuid();
+--------------------------------------+
| uuid()                               |
+--------------------------------------+
| f756a1f6-4de6-11e8-90af-90b11c53e421 |
+--------------------------------------+
1 row in set (0.00 sec)
```

BIN

声明

`bin(N)`

说明

返回数字N的二进制形式。

例子

```
OceanBase > SELECT BIN(12);
+---------+
| BIN(12) |
+---------+
| 1100    |
+---------+
1 row in set (0.00 sec)
```

QUOTE

声明

`quote(str)`

说明

引用一个字符串以产生一个结果可以作为SQL语句中正确地转义数据值。返回的字符串由单引号和单引号的各个实例封闭(''), 反斜线(''), ASCII NUL 和 Control-Z前面加一个反斜杠。如果参数为NULL，则返回值是词 'NULL' 不包围单引号。

例子

```
OceanBase > SELECT QUOTE('Don\'t!');
+------------------+
| QUOTE('Don\'t!') |
+------------------+
| 'Don\'t!'      |
+------------------+
1 row in set (0.00 sec)

OceanBase > SELECT QUOTE(NULL);
+-------------+
| QUOTE(NULL) |
+-------------+
| NULL      |
+-------------+
1 row in set (0.00 sec)
```

REGEXP_SUBSTR

声明

```
regexp_substr(str,pattern,[position[,occurrence[,match_param[,subexpr]]]])
```

说明

在str中搜索匹配正则表达式pattern的子串,子串不存在返回NULL。这个函数支持多字节字元。除match_param外，其他任意参数为NULL，结果为NULL。

- str 待搜索的字符串，支持多字节字元。
- pattern 正则表达式，正则表达式规则兼容mysql。
- position [可选] 搜索的开始位置。必须是大于0的正整数，小于等于0报错，输入为NULL返回NULL，默认值为1，表示从第一个字符开始搜索。
- occurrence [可选] 返回第几次匹配的值。必须是大于等于0的正整数，小于等于0报错，输入为NULL返回NULL，默认值是1，表示返回第一次匹配的结果。
- match_param [可选] 搜索参数，字符串类型，目前仅支持两个字符 ‘i’,‘c’，i: 代表忽略大小写 c: 代表不忽略，其他字符报错。默认值根据str字符集的collection决定。 为NULL时使用默认值。
- subexpr [可选] 返回正则表达式中第几个组的值。大于等于0的整数，小于0则报错。默认值是0，表示返回满足整个pattern的子串。

例子

```
oceanbase> select regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 1) from dual;
+-----------------------------------------------------------------------------+
| regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 1) |
+-----------------------------------------------------------------------------+
| have                                                                        |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)

oceanbase> select regexp_substr('foothebar', 'foo(.*)(bar)', 1, 1, 'c', 1) from dual;
+-----------------------------------------------------------+
| regexp_substr('foothebar', 'foo(.*)(bar)', 1, 1, 'c', 1) |
+-----------------------------------------------------------+
| the                                                       |
+-----------------------------------------------------------+
1 row in set (0.01 sec)
```

转换函数

CAST

声明

```
CAST(expr AS type)
```

说明

将某种数据类型的表达式显式转换为另一种数据类型。

将 expr 字段值转换为 type 数据类型。

参数说明：

- expr: 表示任何有效的SQL表达式。
- AS：用于分隔两个参数，在AS之前的是要处理的数据，在AS之后是要转换的数据类型。
- type：表示目标系统所提供的数据类型。可以是以下值其中的一个：
  - DATE
  - DATETIME
  - DECIMAL
  - SIGNED [INTEGER]
  - TIME
  - UNSIGNED [INTEGER]

在使用CAST函数进行数据类型转换时，在下列情况下能够被接受：

- 两个表达式的数据类型完全相同；
- 两个表达式可隐式转换；
- 必须显式转换数据类型。

如果试图进行不可能的转换，OceanBase将显示一条错误信息。

如果转换时没有指定数据类型的长度，则使用 OceanBase系统内部最大长度。如varchar是262,143字节，number是65个bit位的浮动精度。

支持带符号和无符号的64比特值的运算。若你正在使用数字操作符 (如 +) 而其中一个操作数为无符号整数，则结果为无符号。可使用SIGNED 和UNSIGNED cast操作符来覆盖它。将运算分别派给带符号或无符号64比特整数。

假如任意一个操作数为一个浮点值，则结果为一个浮点值。

例子

```
Oceanbase>SELECT CAST(123 AS BOOL);
+------------------+
| CAST(123 AS bool) |
+------------------+
|                1 |
+------------------+
1 row in set (0.00 sec)

Oceanbase>select cast(1-2 as unsigned), cast(cast(1-2 as unsigned) as signed);
+----------------------+---------------------------------------+
| cast(1-2 as unsigned) | cast(cast(1-2 as unsigned) as signed) |
+----------------------+---------------------------------------+
| 18446744073709551615 |                                    -1 |
+----------------------+---------------------------------------+
1 row in set (0.00 sec)

Oceanbase>SELECT CAST(1 AS UNSIGNED) - 2.0;
+---------------------------+
| CAST(1 AS UNSIGNED) - 2.0 |
+---------------------------+
|                      -1.0 |
+---------------------------+
1 row in set (0.00 sec)

Oceanbase>select cast(0 as date);
+-----------------+
| cast(0 as date) |
+-----------------+
| 0000-00-00      |
+-----------------+
1 row in set (0.00 sec)
```

数学函数

ROUND

声明

`ROUND(X), ROUND(X,D)`

说明

返回一个数值，四舍五入到指定的长度或精度。

返回参数X, 其值接近于最近似的整数。在有两个参数的情况下，返回X，其值保留到小数点后D位，而第D位的保留方式为四舍五入。若要接保留X值小数点左边的D位，可将D设为负值。

返回值的类型同第一个自变量相同(假设它是一个整数、双精度数或小数)。这意味着对于一个整数参数，结果也是一个整数(无小数部分)。

- 对于准确值数字,ROUND()使用“四舍五入” 或“舍入成最接近的数” 的规则：对于一个分数部分为 .5或大于 .5的值，正数则上舍入到邻近的整数值，负数则下舍入临近的整数值。(换言之, 其舍入的方向是数轴上远离零的方向）。对于一个分数部分小于.5 的值，正数则下舍入下一个整数值，负数则下舍入邻近的整数值，而正数则上舍入邻近的整数值。
- 对于近似值数字， ROUND()遵循银行家规则“四舍--大于五入--五取最接近的偶数”的规则： 一个带有任何小数部分的值会被舍入成最接近的偶数整数。

例子

```
Oceanbase>select round(2.15,2);
+---------------+
| round(2.15,2) |
+---------------+
|          2.15 |
+---------------+
1 row in set (0.00 sec)

Oceanbase>select round(2555e-2,1);
+-------------------+
| round(2555e-2,1) |
+-------------------+
|             25.6 |
+-------------------+
1 row in set (0.01 sec)

Oceanbase>select round(25e-1), round(25.3e-1),round(35e-1);
+--------------+----------------+--------------+
| round(25e-1) | round(25.3e-1) | round(35e-1) |
+--------------+----------------+--------------+
|            3 |              3 |            4 |
+--------------+----------------+--------------+
1 row in set (0.00 sec)
```

CEIL

声明

```
CEIL(expr)
```

说明

返回大于或者等于指定表达式的最小整数。

还支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE);

如果输入NULL，返回值为NULL。

如果输入纯数字的字符串，支持自动转换成数字类型。

返回值会被转化为一个BIGINT。

例子

```
Oceanbase>select ceil(1.2), ceil(-1.2), ceil(1+1.5), ceil(1=1),ceil(1<1),ceil(null);
+-----------+-----------+-----------+----------+----------+-----------+
| ceil(1.2) | ceil(-1.2) | ceil(1+1.5) | ceil(1=1) | ceil(1<1) | ceil(null) |
+-----------+-----------+-----------+----------+----------+-----------+
|      2 |     -1 |       3 |      1 |      0 |     NULL |
+-----------+-----------+-----------+----------+----------+-----------+
1 row in set (0.00 sec)

Oceanbase>select ceil(name);
ERROR 1166 (42703): Unkown column name 'name'

Oceanbase>select ceil('2');
+-----------+
| ceil('2') |
+-----------+
|      2 |
+-----------+
1 row in set (0.00 sec)
```

FLOOR

声明

```
FLOOR(expr)
```

说明

和CEIL(expr)函数功能类似，返回小于或者等于指定表达式的最大整数。

还支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE);

如果输入NULL，返回值为NULL。

如果输入纯数字的字符串，支持自动转换成数字类型。

返回值会被转化为一个BIGINT。

例子

```
Oceanbase>select floor(1.2), floor(-1.2), floor(1+1.5), floor(1=1),floor(1<1),floor(null);
+------------+-------------+--------------+------------+------------+-------------+
| floor(1.2) | floor(-1.2) | floor(1+1.5) | floor(1=1) | floor(1<1) | floor(null) |
+------------+-------------+--------------+------------+------------+-------------+
|          1 |          -2 |            2 |          1 |          0 |        NULL |
+------------+-------------+--------------+------------+------------+-------------+
1 row in set (0.00 sec)

Oceanbase>select floor(name);
ERROR 1166 (42703): Unkown column name 'name'

Oceanbase>select floor('2');
+------------+
| floor('2') |
+------------+
|          2 |
+------------+
1 row in set (0.00 sec)
```

ABS

声明

```
ABS(expr)
```

说明

绝对值函数，求表达式绝对值，函数返回值类型与数值表达式的数据类型相同。

还支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE)；

如果输入NULL，返回值为NULL。

如果输入纯数字的字符串，支持自动转换成数字类型。

返回值会被转化为一个BIGINT。

例子

```
Oceanbase>select abs(5), abs(-5.777), abs(0), abs(1/2), abs(1-5);
+--------+-------------+--------+----------+----------+
| abs(5) | abs(-5.777) | abs(0) | abs(1/2) | abs(1-5) |
+--------+-------------+--------+----------+----------+
|      5 |       5.777 |      0 |   0.5000 |        4 |
+--------+-------------+--------+----------+----------+
1 row in set (0.00 sec)
```

NEG

声明

```
NEG(expr)
```

说明

求补函数，对操作数执行求补运算：用零减去操作数，然后结果返回操作数。

支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE)，再对结果求补。

例子

```
Oceanbase>select neg(1), neg(1+1), neg(2*3), neg(1=1), neg(5<1);
+--------+----------+----------+----------+----------+
| neg(1) | neg(1+1) | neg(2*3) | neg(1=1) | neg(5<1) |
+--------+----------+----------+----------+----------+
|   -1 |    -2 |    -6 |    -1 |     0 |
+--------+----------+----------+----------+----------+
1 row in set (0.01 sec)
```

SIGN

声明

```
SIGN(X)
```

说明

SIGN(X)返回参数作为-1、 0或1的符号，该符号取决于X的值为负、零或正。

支持比较运算，结果为BOOL值，被转化为数字类型处理，产生的结果为1(TRUE)、0 (FALSE);

如果输入NULL，返回值为NULL。

支持浮点数、十六进制数。

例子

```
Oceanbase>SELECT SIGN(-32), SIGN(0), SIGN(234);
+-----------+---------+-----------+
| SIGN(-32) | SIGN(0) | SIGN(234) |
+-----------+---------+-----------+
|        -1 |       0 |         1 |
+-----------+---------+-----------+
1 row in set (0.01 sec)

Oceanbase>select sign(null),sign(false),sign(0x01);
+------------+-------------+------------+
| sign(null) | sign(false) | sign(0x01) |
+------------+-------------+------------+
|       NULL |           0 |          1 |
+------------+-------------+------------+
1 row in set (0.00 sec)
```

CONV

声明

```
CONV(N, from_base, to_base)
```

说明

不同数基间转换数字。返回值为一个字符串，由 from_base 基转化为 to_base 基。输入参数N可以是一个整数或字符串。最小基数为 2，而最大基数则为 36。如果 to_base 是一个负数，则N被看作一个带符号数。否则，N 被看作无符号数。from_base 如果是负数，则被当作整数处理，符号被忽略。N 参数仅支持 int 类型和字符串类型输入；from_base 和 to_base 参数仅支持十进制int类型输入，且取值范围为 [-36, -2] U [2, 36]。

非法输入将导致报错，其中非法输入包括以下情况：

- *from_base 或者 to_base 不是一个合法的十进制int类型输入；*
- *from_base 或者 to_base 超出 [-36,-2] U [2,36] 的取值范围；*
- N 不是一个合法的数字表示，例如取值超出 0 ~ 9，a ~ z，A ~ Z 的字符范围；
- N 超出了 from_base 基的取值范围，例如 from_base 为 2，而 N 取值为 3；
- N 的取值超出了 int64 的最大表示范围，即 [-9223372036854775807, 9223372036854775807]。

例子

```
Oceanbase>select conv(9223372036854775807,10,2);
+----------------------------------------------------------------+
| conv(9223372036854775807,10,2)                                 |
+----------------------------------------------------------------+
| 1111111111111111111111111111111111111111111111111111111111111111 |
+----------------------------------------------------------------+
1 row in set (0.00 sec)

Oceanbase>select conv('-acc',21,-7);
+--------------------+
| conv('-acc',21,-7) |
+--------------------+
| -16425             |
+--------------------+
1 row in set (0.00 sec)
```

MOD

声明

```
MOD(N,M)
```

说明

取余函数。MOD(N，M)， N % M， N MOD M 三种形式是等效的。

MOD()对于带有小数部分的数值也起作用，它返回除法运算后的精确余数。

N,M 中任何一个参数为 NULL，返回值都为 NULL。M 为 0 是，也返回 NULL。

例子

```
Oceanbase>select mod(29,19), 29 mod 19, 29 % 19;
+------------+-----------+---------+
| mod(29,19) | 29 mod 19 | 29 % 19 |
+------------+-----------+---------+
|         10 |        10 |      10 |
+------------+-----------+---------+
1 row in set (0.00 sec)
```

POW

声明

```
POW(X,Y)
```

说明

返回X的Y次方。

X与Y中任何一个参数为NULL，返回值都为NULL。

例子

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

POWER

声明

```
POWER(X,Y)
```

说明

POWER(X,Y)和POW(X,Y)是同义函数。

RAND

声明

```
rand([N])
```

说明

RAND([N]) 函数接受0个或者1个参数（N被称为随机数种子），返回一个范围是[0,1.0)的随机浮点数。如果想获得一个范围在 [i, j) 的随机整数，可以使用表达式 FLOOR(I + RAND() * (j - i))。

如果参数N不指定，那么在执行之前会使用随机种子初始化，之后RAND()会根据这个初始化值来生成随机数，所以RAND()每次会生成不同的随机数序列。

如果参数N指定，那么N会作为随机数种子来生成随机数。根据N是否是常量，分成以下两种情况：

- 如果N是常量，那么N在执行之前会被当成随机种子初始化，之后RAND(N)会根据这个初始化值来生成随机数。相同的N值会生成相同的随机数序列。
- 如果N是变量(比如N是一个列值)，那么N会在每次执行的时候作为随机数种子来生成随机数。相同的N值会生成相同的随机数。

除了出现在Select语句中，RAND([N])也可以出现在where，order by和group by语句中，它的执行方式还是按照上述规则执行。比如，如果想随机的对一个表进行排序，可以使用如下SQL：select from t1 order by rand()。如果想随机取样一个表的100行，可以使用如下SQL：select from t1 order by rand() limit 100。

例子

```
mysql> select a, b, rand() from t3;
+------+------+---------------------+
| a    | b    | rand()              |
+------+------+---------------------+
|    1 |    1 | 0.641815407799385 |
|    2 |    2 | 0.16825051248841966 |
```

```
| 3 | 3 | 0.9158063697775886 |
+------+------+--------------------+
3 rows in set (0.00 sec)

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

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

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

mysql> select a, b, rand(a), rand(b) from t3;
+------+------+--------------------+--------------------+
| a | b | rand(a) | rand(b) |
+------+------+--------------------+--------------------+
| 1 | 1 | 0.40540353712197724 | 0.40540353712197724 |
| 2 | 2 | 0.6555866465490187 | 0.6555866465490187 |
```

```
|  3|  3| 0.9057697559760601| 0.9057697559760601|
+------+------+--------------------+--------------------+
3 rows in set (0.00 sec)
```

比较函数

GREATEST

声明

```
GREATEST(value1, ...)
```

说明

返回参数的最大值，和函数LEAST()相对。

参数至少为两个（一个参数将会报错）；如果参数中有NULL, 返回值为NULL。

当参数中同时存在数值和字符时，把字符隐式转换为数值类型处理，不能转换的报错。

例子

```
Oceanbase>select greatest(2,1), greatest('2',1,0), greatest('a','b','c'), greatest('a', NULL, 'c'), greatest('2014-05-15','2014-06-01')\G
  * 1. row *
                 greatest(2,1): 2
               greatest('2',1,0): 2
            greatest('a','b','c'): c
          greatest('a', NULL, 'c'): NULL
  greatest('2014-05-15','2014-06-01'): 2014-06-01
  1 row in set (0.01 sec)
  Oceanbase>select greatest(2);
  ERROR 1582 (42000): Incorrect parameter count in the call to native function 'greatest'
```

LEAST

声明

```
LEAST(value1, ...)
```

说明

返回参数的最小值，和函数GREATEST()相对。

参数至少为两个；如果参数中有NULL，返回值为NULL。

当参数中同时存在数值和字符时，把字符隐式转换为数值类型处理，不能转换的报错。

例子

```
Oceanbase>select least(2, null), least('2',4,9), least('a','b','c'), least('a',NULL,'c'), least('2014-05-15','2014-06-0
1')\G;
* 1. row *
              least(2, null): NULL
              least('2',4,9): 2
           least('a','b','c'): a
          least('a',NULL,'c'): NULL
least('2014-05-15','2014-06-01'): 2014-05-15
1 row in set (0.01 sec)
Oceanbase>select least(2);
ERROR 1582 (42000): Incorrect parameter count in the call to native function 'least'
```

ISNULL

声明

```
ISNULL(expr)
```

说明

如果参数 expr 为 NULL，那么 ISNULL() 的返回值为 1，否则范围值为 0。

ISNULL() 函数可以用来替代针对NULL的等值（=）比较。（使用=的NULL值比较通常都是错误的。）ISNULL() 函数同 IS NULL 比较操作符具有一些相同的特性。

例子

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

Oceanbase>SELECT ISNULL(null+1);
+----------------+
| ISNULL(null+1) |
+----------------+
|              1 |
+----------------+
1 row in set (0.00 sec)
```

流程控制函数

CASE

声明

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

说明

在第一个方案的返回结果中，value = compare-value。而第二个方案的返回结果是第一种条件为真。如果没有匹配的结果值，则返回结果为 ELSE 后的结果，如果没有 ELSE 部分，则返回值为 NULL。

例子

```
Oceanbase>select CASE 'b' when 'a' then 1 when 'b' then 2 END;
+-----------------------------------------------+
| CASE 'b' when 'a' then 1 when 'b' then 2 END |
+-----------------------------------------------+
|                                             2 |
+-----------------------------------------------+
1 row in set (0.01 sec)

Oceanbase>select CASE concat('a','b') when concat('ab','') then 'a' when 'b' then 'b' end;
+-------------------------------------------------------------------------------+
| CASE concat('a','b') when concat('ab','') then 'a' when 'b' then 'b' end |
+-------------------------------------------------------------------------------+
| a                                                                             |
+-------------------------------------------------------------------------------+
1 row in set (0.01 sec)

Oceanbase>select case when 1>0 then 'true' else 'false' end;
+-------------------------------------------+
| case when 1>0 then 'true' else 'false' end |
+-------------------------------------------+
| true                                      |
+-------------------------------------------+
1 row in set (0.00 sec)
```

IF

声明

```
IF(expr1,expr2,expr3)
```

说明

如果 expr1 的值为 TRUE（即：expr1<>0 且 expr1<>NULL），返回结果为 expr2; 否则返回结果为 expr3。

IF() 返回结果可以是数值或字符串类型，它取决于使用的内容。

如果 expr2, expr3 中只有一个明确是 NULL，则 IF() 函数的结果类型为非 NULL 表达式的结果类型。

例子

```
Oceanbase>select if(5>6, 'T','F'), if (5>6, 1, 0), if(null, 'True', 'False'), if(0, 'True', 'False')\G
* 1. row *
      if(5>6, 'T','F'): F
       if (5>6, 1, 0): 0
if(null, 'True', 'False'): False
  if(0, 'True', 'False'): False
1 row in set (0.01 sec)
```

IFNULL

声明

```
IFNULL(expr1, expr2)
```

说明

假设 expr1 不为 NULL，则 IFNULL() 的返回值为 expr1；否则其返回值为 expr2。IFNULL() 的返回值是数值或字符串，具体情况取决于其所使用的语境。

IF()的默认返回值类型按照以下方式计算：

表达式	返回值
expr1 或 expr2 返回值为一个字符串。	字符串
expr1 或 expr2 返回值为一个浮点值。	浮点
expr1 或 expr2 返回值为一个整数。	整数

假如 expr1 和 expr2 都是字符串，且其中任何一个字符串区分大小写，则返回结果是区分大小写。

例子

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

NULLIF

声明

```
NULLIF(expr1, expr2)
```

说明

如果 expr1 = expr2 成立，那么返回值为 NULL，否则返回值为 expr1。这和 CASE WHEN *expr1 = expr2 THEN NULL ELSE expr1 END相同。注意，如果参数不相等，则两次求得的值为 expr1。*

例子

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

ORA_DECODE

声明

```
ora_decode(条件, 值1, 返回值1, 值2, 返回值2, ... 值n, 返回值n, 缺省值)
```

说明

ORA_DECODE()函数功能等同于Oracle的DECODE()函数功能（OceanBase 1.0兼容MySQL，同时也增加了部分Oracle的函数，这部分Oracle同功能的函数在OceanBase 1.0中被统一命名以ORA_开头)。

该函数的含义如下:

```
IF 条件=值1
THEN RETURN(返回值1)
ELSIF 条件=值2
THEN RETURN(返回值2)
......
ELSIF 条件=值n
THEN RETURN(返回值n)
ELSE RETURN(缺省值)
END IF
```

5.3.2. 聚集函数

聚合函数对一组值执行计算并返回单一的值。聚合函数忽略空值。聚合函数经常与SELECT语句的GROUP BY子句一同使用。

所有聚合函数都具有确定性。任何时候用一组给定的输入值调用它们时，都返回相同的值。

在OceanBase的聚合函数中，Value表达式只能出现一个。例如：不支持COUNT(c1, c2)，仅支持COUNT(c1）。

AVG

声明

```
AVG(([DISTINCT] expr)
```

说明

返回指定组中的平均值，空值被忽略。DISTINCT选项可用于返回expr的不同值的平均值。若找不到匹配的行，则AVG()返回NULL。

例子

```
Oceanbase>select * from oceanbasetest;
+----+------+------+
| id | ip   | ip2 |
+----+------+------+
| 1 |   4 | NULL |
| 3 |   3 | NULL |
| 4 |   3 | NULL |
+----+------+------+
3 rows in set (0.01 sec)

Oceanbase>select avg(ip2), avg(ip), avg(distinct(ip)) from oceanbasetest;
+----------+---------+-------------------+
| avg(ip2) | avg(ip) | avg(distinct(ip)) |
+----------+---------+-------------------+
|    NULL | 3.3333 |         3.5000 |
+----------+---------+-------------------+
1 row in set (0.00 sec)

Oceanbase>select avg(distinct(ip)),avg(ip),avg(ip2) from oceanbasetest;
+-------------------+---------+----------+
| avg(distinct(ip)) | avg(ip) | avg(ip2) |
+-------------------+---------+----------+
|         3.5000 | 3.3333 |    NULL |
+-------------------+---------+----------+
1 row in set (0.00 sec)
```

COUNT

声明

```
COUNT([DISTINCT] expr)
```

说明

COUNT([DISTINCT] expr)返回SELECT语句检索到的行中非NULL值的数目。若找不到匹配的行，则COUNT()返回0。DISTINCT选项可用于返回 expr 的不同值的数目。

COUNT(*)的稍微不同之处在于，它返回检索行的数目，不论其是否包含NULL值。

例子

```
Oceanbase>select * from oceanbasetest;
+----+------+------+
| id | ip   | ip2 |
+----+------+------+
| 1 |   4 | NULL |
| 3 |   3 | NULL |
| 4 |   3 | NULL |
+----+------+------+
3 rows in set (0.00 sec)

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

MAX

声明

```
MAX([DISTINCT] expr)
```

说明

返回指定数据中的最大值。

MAX()的取值可以是一个字符串参数；在这些情况下，它们返回最大字符串值。DISTINCT关键字可以被用来查找 expr的不同值的最大值，这产生的结果与省略DISTINCT 的结果相同。

假设表a有三行数据：id=1，num=10；id=2，num=20；id=3，num=30。

例子

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

MIN

声明

```
MIN([DISTINCT] expr)
```

说明

返回指定数据中的最小值。

MIN()的取值可以是一个字符串参数；在这些情况下，它们返回最小字符串值。DISTINCT关键字可以被用来查找expr 的不同值的最小值，然而，这产生的结果与省略DISTINCT 的结果相同。

假设表a有三行数据：id=1，num=10；id=2，num=20；id=3，num=30。

例子

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

SUM

声明

```
SUM([DISTINCT] expr)
```

说明

返回 expr 的总数。若返回集合中无任何行，则 SUM() 返回NULL。DISTINCT关键字可用于求得 expr 不同值的总和。

若找不到匹配的行，则SUM()返回NULL。

例子

```
Oceanbase>select * from oceanbasetest;
+------+------+------+
| id  | ip  | ip2 |
+------+------+------+
|  1|  4 | NULL |
|  3|  3 | NULL |
|  4|  3 | NULL |
+------+------+------+
3 rows in set (0.00 sec)

Oceanbase>select sum(ip2),sum(ip),sum(distinct(ip)) from oceanbasetest;
+----------+---------+------------------+
| sum(ip2) | sum(ip) | sum(distinct(ip)) |
+----------+---------+------------------+
|   NULL |    10 |           7 |
+----------+---------+------------------+
1 row in set (0.00 sec)
```

GROUP_CONCAT

声明

```
GROUP_CONCAT([DISTINCT] expr)
```

说明

该函数返回带有来自一个组的连接的非NULL值的字符串结果。

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

例子

```
Oceanbase>select * from book;  //表book（书编号，书名，出版社）
+--------+--------------------------------+----------------------------+
| bookid | bookname                | publishname          |
+--------+--------------------------------+----------------------------+
|    1 | git help                | alibaba group publisher   |
|    2 | MySQL性能优化           | 浙江大学图文出版社      |
|    3 | JAVA编程指南            | 机械工业出版社        |
|    3 | JAVA编程指南            | 机械工业出版社        |
|    4 | 大规模分布式存储系统      | 机械工业出版社        |
```

```
+--------+-------------------------------+---------------------------+
5 rows in set (0.00 sec)

//查找书名信息
Oceanbase>select group_concat(bookname) from book group by bookname;
+-----------------------------------+
| group_concat(bookname)        |
+-----------------------------------+
| git help                  |
| JAVA编程指南,JAVA编程指南      |
| MySQL性能优化               |
| 大规模分布式存储系统          |
+-----------------------------------+
4 rows in set (0.00 sec)

//查找书名信息，书名唯一
Oceanbase>select group_concat(distinct(bookname)) from book group by bookname;
+-----------------------------------+
| group_concat(distinct(bookname)) |
+-----------------------------------+
| git help                  |
| JAVA编程指南               |
| MySQL性能优化              |
| 大规模分布式存储系统         |
+-----------------------------------+
4 rows in set (0.01 sec)

//查找书名和出版社信息，以书名分组，出版社信息降序排序显示
Oceanbase>select bookname, group_concat(publishname order by publishname desc separator ';' ) from b
ook group by bookname;
+--------------------------------+-----------------------------------------------------------------------------+
| bookname                | group_concat(publishname order by publishname desc separator ';' ) |
+--------------------------------+-----------------------------------------------------------------------------+
| git help                | alibaba group publisher                                     |
| JAVA编程指南             | 机械工业出版社;机械工业出版社                                    |
| MySQL性能优化            | 浙江大学图文出版社                                              |
| 大规模分布式存储系统       | 机械工业出版社                                                 |
+--------------------------------+-----------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

5.3.3. 分析函数

简介

分析函数（某些数据库下也叫做**窗口函数**）与聚集函数类似，计算总是基于一组行的集合，不同的是，聚集函数一组只能返回一行，而窗口函数每组可以返回多行，组内每一行都是基于窗口的逻辑计算的结果。分析函数可以显著优化需要self-join的查询。

功能支持

目前OceanBase支持以下分析函数：

- SUM
- MIN
- MAX
- COUNT
- AVG
- GROUP_CONCAT（ob特有）
- ROW_NUMBER
- RANK
- DENSE_RANK
- PERCENT_RANK
- CUME_DIST
- FIRST_VALUE
- LAST_VALUE
- NTH_VALUE
- NTILE
- LEAD
- LAG
- RATIO_TO_REPORT
- STDDEV
- VARIANCE
- STDDEV_SAMP
- STDDEV_POP
- LISTAGG

其中oracle支持ob还没有的窗口函数还有：

- CORR
- COVAR_POP
- COVAR_SAMP
- MEDIAN
- PERCENTILE_CONT

- PERCENTILE_DISC
- VAR_POP
- VAR_SAMP
- REGR_(Linear Regression) Functions

窗口函数语法

“窗口”也称为FRAME，OceanBase同时支持ROWS与RANGE两种FRAME语义，前者是基于物理行偏移的窗口，后者则是基于逻辑值偏移的窗口。

analytic_function的语法为analytic_function([arguments]) **OVER** (analytic_clause)

analytic_clause的语法为[query_partition_clause] [order_by_clause [windowing_clause]]

query_partition_clause的语法为**PARTITION BY** { expr[, expr]... | (expr[, expr]...) }

order_by_clause的语法为**ORDER** [**SIBLINGS**] **BY**{ expr | position | c_alias } [**ASC** | **DESC**] [**NULLS FIRST** | **NULLS LAST**] [, { expr | position | c_alias } [**ASC** | **DESC**][**NULLS FIRST** | **NULLS LAST**]]...

窗口语法

windowing_clause的语法为

{ **ROWS** | **RANGE** } { **BETWEEN** { **UNBOUNDED PRECEDING** | **CURRENT ROW** | value_expr { **PRECEDING** | **FOLLOWING** } } **AND**{ **UNBOUNDED FOLLOWING** | **CURRENT ROW** | value_expr { **PRECEDING** | **FOLLOWING** } } | { **UNBOUNDED PRECEDING** | **CURRENT ROW**| value_expr **PRECEDING**}}

window_clause的语法与oracle的语法兼容。

分析函数解释

以下逐个介绍分析函数的语法支持，功能及和oracle数据库的异同：

sum/min/max/count/avg

sum/min/max/count/avg的语法为**AVG**([**DISTINCT** | **ALL**] expr) [**OVER**(analytic_clause)]

sum的语法为**SUM**([**DISTINCT** | **ALL**] expr) [**OVER** (analytic_clause)]

min的语法为**MIN**([**DISTINCT** | **ALL**] expr) [**OVER** (analytic_clause)]

max的语法为**MAX**([**DISTINCT** | **ALL**] expr) [**OVER** (analytic_clause)]

count的语法为**COUNT** ({ * | [**DISTINCT** | **ALL**] expr }) [**OVER** (analytic_clause)]

以上带聚集函数的窗口函数，sum返回expr的和，min/max返回expr的最小值/最大值，count返回是窗口中查询的行数，avg返回的是expr的平均值。

对于count函数，从不返回null，如果指定了expr，即返回expr不为null的统计个数，如果指定count(*)返回所有行的统计数目。

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, sum(salary) over(partition by job_id) totol_s, min(salary) over(p
artition by job_id) min_s, max(salary) over(partition by job_id) max_s, count(*) over(partition by job_id) cou
nt_s from exployees;
+------------+---------+-------+-------+---------+
| LAST_NAME | TOTOL_S | MIN_S | MAX_S | COUNT_S |
+------------+---------+-------+-------+---------+
| jim       |  2000 | 2000 | 2000 |     1 |
| mike      |  36000 | 11000 | 13000 |     3 |
| lily      |  36000 | 11000 | 13000 |     3 |
| tom       |  36000 | 11000 | 13000 |     3 |
+------------+---------+-------+-------+---------+
```

以上五个函数，ob还没有完全做到和oracle语法上兼容。

对于distinct关键字，sum/count/avg，如果指定了distinct关键字，就只能指定analytic_clause中的query_partition_clause了。order_by_clause和windowing_clause是不被允许出现的。**这点ob还没有支持，ob仍然可以在这些函数的distinct的同时写order_by_clause和windowing_clause**。

现在order_by_clause中的nulls first和nulls last，现在还不支持。

analytic_clause的支持参照之前分析窗口语法分析。

注意方差函数在聚集函数中的用法和窗口函数略有不同，详细见聚集函数中的方差函数语法。

nth_value/first_value/last_value

nth_value的语法为**NTH_VALUE** (measure_expr, n) [**FROM** { **FIRST** | **LAST** }][{ **RESPECT** | **IGNORE** } **NULLS**] **OVER** (analytic_clause)

从名称可以看出，nth_value是第几个值的意思，方向由[**FROM** { **FIRST** | **LAST** }]确定，默认为from first，含有是否忽略null值的标志。其窗口为统一的analytic_clause。这里n应该是正数，如果n是null，函数将返回错误，如果n大于窗口内所有的行数，此函数将返回null。

first_value/last_value的语法为**FIRST_VALUE** { (expr) [{**RESPECT** | **IGNORE**} NULLS] | (expr [{**RESPECT** | **IGNORE**} NULLS])} **OVER** (analytic_clause)

从名称上可以看出，first_value和last_value是从第一个开始计数或者是从最后一个开始计数。

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, first_value(salary) over(partition by job_id) totol_s, last_value(
salary) over(partition by job_id) min_s, max(salary) over(partition by job_id) max_s from exployees;
+------------+---------+-------+-------+
| LAST_NAME | TOTOL_S | MIN_S | MAX_S |
+------------+---------+-------+-------+
| jim      |  2000 | 2000 | 2000 |
| mike     |  12000 | 11000 | 13000 |
| lily     |  12000 | 11000 | 13000 |
| tom      |  12000 | 11000 | 13000 |
+------------+---------+-------+-------+
```

以上三个函数，和oracle是兼容的，这三个函数没有对应的聚集函数。

lead/lag

lag和lead含义为可以在一次查询中取出当前行的同一个字段的前面第N行的数据，这种操作可以使用相同表的自连接来实现，但lag/lead窗口函数有更高的效率。语法如下：

lag/lead { (value_expr [, offset [, default]]) [{ **RESPECT** | **IGNORE** } **NULLS**] | (value_expr [{ **RESPECT** | **IGNORE** } **NULLS**] [, offset [, default]])} **OVER** ([query_partition_clause] order_by_clause)

其中，value_expr是要做比对的字段，offset是value_expr的偏移量，default参数的默认值为空值null，即如果在lag/lead没有显示的设置default值的情况下，返回值为空值null。例如：对lag来说，当前行为4，offset值为6，这时候所要找的数据就是第-2行，不存在此行即返回default的值。

[{ **RESPECT** | **IGNORE** } **NULLS**]的语法为是否考虑null值，默认为**RESPECT**，考虑null值。

注意lead/lag两个函数后必须跟order_by_clause，数据应该在一个列上排序之后才能有前多少行后多少行的概念。query_partition_clause是可选的，如果没有query_partition_clause的话，就是全局的数据。

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, lead(salary) over(order by salary) lead, lag(salary) over(order by
salary) lag from exployees;
+------------+-------+-------+
| LAST_NAME | LEAD | LAG  |
+------------+-------+-------+
| jim      | 11000 |  NULL |
| tom      | 12000 |  2000 |
| mike     | 13000 | 11000 |
| lily     |  NULL | 12000 |
+------------+-------+-------+
```

以上两个函数，和oracle是兼容的，注意value_expr和lag的结果都是可以对null做忽略或者不忽略的。
lead/lag两个函数没有对应的聚集函数。

stddev/variance/stddev_samp/stddev_pop

variance的语法为**VARIANCE**([**DISTINCT** | **ALL**] expr) [**OVER** (analytic_clause)]返回的是expr的方差，expr可能是数值类型或者可以转换成数值类型的类型，方差的类型和输入的值的类型相同。

stddev的语法为**STDDEV**([**DISTINCT** | **ALL**] expr) [**OVER** (analytic_clause)]，它的语法和variance是一样的，stddev的含义返回的是expr的标准差，参数类型方面和variance的相同。

stddev_samp的语法为**STDDEV_SAMP**(expr) [**OVER** (analytic_clause)]

stddev_pop的语法为**STDDEV_POP**(expr) [**OVER** (analytic_clause)]

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, stddev(salary) over(order by salary) std, variance(salary) over(o
rder by salary) var, stddev_pop(salary) over() std_pop, stddev_samp(salary) over() from exployees;
+------------+-----------------------------------------+-----------------------------------------+-----------------------------------------
+-----------------------------------------+
| LAST_NAME | STD                                     | VAR                                     | STD_POP                                 | STDDEV_SAMP(SALARY
)OVER()                                     |
+------------+-----------------------------------------+-----------------------------------------+-----------------------------------------
+-----------------------------------------+
| jim        |                                       0 |                                       0 | 4387.4821936960610302031941537081547804380 | 5066.2280
5119022121055624806867988167922 7 |
| tom        | 6363.961030678927719607599258943641353564 |                                40500000 | 4387.4821936960610302
0319415370815478043 8 | 5066.228051190221210556248068679881679227 |
| mike       | 5507.57054728610202060580877087207347091 6 | 30333333.333333333333333333333333333 | 4
387.4821936960610302031941537081547804380 | 5066.228051190221210556248068679881679227 |
| lily       | 5066.228051190221210556248068679881679227 | 25666666.666666666666666666666666667 | 438
7.4821936960610302031941537081547804380 | 5066.228051190221210556248068679881679227 |
+------------+-----------------------------------------+-----------------------------------------+-----------------------------------------
+-----------------------------------------+
```

以上4个函数，ob还没有完全做到和oracle语法上兼容。

对于distinct关键字，variance/stddev函数，如果指定了distinct关键字，就只能指定analytic_clause中的query_partition_clause了。order_by_clause和windowing_clause是不被允许出现的。**这点ob还没有支持，ob仍然可以在这些函数的distinct的同时写order_by_clause和windowing_clause。**
stddev_pop/stddev_samp函数，则不被允许指定distinct/all关键字。**这点ob也还没有支持。**

现在order_by_clause中的nulls first和nulls last，现在还不支持。

analytic_clause的支持参照之前分析窗口语法分析。

注意方差函数在聚集函数中的用法和窗口函数略有不同，详细见聚集函数中的方差函数语法。

ntile

ntile的语法为**NTILE**(expr) **OVER** ([query_partition_clause] order_by_clause)，ntile函数将分区中已经排序的行划分为大小尽可能相同的指定数量的分组，它的语法中query_partition_clause可选。并返回给每行组号。expr如果是null也会返回null。

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, ntile(10) over(partition by job_id order by salary) ntl from explo
yees;
+------------+------+
| LAST_NAME | NTL |
+------------+------+
| jim     |  1 |
| tom     |  1 |
| mike    |  2 |
| lily    |  3 |
+------------+------+
```

ob中的ntile函数与oracle兼容，注意ntile的窗口中是有order_by_clause的。没有ntile相应的聚集函数。

row_number

row_number的语法为**ROW_NUMBER( ) OVER** ([query_partition_clause] order_by_clause)，它的语法中query_partition_clause可选，row_number在order by之后的expr进行排序的情况下每行一个编号，前提是排序。

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, row_number() over(partition by job_id order by salary) ntl from
exployees;
+------------+------+
| LAST_NAME | NTL |
+------------+------+
| jim     |  1 |
| tom     |  1 |
| mike    |  2 |
| lily    |  3 |
+------------+------+
```

ob中的row_number函数与oracle兼容，注意row_number的窗口中是有order_by_clause的。没有row_number相应的聚集函数。

rank/dense_rank/percent_rank

rank的语法为**RANK( ) OVER** ([query_partition_clause] order_by_clause)，作为一个分析函数，rank计算每一行数据在某列上的排序，该列由order_by_clause中的列决定。如下按照salary排序可以看出员工的收入排名。

dense_rank的语法为**DENSE_RANK( ) OVER**([query_partition_clause] order_by_clause)，它的语义基本和rank函数相同，但是rank的排序中间会有'跳过'，但是dense_rank中不会有。

percent_rank的语法为**PERCENT_RANK( ) OVER** ([query_partition_clause] order_by_clause)，它的语义基本和rank函数相同，但是percent_rank排序的是百分比，计算的是给定行的百分比。

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, rank() over(partition by job_id order by salary) rank, dense_ran
k() over(partition by job_id order by salary) dense_rank, percent_rank() over(partition by job_id order by sa
lary) percent_rank from exployees;
+------------+------+------------+--------------+
| LAST_NAME | RANK | DENSE_RANK | PERCENT_RANK |
+------------+------+------------+--------------+
| jim | 1 | 1 | 0 |
| tom | 1 | 1 | 0 |
| mike | 2 | 2 | .5 |
| lily | 3 | 3 | 1 |
+------------+------+------------+--------------+
```

以上三个函数：ob中的rank函数与oracle兼容，注意rank的窗口中是有order_by_clause的。注意rank在聚集函数中的用法和窗口函数中略有不同，详细请见聚集函数中的rank函数语法；ob中的dense_rank函数与oracle兼容，注意dense_rank的窗口中是有order_by_clause的。注意dense_rank在聚集函数中的用法和窗口函数中略有不同，详细请见聚集函数中的dense_rank函数语法；ob中的percent_rank函数和oracle兼容，注意percent_rank的窗口中是有order_by_clause的，注意percent_rank在聚集函数中的用法和窗口函数略有不同，详细见聚集函数中的percent_rank函数语法。

cume_dist

cume_dist的语法为**CUME_DIST** () **OVER** ([query_partition_clause] order_by_clause)，cume_dist计算一个值的分布，返回值为>0 <=1，作为一个分析函数，cume_dist在生序情况下计算比当前行的特定列小的数据的占比。比如如下例子是，按job_id分组并在薪水排序的情况下，每行数据在窗口内的排序列上的占比。

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, cume_dist() over(partition by job_id order by salary) rank from
exployees;
+------------+-----------------------------------------+
| LAST_NAME | RANK                                    |
+------------+-----------------------------------------+
| jim      |                                       1 |
| tom      | .33333333333333333333333333333333333333 |
| mike     | .66666666666666666666666666666666666667 |
| lily     |                                       1 |
+------------+-----------------------------------------+
```

ob中的cume_dist与oracle兼容。注意cume_dist函数的语法中，窗口中可以含有query_partition_clause也可以含有order_by_clause。注意cume_dist在聚集函数中的用法和窗口函数中略有不同，详细请见聚集函数中的cume_dist函数语法。

ratio_to_report

ratio_to_report的语法为**RATIO_TO_REPORT** (expr) **OVER** ([query_partition_clause])，它计算expr的值和窗口内这个值集合sum的比率，如果expr为null，ratio_to_report的值一样为null。

值集合的sum为query_partition_clause所决定，如果省略此clause，ratio_to_report在expr所有值集合上计算sum。expr不能是ratio_to_report或者其他分析函数的嵌套，但可以是类似与聚集函数中expr的内部函数嵌套表达式。下面举例说明ratio_to_report的功能，下面的例子计算的是每个公司职员在全公司中的工资占比：

```
create table exployees(last_name char(10), salary decimal, job_id char(32));
insert into exployees values('jim', 2000, 'cleaner');
insert into exployees values('mike', 12000, 'engineering');
insert into exployees values('lily', 13000, 'engineering');
insert into exployees values('tom', 11000, 'engineering');
OceanBase(TEST@TEST)>select last_name, salary, ratio_to_report(salary) over() as salary_ratio from explo
yees where job_id = 'engineering';
+------------+--------+-----------------------------------------+
| LAST_NAME | SALARY | SALARY_RATIO                            |
+------------+--------+-----------------------------------------+
| mike     | 12000 | .33333333333333333333333333333333333333 |
| lily     | 13000 | .36111111111111111111111111111111111111 |
| tom      | 11000 | .30555555555555555555555555555555555556 |
+------------+--------+-----------------------------------------+
```

ob中的ratio_to_report与oracle兼容。注意ratio_to_report的语法中，窗口中不能含有order_by_clause。

需要根据oracle每个窗口函数的语法，包括每个窗口函数是否会出现
windowing_clause，ordering_clause，query_partition_exprs，两个纬度首先测试一下正确性和功能。

执行计划

OceanBase分析函数的相应算子是： WINDOW FUNCTION，该算子依赖下层算子的输出按分区 + 排序键有序，所以在必要时会分配排序算子

注：

```
mysql> explain select row_number() over (partition by table_id order by partition_id) from __all_root_table
;
+-----------------------
```

如果优化阶段已经知道下层输出是有序的，则不会分配排序算子（(tenant_id，table_id)为__all_root_table主键前缀）

```
explain select row_number() over (partition by tenant_id, table_id order by partition_id) from __all_root_table;
+----------------------
```

当同时存在多个分析函数时，如果存在可以共享一个序的子集，子集内所有函数会在同一个算子里计算而不是拆到多个算子，这样可以有效减少内存开销和边界判断。

如下面的例子所示，row_number()与rank()两个分析函数的计算可以共用同一个排序的输出，因此，只会分配一个sort节点：

```
mysql> explain select row_number() over (partition by table_id order by partition_id), rank() over (partition
by table_id, partition_id order by svr_ip) from __all_root_table;
| ==============================================
|ID|OPERATOR     |NAME         |EST. ROWS|COST|
----------------------------------------------
|0 |WINDOW FUNCTION|           |1000     |3017|
|1 | SORT       |              |1000     |2826|
|2 |  TABLE SCAN  |__all_root_table|1000   |499 |
==============================================

Outputs & filters:
-------------------------------------
 0 - output([T_WIN_FUN_ROW_NUMBER()], [T_WIN_FUN_RANK()]), filter(nil),
    win_expr(T_WIN_FUN_RANK()), partition_by([__all_root_table.table_id], [__all_root_table.partition_id]),
order_by([__all_root_table.svr_ip, ASC]), window_type(RANGE), uppe              r(UNBOUNDED PRECEDING), l
ower(UNBOUNDED FOLLOWING)
    win_expr(T_WIN_FUN_ROW_NUMBER()), partition_by([__all_root_table.table_id]), order_by([__all_root_
table.partition_id, ASC]), window_type(RANGE), upper(UNBOUNDED PRECEDING              ), lower(UNBOUND
ED FOLLOWING)
 1 - output([__all_root_table.table_id], [__all_root_table.partition_id], [__all_root_table.svr_ip]), filter(nil),
sort_keys([__all_root_table.table_id, ASC], [__all_root_tabl              e.partition_id, ASC], [__all_root_table.s
vr_ip, ASC])
 2 - output([__all_root_table.table_id], [__all_root_table.partition_id], [__all_root_table.svr_ip]), filter(nil),
    access([__all_root_table.table_id], [__all_root_table.partition_id], [__all_root_table.svr_ip]), partitions(
p0)
```

落盘

分析函数的计算也支持执行期的落盘操作，其逻辑与排序操作符的落盘逻辑一致。

窗口函数已知问题汇总

以下问题是窗口函数已知的问题汇总，测试如果测到，如有必要在原issue中补充说明，不必再提新的issue：

1. https://work.aone.alibaba-inc.com/issue/22521949
2. 部分窗口函数用了distinct关键字就不能在用orderby_clause和window_clause。
   https://work.aone.alibaba-inc.com/issue/23279380

3. 部分窗口函数执行可以计算增量窗口来提升性能，而不是每次都是partition by后的窗口内所有的行参与计算。

   https://aone.alibaba-inc.com/req/21444165

5.3.4. 信息函数

FOUND_ROWS

声明

```
found_rows()
```

说明

一个SELECT语句可能包含一个LIMIT子句，用来限制数据库服务器端返回客户端的行数。在某些情况下，我们需要不再次运行该语句而得知在没有LIMIT时到底该语句返回了多少行。我们可以在SELECT语句中选择使用SQL_CALC_FOUND_ROWS, 然后调用FOUND_ROW()函数，获取该语句在没有LIMIT时返回的行数。

例如：

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

第二个SELECT语句返回一个数字，表示在没有LIMIT子句的情况下，第一个SELECT语句返回了多少行。（若上述的SELECT语句在不使用SQL_CALC_FOUND_ROWS选项时，使用LIMIT和不使用LIMIT时候，FOUND_ROWS()可能会返回不同的结果）。

通过FOUND_ROWS()函数返回的有效行数是瞬时的，并且不能越过SELECT SQL_CALC_FOUND_ROWS语句后面的语句。如果你后续还需要用到这个值，就需要将其保存。

例如：

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

假如你正在使用SQL_CALC_FOUND_ROWS，系统必须计算出在全部结果集合中有多少行。尽管如此，这也还是比不用LIMIT而再次运行查询要快，原因是结果集合不需要被发送到客户端。

SQL_CALC_FOUND_ROWS和FOUND_ROWS()在当你希望限制一个查询返回的行数时是很有用的，同时还能不需要再次运行查询就可以确定全部结果集合中的行数。一个**例子**就是提供页式显示的Web脚本，该显示包含显示搜索结果其他部分的页的链接。使用FOUND_ROWS()使你确定剩下的结果需要多少其他的页。

SQL_CALC_FOUND_ROWS 和 FOUND_ROWS() 的应用对于UNION 查询比对于简单SELECT 语句更为复杂，原因是在UNION 中，LIMIT 可能会出现在多个位置。它可能适用于UNION中的独立的SELECT语句，或是整个的UNION 结果。

SQL_CALC_FOUND_ROWS对于 UNION的期望结果是它返回在没有全局的LIMIT的条件下而应返回的行数。SQL_CALC_FOUND_ROWS 和UNION 一同使用的条件是：

- SQL_CALC_FOUND_ROWS 关键词必须出现在UNION的第一个 SELECT中。
- FOUND_ROWS()的值只有在使用 UNION ALL时才是精确的。若使用不带ALL的UNION，则会发生两次删除，而FOUND_ROWS() 的指只需近似的。

- 假若UNION 中没有出现LIMIT ，则SQL_CALC_FOUND_ROWS 被忽略，返回临时表中的创建的用来处理UNION的行数。

LAST_INSERT_ID()

声明

```
last_insert_id()
```

说明

返回本session最后一次插入的自增字段值，如最近一条insert插入多条记录，LAST_INSERT_ID()返回第一条记录的自增字段值。

例子

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

5.3.5. 其它函数

COALESCE

声明

```
COALESCE(expr, expr, expr,...)
```

说明

依次参考各参数表达式，遇到非NULL值即停止并返回该值。如果所有的表达式都是空值，最终将返回一个空值。

所有表达式必须是相同类型，或者可以隐性转换为相同的类型。

例子

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

NVL

声明

NVL(str1,replace_with)

说明

如果 str1 为 NULL，则替换成 replace_with。

任何时候给它一个空值，它都返回一个你所选择的值。这种能够自动替换空值的能力有助于提供看上去更为完善的输出。其中 str1 一般是一个列名。replace_with 可以是任何值：直接值（即硬编码）、对其他列的引用或者表达式。

例子

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

SLEEP

声明

SLEEP(duration)

说明

SLEEP函数根据duration指定的数值暂停相应的时间（单位为秒），并在暂停结束后返回0。

如果SLEEP单独执行且没有被中断，返回结果0；

如果SLEEP单独执行期间被中断，返回结果1，但不会返回任何错误码；

如果SLEEP是查询的一部分，且暂停期间被中断，将会返回错误码1317；

例子

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

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

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

全文查找函数

声明

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

说明

OceanBase1.0已经支持使用全文查找函数来对全文索引进行查找，并且使用全文查找函数有以下要求：

- 全文查找函数MATCH(col1,col2,...)中指定的列上必须有全文索引(OB目前只支持FULLTEXT CTXCAT索引)
- 在FULLTEXT CTXCAT索引中，全文查找函数MATCH(col1,col2,...)中指定的列必须完整的覆盖索引中的全文列，例如FULLTEXT INDEX(c1, c2, c3), CTXCAT(c2, c3) 必须是 MATCH(c2,c3) 才能完全的匹配
- 全文查找函数可以通过上述的关键字来指定查找模式(OB目前只支持NATURAL LANGUAGE MODE和BOOLEAN MODE两种模式)，缺省是NATURAL LANGUAGE MODE

默认情况下或者指定IN NATURAL LANGUAGE MODE标示符，MATCH...AGAINST将使用NATURAL LANGUAGE模式来进行全文查找，在NATURAL LANGUAGE模式下，AGAINST接受一个查找字符串，并且按照字符集的比较方式在索引中进行查找，对于表中的每一行数据，MATCH的返回值代表了查找字符串和行中数据的相关度，也就是查找字符串中的文本和数据表中的文本相似度。在默认情况下，OB创建的字符串相关的列是大小写不敏感的，因此，全文查找的关键字是不区分大小写的。如果需要区分大小写，可以为创建全文索引的列指定大小写敏感的数据类型，例如UTF8MB4_BIN。如果MATCH...AGAINST函数被用在WHERE子句中，MATCH被用来过滤跟关键字相关度不匹配的数据，MATCH...AGAINST=0表示没有和关键字相似的数据，目前OB只支持MATCH...AGAINST=0和MATCH...AGAINST>0两种形式，即完全不相关或者有任意一个关键字相关即可。AGAINST参数中可以接受多个关键字，关键字之间使用' '隔开，表示OR关系，只要有任意一个关键字匹配，即认为符合查找的要求。

OB可以通过使用IN BOOLEAN MODE关键字来进行BOOLEAN模式的全文查找。在这种模式中，关键字前面一些特殊的操作符含有特殊的语义。例如：

```
SELECT * FROM t1 WHERE MATCH (a, b) AGAINST ('菊花 茉莉花' IN BOOLEAN MODE);
+----+------------+------------+
| id | a          | b          |
+----+------------+------------+
|  1 | 支付宝     | 菊花茶     |
|  2 | 淘宝       | 茉莉花     |
+----+------------+------------+

SELECT * FROM t1 WHERE MATCH (a, b)
  AGAINST ('+菊花 -茉莉花' IN BOOLEAN MODE);
+----+------------+------------+
| id | a          | b          |
+----+------------+------------+
|  1 | 支付宝     | 菊花茶     |
+----+------------+------------+
```

OB的BOOLEAN全文查找目前支持以下几种操作符：+ 代表AND关系，表示查找结果中同时要包含被+修饰的关键字- 代表NOT关系，表示查找结果中不能包含被-修饰的关键字。

```
(no operator) 代表OR关系，表示查找结果中只要包含任意一个没有操作符修饰的关键字即可
```

在BOOLEAN全文查找模式中有以下几点需要注意：操作符必须位于关键字的前面，关键字后面的操作符没有修饰意义，例如+菊花是有意义的操作符修饰，而菊花+中的操作符没有修饰意义。操作符和关键字必须紧密相连，不能被其它符号分割开，否者没有修饰意义，例如：+ 菊花前面的操作符没有修饰意义。

5.4. 查询和子查询

5.4.1. 概述

查询(SQL)是指数据库中用来获取数据的方式，它可搭配条件限制的子句(如where)，排列顺序的子句（如order by）等语句来获取查询结果。子查询是指嵌套在一个上层查询中的查询。上层的查询一般被称为父查询或外层查询。子查询的结果作为输入传递回“父查询”或“外部查询”。父查询将这个值结合到计算中，以便确定最后的输出。SQL语言允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询。同时，子查询可以出现在SQL语句中的各种子句中，比如select语句，from语句，where语句等。

子查询

在数据库中，子查询可以分成有依赖关系的子查询和没有依赖关系的子查询。有依赖关系的子查询是指该子查询的执行依赖了外部查询的'变量'，所以这种子查询通常会被计算多次。没有依赖关系的子查询是指该子查询的执行不依赖外部查询的'变量', 这种子查询一般只需要计算一次。下图分别展示了一个没有依赖关系的子查询和一个有依赖关系的子查询。

```
OceanBase (root@test)> create table t1(a int primary key, b int, c int);
Query OK, 0 rows affected (0.70 sec)
OceanBase (root@test)> create table t2(a int primary key, b int, c int);
Query OK, 0 rows affected (0.92 sec)
-- 没有依赖关系的子查询
OceanBase (root@test)> select * from t1 where t1.a in (select t2.a from t2);
Empty set (0.22 sec)
-- 有依赖关系的子查询，子查询中用到了外层查询变量t1.b
OceanBase (root@test)> select * from t1 where t1.a in (select t2.a from t2 where t2.b = t1.b);
Empty set (0.05 sec)
```

子查询展开(subquery unnesting)

子查询展开是数据库的一种优化策略，它把一些子查询置于外层的父查询中，其实质是把某些子查询转化为等价的多表连接操作。这种策略带来的一个明显的好处就是，有写访问路径，连接方法和连接顺序可能被有效的利用，使得查询语句的层次尽可能的减少。下图展示了一个子查询展开的例子，既子查询被改写成了连接语句。

```
OceanBase (root@test)> create table t1(a int primary key, b int, c int);
Query OK, 0 rows affected (0.70 sec)
OceanBase (root@test)> create table t2(a int primary key, b int, c int);
Query OK, 0 rows affected (0.92 sec)
--- 有依赖关系的子查询被展开改写成连接
OceanBase (root@test)> explain select * from t1 where t1.a in (select t2.b from t2 where t2.c = t1.c);
| ======================================
|ID|OPERATOR     |NAME|EST. ROWS|COST|
--------------------------------------
|0 |HASH SEMI JOIN|   |1      |2924|
|1 | TABLE SCAN   |t1  |1000    |455 |
|2 | TABLE SCAN   |t2  |1000    |455 |
======================================

Outputs & filters:
-------------------------------------
 0 - output([t1.a], [t1.b], [t1.c]), filter(nil),
    equal_conds([t1.a = t2.b], [t2.c = t1.c]), other_conds(nil)
 1 - output([t1.c], [t1.a], [t1.b]), filter(nil),
    access([t1.c], [t1.a], [t1.b]), partitions(p0)
 2 - output([t2.c], [t2.b]), filter(nil),
    access([t2.c], [t2.b]), partitions(p0)
```

5.4.2. 连接

数据库中的连接语句用于将数据库中的两个或多个表根据连接条件组合起来。由"连接"生成的集合，可以被保存为表，或者当成表来使用。连接语句的含义是把两张表的属性通过它们的值组合在一起。数据库中的连接类型一般包括inner join，outer join，semi-join和anti-join。其中Semi-join和Anti-join都是通过子查询改写得到，SQL本身并没有表述Anti-join和Semi-join的语法

连接条件

连接条件可以分为等值连接（比如t1.a = t2.b）和非等值连接（t1.a < t2.b）。相比于非等值连接条件，等值连接条件的一个好处是允许数据库中使用高效的连接算法，比如Hash Join和Merge-Sort join。

Self-join

Self-Join是指跟跟自己表做连接的join。下图展示了一个self join的例子。

```
OceanBase (root@test)> create table t1(a int primary key, b int, c int);
Query OK, 0 rows affected (0.70 sec)
---一个self join的例子
OceanBase (root@test)> select * from t1 as ta, t1 as tb where ta.b = tb.b
```

内连接(inner join)

Inner Join(内连接)是数据库中最基本的连接操作。内连接基于连接条件将两张表(如 A 和 B)的列组合在一起，产生新的结果表。查询会将 A 表的每一行和 B 表的每一行进行比较，并找出满足连接条件的组合。当连接条件被满足，A 和 B 中匹配的行会按列组合(并排组合)成结果集中的一行。连接产生的结果集，可以定义为首先对两张表做笛卡尔积(交叉连接) -- 将 A 中的每一行和 B 中的每一行组合，然后返回满足连接条件的记录。

外连接(outer join)

Outer Join（外连接）并不要求连接的两表的每一条记录在对方表中都一条匹配的记录。要保留所有记录（甚至这条记录没有匹配的记录也要保留）的表称为保留表。外连接可依据连接表保留左表, 右表或全部表的行而进一步分为左外连接, 右外连接和全连接。其中左外连接中左表的一行未在右表中找到的时候，就在右表自动填充NULL。右外连接中右表的一行未在左表中找到的时候，就在左表自动填充NULL。全连接就是左表或者右表找不匹配行的时候都会自动填充。

Semi连接 (semi-join)

当A表和B表进行left/right semi-join的时候，它只返回A/B表中所有能够在B/A中找到匹配的行。Semi-join只能通过子查询展开得到，如下图所示。

```
OceanBase (root@test)> create table t1(a int primary key, b int, c int);
Query OK, 0 rows affected (0.70 sec)
OceanBase (root@test)> create table t2(a int primary key, b int, c int);
Query OK, 0 rows affected (0.92 sec)
--- 有依赖关系的子查询被展开改写成Semi-join
OceanBase (root@test)> explain select * from t1 where t1.a in (select t2.b from t2 where t2.c = t1.c);
| ======================================
|ID|OPERATOR    |NAME|EST. ROWS|COST|
--------------------------------------
|0 |HASH SEMI JOIN|   |1     |2924|
|1 | TABLE SCAN  |t1 |1000   |455 |
|2 | TABLE SCAN  |t2 |1000   |455 |
======================================

Outputs & filters:
-------------------------------------
 0 - output([t1.a], [t1.b], [t1.c]), filter(nil),
    equal_conds([t1.a = t2.b], [t2.c = t1.c]), other_conds(nil)
 1 - output([t1.c], [t1.a], [t1.b]), filter(nil),
    access([t1.c], [t1.a], [t1.b]), partitions(p0)
 2 - output([t2.c], [t2.b]), filter(nil),
    access([t2.c], [t2.b]), partitions(p0)
```

Anti-连接(anti-join)

当A表和B表进行left/right anti-join的时候，它只返回A/B中所有不能再B/A中找到匹配的行。类似于Semi-join, anti-join也只能通过子查询展开得到，如下图所示。

```
OceanBase (root@test)> create table t1(a int primary key, b int, c int);
Query OK, 0 rows affected (0.70 sec)
OceanBase (root@test)> create table t2(a int primary key, b int, c int);
Query OK, 0 rows affected (0.92 sec)
---有依赖关系的子查询被改写成Anti-join
OceanBase (root@test)> explain select * from t1 where t1.a not in (select t2.b from t2 where t2.c = t1.c);
| ====================================
|ID|OPERATOR     |NAME|EST. ROWS|COST|
------------------------------------
|0 |HASH ANTI JOIN|   |995     |3262|
|1 | TABLE SCAN   |t1  |1000    |455 |
|2 | TABLE SCAN   |t2  |1000    |455 |
====================================

Outputs & filters:
-------------------------------------
 0 - output([t1.a], [t1.b], [t1.c]), filter(nil),
   equal_conds([t2.c = t1.c]), other_conds([t1.a = t2.b OR (T_OP_IS, t2.b, NULL, 0)])
 1 - output([t1.c], [t1.a], [t1.b]), filter(nil),
   access([t1.c], [t1.a], [t1.b]), partitions(p0)
 2 - output([t2.c], [t2.b]), filter(nil),
   access([t2.c], [t2.b]), partitions(p0)
```

5.4.3. 集合

数据库中的集合操作可以把多个查询的结果组合成一个结果集。集合操作主要包含：

- UNION
- INTERSECT
- EXCEPT / MINUS在Ocenabase中，我们同时支持EXCEPT与MINUS，这两者的语义是相同的。这里需要注意的是参加集合操作的各查询结果的列数必须相同，对应的数据类型也必须兼容。对与UNION来说用户可以指定UNION的属性为ALL和DISTINCT / UNIQUE。分别代表集合可重复，和集合不可重复。而其它几种集合操作是不能指定ALL属性的(它们只有DISTINCT属性)。所有的集合操作默认的属性是DISTINCT。在Oceanbase中，集合操作中可以指定order by和 limit子句，但是不允许其他子句的出现，如下图所示

```
OceanBase (root@test)> create table t1(a int primary key, b int, c int);
Query OK, 0 rows affected (0.16 sec)
OceanBase (root@test)> create table t2(a int primary key, b int, c int);
Query OK, 0 rows affected (0.10 sec)
--支持union语句中出现order by和limit子句
OceanBase (root@test)> (select * from t1 union all select * from t2) order by a limit 10;
Empty set (0.02 sec)
--不支持union语句中出现除order by和limit子句的其他子句，比如group by
OceanBase (root@test)> OceanBase (root@test)> (select * from t1 union all select * from t2) group by a limit
10;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your Ocea
nBase version for the right syntax to use near 'OceanBase (root@test)> (select * from t1 union all select * fro
m t2) group by a ' at line 1
```

UNION 例子

该例子获取t1和t2中所有不重复的行。

```
OceanBase (root@test)> create table t1(a int, b int, c int);
Query OK, 0 rows affected (0.12 sec)
OceanBase (root@test)> create table t2(a int, b int, c int);
Query OK, 0 rows affected (0.11 sec)
OceanBase (root@test)> insert into t1 values (1,1,1),(2,2,2),(3,3,3);
Query OK, 3 rows affected (0.07 sec)
Records: 3  Duplicates: 0  Warnings: 0
OceanBase (root@test)> insert into t2 values (2,2,2),(3,3,3),(4,4,4);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
OceanBase (root@test)> select * from t1 union select * from t2;
+------+------+------+
|a  |b  |c  |
+------+------+------+
|  1|  1|  1|
|  2|  2|  2|
|  3|  3|  3|
|  4|  4|  4|
+------+------+------+
4 rows in set (0.01 sec)
```

UNION All 例子

该例子获取t1和t2中的所有行，不进行去重。

```
OceanBase (root@test)> create table t1(a int, b int, c int);
Query OK, 0 rows affected (0.12 sec)
OceanBase (root@test)> create table t2(a int, b int, c int);
Query OK, 0 rows affected (0.11 sec)
OceanBase (root@test)> insert into t1 values (1,1,1),(2,2,2),(3,3,3);
Query OK, 3 rows affected (0.07 sec)
Records: 3  Duplicates: 0  Warnings: 0
OceanBase (root@test)> insert into t1 values (2,2,2),(3,3,3),(4,4,4);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
OceanBase (root@test)> select * from t1 union all select * from t2;
+------+------+------+
|a  |b  |c  |
+------+------+------+
|  1|  1|  1|
|  2|  2|  2|
|  3|  3|  3|
|  2|  2|  2|
|  3|  3|  3|
|  4|  4|  4|
+------+------+------+
6 rows in set (0.02 sec)
```

INTERSECT 例子

该例子获取同时出现在t1和t2中行，并且去重。

```
OceanBase (root@test)> create table t1(a int, b int, c int);
Query OK, 0 rows affected (0.12 sec)
OceanBase (root@test)> create table t2(a int, b int, c int);
Query OK, 0 rows affected (0.12 sec)
OceanBase (root@test)> insert into t1 values (1,1,1),(2,2,2),(3,3,3);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
OceanBase (root@test)> insert into t2 values (2,2,2),(3,3,3),(3,3,3),(4,4,4);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
OceanBase (root@test)> select * from t1 intersect select * from t2;
+------+------+------+
|a  |b  |c  |
+------+------+------+
|  2|  2|  2|
|  3|  3|  3|
+------+------+------+
2 rows in set (0.01 sec)
```

EXCEPT/MINUS 例子

该例子获取出现在t1中，但是不出现在t2中的行，并且去重。

```
OceanBase (root@test)> create table t1(a int, b int, c int);
Query OK, 0 rows affected (0.12 sec)
OceanBase (root@test)> create table t2(a int, b int, c int);
Query OK, 0 rows affected (0.12 sec)
OceanBase (root@test)> insert into t1 values (1,1,1),(2,2,2),(3,3,3);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
OceanBase (root@test)> insert into t2 values (2,2,2),(3,3,3),(3,3,3),(4,4,4);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0
OceanBase (root@test)> select * from t1 except select * from t2;
+------+------+------+
|a  |b  |c  |
+------+------+------+
|  1|  1|  1|
+------+------+------+
1 row in set (0.02 sec)
```

5.5. SQL语句

5.5.1. 通用语法

常量

- `INT_VALUE`：匹配整数类型常量，如 `123`。
- `DECIMAL_VALUE`：匹配定点类型常量，如 `123.456`。
- `STR_VALUE`：匹配字符串常量，如 `abc`。
- `NULL`：匹配NULL常量。
- `STORAGE_SIZE`：匹配存储长度常量，整数形式默认单位为字节，字符串形式可指定单位为M或G，如 `1024` 或 `'500M'`。

```
const_value:
    INT_VALUE
  | DECIMAL_VALUE
  | STR_VALUE
  | NULL
  | STORAGE_SIZE

STORAGE_SIZE:
    INT_VALUE
  | 'INT_VALUE[M|G]'
```

字符集

```
charset:
    default_charset
  | column_charset

default_charset:
  [DEFAULT] {CHARSET | CHARACTER SET} [=] {UTF8 | UTF8MB4 | BINARY}

column_charset:
  {CHARSET | CHARACTER SET} {UTF8 | UTF8MB4 | BINARY}
```

排序规则

```
collate:
    default_collate
  | column_collate

default_collate:
  [DEFAULT] COLLATE [=] {UTF8MB4_GENERAL_CI | UTF8MB4_BIN | BINARY}

column_collate:
  COLLATE {UTF8MB4_GENERAL_CI | UTF8MB4_BIN | BINARY}
```

数据类型

```
data_type:
    TINYINT[(precision)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(precision)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(precision)] [UNSIGNED] [ZEROFILL]
  | INT[(precision)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(precision)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(precision)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(precision, scale)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(precision, scale)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(precision [, scale])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(precision [, scale])] [UNSIGNED] [ZEROFILL]
  | DATETIME[(scale)]
  | TIMESTAMP[(scale)]
  | DATE
  | TIME[(scale)]
  | YEAR
  | VARCHAR(length) column_charset column_collate
  | VARBINARY(length)
  | CHAR[(length)] column_charset column_collate
  | BINARY[(length)]
  | TINYTEXT column_charset column_collate
  | TINYLOB
  | TEXT[(length)] column_charset column_collate
  | BLOB[(length)]
  | MEDIUMTEXT column_charset column_collate
  | MEDIUMBLOB
  | LONGTEXT column_charset column_collate
  | LONGBLOB

precision | scale | length:
  INT_VALUE
```

SQL属性

- 对象名

```
tenant_name | pool_name | unit_name | zone_name | region_name:
  STR_VALUE

database_name | table_name | table_alias_name | column_name| column_alias_name  | partition_name | su
bpartition_name:
  STR_VALUE

index_name | view_name | object_name | constraint_name | tablegroup_name:
  STR_VALUE

outline_name | user_name:
  STR_VALUE

table_factor:
  [[database_name].] table_name

column_factor:
  [table_factor.] column_name
```

- 表达式

```
expression:
    const_value
  | column_factor
  | operator_expression
  | function_expression
```

- 注释

```
comment:
  COMMENT 'comment_text'

comment_text:
  STR_VALUE
```

分布式属性

- `PRIMARY_ZONE`：指定数据的主副本分布策略。

```
primary_zone:
  PRIMARY_ZONE [=] zone_name
```

- `ZONE_LIST`：指定租户的资源分布策略。

```
zone_list:
  ZONE_LIST [=] (zone_name [, zone_name ...])
```

- `REPLICA_NUM`：指定数据的副本份数。

```
replica_num:
  REPLICA_NUM [=] INT_VALUE
```

- `TABLEGROUP`：指定多份数据的主副本分布策略。

```
tablegroup:
    default_tablegroup
  | table_tablegroup

default_tablegroup:
  DEFAULT TABLEGROUP [=] {tablegroup_name | NULL}

table_tablegroup:
  TABLEGROUP [=] {tablegroup_name | NULL}
```

存储属性

- `BLOCK_SIZE`：指定对象存储的微块长度。

```
block_size:
  BLOCK_SIZE [=] INT_VALUE
```

- `COMPRESSION`：指定对象存储的压缩算法。

```
compression:
  COMPRESSION [=] {NONE | LZ4_1.0 | LZ0_1.0 | SNAPPY_1.0 | ZLIB_1.0}
```

- `PCTFREE`：指定对象存储的宏块预留百分比。

```
pctfree:
  PCTFREE [=] INT_VALUE
```

- `TABLET_SIZE`：指定并行合并时单个任务的最小分片。

```
tablet_size:
  TABLET_SIZE [=] INT_VALUE
```

5.5.2. ALTER DATABASE

描述

该语句用来修改 DataBase 的属性。

格式

```
alter_database_stmt:
    ALTER DATABASE [database_name] [SET] alter_specification_list;

alter_specification_list:
  alter_specification [alter_specification ...]

alter_specification:
  [DEFAULT] {CHARACTER SET | CHARSET} [=] charset_name
 | [DEFAULT] COLLATE [=] collation_name
 | REPLICA_NUM [=] int_num
 | PRIMARY_ZONE [=] zone_name
 | {READ ONLY | READ WRITE}
  | DEFAULT TABLEGROUP [=] {NULL | table_group_name}
```

参数解释

参数	描述
database_name	指定要修改属性的数据库名称。如果不指定，会对当前默认数据库进行修改。
CHARSET charset_name	指定要修改的字符集。
COLLATE collation_name	指定校对规则。
REPLICA_NUM int_num	指定副本数。
PRIMARY_ZONE zone_name	指定主 Zone。

参数	描述
READ ONLY | READ WRITE	设置数据库级只读或读写属性。
DEFAULT TABLEGROUP table_group_name	设置数据库默认表组信息，NULL 表示取消数据库默认表组。

示例

- 修改数据库 test2 的字符集为 UTF8MB4，校对规则为 UTF8MB4_BIN，且为读写属性。

```
OceanBase(admin@test)>alter database test2 DEFAULT CHARACTER SET UTF8MB4;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)>alter database test2 DEFAULT COLLATE UTF8MB4_BIN;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)>alter database test2 READ WRITE;
Query OK, 0 rows affected (0.02 sec)
```

5.5.3. ALTER OUTLINE

描述

该语句仅支持使用SQL_TEXT创建的 OUTLINE，可用于添加绑定OUTLINE和限流规则。

格式

```
ALTER OUTLINE outline_name ADD stmt [ TO target_stmt ]
```

参数解释

参数	描述
outline_name	指定要创建的 OUTLINE 名称。
stmt	一般为一个带有 hint 和原始参数的 DML 语句。

参数	描述
TO target_stmt	如果不指定 TO target_stmt， 则表示如果数据库接受的SQL参数化后与stmt去掉hint参数化文本相同，则将该SQL绑定stmt中hint生成执行计划；如果期望对含有hint的语句进行固定计划，则需要 TO target_stmt 来指明原始的SQL。 **注意** 在使用target_stmt时，严格要求stmt与target_stmt在去掉hint后完全匹配。

示例

- 通过ALTER OUTLINE来添加限流规则。

```
OceanBase (root@oceanbase)> alter outline ol_1 add select /*+max_concurrent(1)*/ * from t1 where c1 = 1 a
nd c2 = ?;
OceanBase (root@oceanbase)> alter outline ol_1 add select /*+max_concurrent(1)*/ * from t1 where c1 = ? a
nd c2 = 1;
```

- 通过ALTER OUTLINE来添加执行计划。

```
OceanBase (root@oceanbase)> create outline ol_2 on select /*+max_concurrent(1)*/ * from t1,t2 where t1.c
1 = 1;
OceanBase (root@oceanbase)> alter outline ol_2 add select /*+use_nl(t2)*/ * from t1,t2 where t1.c1 = 1;
```

注意事项

- 同一个outline_name 只能指定一个执行计划。如果通过create outline语句指定了执行计划，则无法通过执行alter outline时再添加。
- 同create outline类似，在alter outline时不能同时指定限流规则和执行计划。
- 执行alter outline时，需要outline_name和signature同时匹配。

5.5.4. ALTER RESOURCE POOL

描述

该语句用来修改资源池。

格式

```
ALTER RESOURCE POOL poolname
 UNIT [=] unitname,
 UNIT_NUM [=] unitnum,
 ZONE [=] ( ‘zone’ [, ‘zone’ …]);
```

参数解释

参数	描述
UNIT	指定要使用的资源单元配置的名称
UNIT_NUM	指定要创建的一个 ZONE 下的资源单元个数，取值要小于对应 ZONE 中的 OBServer 个数。
ZONE_LIST	指定要创建的资源单元的 ZONE 列表

示例

- 不能同时修改 RESOURCE POOL 的 unit, unit_num, zone_list 多个属性，执行将报错。

```
OceanBase(admin@test)> ALTER RESOURCE POOL pool1 unit='unit2', unit_num=1, zone_list=('zone1');
ERROR 1235 (0A000): alter unit_num, resource_unit, zone_list in one cmd not supported
```

- 修改 RESOURCE POOL 的单条 unit 属性。

```
OceanBase(admin@test)> ALTER RESOURCE POOL pool1 unit='unit2';
Query OK, 0 rows affected (0.00 sec)
```

5.5.5. ALTER RESOURCE UNIT

描述

该语句用来修改资源单元配置。

格式

```
ALTER RESOURCE UNIT unitname
 MAX_CPU [=] cpunum,
 MAX_MEMORY [=] memsize,
 MAX_IOPS [=] iopsnum,
 MAX_DISK_SIZE [=] disksize,
 MAX_SESSION_NUM [=] sessionnum,
 [MIN_CPU [=] cpunum,]
 [MIN_MEMORY [=] memsize,]
 [MIN_IOPS [=] iopsnum] ;
```

参数解释

参数	描述
MAX_CPU	指定要修改的 CPU 最多数量
MAX_MEMORY	指定要修改的最大内存容量，取值范围为[1073741824, +∞) 单位为字节，即最小值为1G
MAX_IOPS	指定要修改的 IOPS 最多数量，取值范围为[128,+∞)
MAX_DISK_SIZE	指定要修改的最大硬盘容量，取值范围为［536870912,+∞］单位为字节，即最小值为512M
MAX_SESSION_NUM	指定要修改的 session 最多数量，取值范围为[64,+∞)
MIN_CPU	指定要修改的 CPU 最少数量
MIN_MEMORY	指定要修改的最小内存容量
MIN_IOPS	指定要修改的 IOPS 最少数量

示例

- 修改资源单元 unit1 最高配置为 2个CPU，2G 的内存容量。

```
OceanBase(admin@test)> ALTER RESOURCE UNIT unit1 max_cpu 2, max_memory '2G';
Query OK, 0 rows affected (0.02 sec)
```

5.5.6. ALTER SYSTEM

ALTER SYSTEM 语句用来对 OceanBase 发送命令，执行某项指定操作。

BOOTSTRAP

描述

该语句用于自举 OceanBase 集群。

格式

```
alter_system_bootstrap_stmt:
  ALTER SYSTEM BOOTSTRAP opt_cluster_type region_zone_server_list;

opt_cluster_type:
  [CLUSTER cluster_role]

cluster_role:
  PRIMARY | STANDBY

region_zone_server_list:
  region_zone_server [, region_zone_server ...]

region_zone_server:
  [region] zone server

region:
  REGION [=] region_name

zone:
  ZONE [=] zone_name

server:
  SERVER [=] ip_port

ip_port:
  'STR_VALUE:INT_VALUE'
```

参数解释

系统自举时，需要指定 RootService 信息，多台 RootService 通过逗号（，）分隔。

参数	描述
region_name	指定 RootService 所在的地域，通过在“多地多中心”部署时指定。
zone_name	指定 RootService 所属的Zone。
ip_port	指定 RootService 的IP和端口。
PRIMARY | STANDBY	指定主备库。系统启动时需要指定主备库，如果不指定，默认采用主库启动。

示例

- 指定一台 RootService：

```
ALTER SYSTEM BOOTSTRAP ZONE 'zone1' SERVER '10.218.248.178:55410';
```

- 多台RootService通过逗号（，）分隔：

```
ALTER SYSTEM BOOTSTRAP ZONE 'zone1' SERVER '172.24.65.24:55410', ZONE 'zone2'
SERVER '172.24.65.114:55410';
```

- 启动备库

```
ALTER SYSTEM BOOTSTRAP CLUSTER STANDBY ZONE 'zone1' SERVER '10.218.248.178:55410';
```

JOB

描述

该语句触发参数指定的后台任务。

格式

```
alter_system_job_stmt:
  ALTER SYSTEM RUN JOB job_name
  [zone | server];
```

参数解释

参数	描述
JOB job_name	指定任务名称。有特殊字符时需要单引号，其它情况下单引号加与不加都支持，已支持的 JOB 有： • check_partition_table：OBServer 执行 partition table 核对删除任务 • root_inspection：触发 RootService 自检
zone | server	该任务指定 Zone 或者 Server 执行。

示例

- 触发 RootService 自检任务。

```
ALTER SYSTEM RUN JOB "root_inspection";
```

MERGE

描述

该语句发起存储层合并。

格式

```
alter_system_merge_stmt:
  ALTER SYSTEM merge_action;

merge_action:
    MAJOR FREEZE
  | MINOR FREEZE
    [tenant_list | replica] [server_list] [zone]
  | START MERGE
    zone
  | {SUSPEND | RESUME} MERGE
    [zone]
  | CLEAN MERGE ERROR

tenant_list:
  TENANT [=] (tenant_name_list)

tenant_name_list:
  tenant_name [, tenant_name ...]

replica:
  PARTITION_ID [=] 'partition_id%partition_count@table_id'

server_list:
  SERVER [=] ip_port_list
```

参数解释

参数	描述
MAJOR FREEZE	发起每日合并。
MINOR FREEZE	发起转储。
START MERGE	开始每日合并。
{SUSPEND | RESUME} MERGE	暂停或恢复每日合并。

参数	描述
CLEAN MERGE ERROR	清理合并出错的标记。
tenant_name	指定租户转储。
PARTITION_ID	指定副本转储。
SERVER	指定机器转储。
zone	指定Zone开始合并。

示例

- 发起每日合并。

```
OceanBase(root@oceanbase)>alter system major freeze;
Query OK, 0 rows affected (0.06 sec)
```

PARAMETER

描述

该语句用来修改配置项信息。

格式

```
alter_system_parameter_stmt:
  ALTER SYSTEM [SET]
  parameter_name = expression [SCOPE = {MEMORY | SPFILE | BOTH}] [COMMENT [=] 'text']
                              {SERVER [=] 'ip:port' | ZONE [=] 'zone' };
```

参数解释

参数	描述
parameter_name	指定要修改的配置项名称。
expression	指定修改后的配置项的值。

参数	描述
COMMENT 'text'	用于添加关于本次修改的注释。该参数为可选，建议不要省略。
SCOPE	指定本次配置项修改的生效范围。它的值主要有以下三种： • MEMORY：表明只修改内存中的配置项，修改立即生效，且本修改在 Server 重启以后会失效（目前暂时没有配置项支持这种方式）。 • SPFILE：表明只修改配置表中的配置项值，当 Server 重启以后才生效。 • BOTH：表明既修改配置表，又修改内存值，修改立即生效，且 Server • 重启以后配置值仍然生效。 SCOPE 默认值为 BOTH。对于不能立即生效的配置项，如果使用 BOTH 或 MEMORY，会报错 。
SERVER	只修改指定 Server 实例的某个配置项。
ZONE	表明本配置项的修改针对指定集群的特定 Server 类型，否则，针对所有集群的特定 Server 类型。

说明

同时修改多个系统配置项时，用“,”隔开。

查看系统配置项的格式：`SHOW PARAMETERS [LIKE 'pattern' | WHERE expr];`

示例

- 修改配置项 enable_sql_audit 的信息。

```
OceanBase(root@oceanbase)>show parameters like 'enable_sql_audit';
+------+----------+---------------+----------+-----------------+-----------+-------+-----------------------------------------------------------------
-----------------------------------------------------------+----------+---------+---------+------------------+
| zone | svr_type | svr_ip      | svr_port | name          | data_type | value | info
| section  | scope   | source  | edit_level      |
+------+----------+---------------+----------+-----------------+-----------+-------+-----------------------------------------------------------------
-----------------------------------------------------------+----------+---------+---------+------------------+
| z1   | observer | 11.11.111.111 |   19510 | enable_sql_audit | NULL      | True  | specifies whether SQL audit is tur
ned on. The default value is TRUE. Value: TRUE: turned on FALSE: turned off | OBSERVER | CLUSTER | DEFAULT
| DYNAMIC_EFFECTIVE |
+------+----------+---------------+----------+-----------------+-----------+-------+-----------------------------------------------------------------
-----------------------------------------------------------+----------+---------+---------+------------------+
1 rows in set (0.02 sec)

OceanBase(root@oceanbase)>alter system set enable_sql_audit = false scope = BOTH;
Query OK, 0 rows affected (0.05 sec)

OceanBase(root@oceanbase)>show parameters like 'enable_sql_audit';
+------+----------+---------------+----------+-----------------+-----------+-------+-----------------------------------------------------------------
-----------------------------------------------------------+----------+---------+---------+------------------+
| zone | svr_type | svr_ip      | svr_port | name          | data_type | value | info
| section  | scope   | source  | edit_level      |
+------+----------+---------------+----------+-----------------+-----------+-------+-----------------------------------------------------------------
-----------------------------------------------------------+----------+---------+---------+------------------+
| z1   | observer | 11.11.111.111 |   19510 | enable_sql_audit | NULL      | False | specifies whether SQL audit is tur
ned on. The default value is TRUE. Value: TRUE: turned on FALSE: turned off | OBSERVER | CLUSTER | DEFAULT
| DYNAMIC_EFFECTIVE |
+------+----------+---------------+----------+-----------------+-----------+-------+-----------------------------------------------------------------
-----------------------------------------------------------+----------+---------+---------+------------------+
1 rows in set (0.02 sec)
```

REFRESH

描述

该语句系列用来触发刷新操作，包括刷新 schema、时区信息、各类缓存等。

格式

```
alter_system_refresh_stmt:
  ALTER SYSTEM refresh_action;

refresh_action:
    REFRESH SCHEMA
    [zone | server]
  | REFRESH TIME_ZONE_INFO
  | FLUSH cache_type CACHE
    [tenant_list] [GLOBAL]
  | FLUSH KVCACHE
    [tenant] [CACHE [=] cache_name]

cache_type:
    ALL
  | LOCATION
  | CLOG
  | ILOG
  | COLUMN_STAT
  | BLOCK_INDEX
  | BLOCK
  | ROW
  | BLOOM_FILTER
  | SCHEMA
  | PLAN

tenant:
  TENANT [=] tenant_name

cache_name:
```

参数解释

参数	描述
REFRESH SCHEMA	刷新 Schema。正常情况下，当系统执行 DDL 操作，RootService 都会主动通知所有 OBServer 刷新 Schema。当某些 OBServer 发生异常，和 RootService 网络中断，此时需要手动刷新 Schema。刷新Schema 可以针对单独的机器，也可以针对某个集群。

参数	描述
REFRESH TIME_ZONE_INFO	通知集群中的所有 Server 更新本地的时区信息。
FLUSH cache_type CACHE	清空指定 Cache 指令。
FLUSH KVCACHE	清空 KV Cache 指令。 • 如果指定 tenant 和 cache_name，则清除指定租户的指定 KV Cache • 如果只指定 tenant，则清除指定租户的全部KV Cache • 如果不指定 tenant 和 cache_name，则清除所有租户的全部KV Cache

示例

- 刷新单台 OBServer 的 Schema：

```
ALTER SYSTEM REFRESH SCHEMA SERVER='172.24.65.24:55410';
```

- 刷新整个 Zone 的 Schema：

```
ALTER SYSTEM REFRESH SCHEMA ZONE='zone1';
```

REPLICA

描述

该语句用来处理副本的迁移，复制，删除，副本类型转换，角色切换等行为。

格式

```
alter_system_replica_stmt:
  ALTER SYSTEM replica_action;

replica_action:
    SWITCH REPLICA
    {LEADER | FOLLOWER}
    {replica server | server [tenant_name] | zone [tenant_name]}
  | DROP REPLICA partition_id_desc
    replica server [create_timestamp] [zone] [FORCE]
  | {MOVE | COPY} REPLICA
    replica source destination
  | REPORT REPLICA partition_id_desc
            {zone | server}
      | RECYCLE REPLICA partition_id_desc
            {zone | server}
      | {ALTER | CHANGE | MODIFY} REPLICA
    replica server [set] REPLICA_TYPE = replica_type

source:
  SOURCE [=] 'ip:port'

destination:
  DESTINATION [=] 'ip:port'

partition_id_desc
  PARTITION_ID partition_id%partition_count@table_id

partition_idx | partition_count | table_id | task_id:
  INT_VALUE

create_timestamp:
  CREATE_TIMESTAMP [=] INT_VALUE

tenant_name_list:
  tenant_name [, tenant_name ...]

replica_type:
    {FULL | F}
  | {READONLY | R}
  | {LOGONLY | L}
```

参数解释

参数	描述
SWITCH REPLICA	Leader改选。
DROP REPLICA	副本删除。删除指定 OBServer 上的 Replica，必须指定 partition_id，OBServer 地址以及 create_timestamp。
{MOVE | COPY} REPLICA	副本迁移/复制，需要指定源 OBServer 和目的 OBServer，以及partition_id。
REPORT REPLICA	副本汇报，强制要求某个 OBServer 或者某个 Zone 内的所有 OBServer 进行 Replica 汇报。
RECYCLE REPLICA	回收无用副本。
{ALTER | CHANGE | MODIFY} REPLICA	修改副本属性，可以修改指定副本的副本类型。现在已经支持的副本类型包括：FULL/READONLY/LOGONLY。'replica_type'的值可以是这三种副本类型的全称或者简写的首字符：'F/R/L'；大小写通用。

示例

- 副本迁移

```
ALTER SYSTEM MOVE REPLICA PARTITION_ID '0%4@1100611139403777'
SOURCE '172.24.65.24:55410'
DESTINATION '172.24.65.26:55410';
```

- 删除副本

```
ALTER SYSTEM DROP REPLICA PARTITION_ID '0%4@1100611139403777'
SERVER '172.24.65.26:55410';
```

- 修改副本类型

```
ALTER SYSTEM CHANGE REPLICA PARTITION_ID '0%4@1100611139403777'
SERVER '172.24.65.26:55410';
CHANGE REPLICA_TYPE = 'L';
```

- Leader改选

```
ALTER SYSTEM SWITCH REPLICA LEADER PARTITION_ID '0%4@1100611139403777'
SERVER '172.24.65.26:55410';
```

ROOTSERVICE

描述

切换 RS 的角色。

格式

```
alter_system_rootservice_stmt:
  ALTER SYSTEM SWITCH ROOTSERVICE {LEADER | FOLLOWER} {zone | server};
```

参数解释

参数	描述
LEADER | FOLLOWER	把 RS 的角色设置为 Leader 或者 Follower。
zone | server	修改指定 Zone 或 Server 的 RS 角色。

示例

- 把z1的 RS 切换为 Leader

```
ALTER SYSTEM SWITCH ROOTSERVICE LEADER ZONE 'z1';
```

SERVER

描述

该语句维护系统中 OBServer 状态，控制 OBServer 的增、删、启、停等操作。

格式

```
alter_system_server_stmt:
  ALTER SYSTEM server_action SERVER ip_port_list [zone];

server_action:
    ADD
  | DELETE
  | CANCEL DELETE
  | START
  | STOP
  | FORCE STOP

ip_port_list:
  ip_port [, ip_port ...]
```

参数解释

服务器状态图如下：



参数	描述
ip_port	指定 OBServer 的 IP 地址和端口
zone	如果指定 Zone，将会对待维护 Server 执行 Zone 校验。
ADD	新增服务器。

参数	描述
DELETE	删除服务器。
CANCEL DELETE	取消删除服务器。
START	启动服务器。
STOP	停止服务器。
FORCE STOP	强制停止服务器。

示例

- 新增服务器

```
ALTER SYSTEM ADD SERVER '172.24.65.113:55410' ZONE 'zone1';
```

> 注意
>
> - 通过Add和Delete命令将服务器加入到服务列表，只有服务列表中的服务器才可以提供服务。
> - Delete操作执行leader改选以及replica复制。
> - Delete操作执行时间很长，因此，允许通过Cancel命令取消该操作。

THROTTLE

描述

该语句用来设置特征限流。

格式

```
alter_system_throttle_stmt:
  ALTER SYSTEM throttle_action;

throttle_action:
   ENABLE SQL THROTTLE [priority_option] [using_metric_option_list]
  | DISABLE SQL

priority_option:
  FOR PRIORITY <= INT_VALUE

using_metric_option_list:
  USING metric_option_list

metric_option_list:
  metric_option [metric_option ...]

metric_option:
   RT = {INT_VALUE | DECIMAL_VALUE}
  | CPU = {INT_VALUE | DECIMAL_VALUE}
  | IO = INT_VALUE
  | NETWORK = {INT_VALUE | DECIMAL_VALUE}
  | QUEUE_TIME = {INT_VALUE | DECIMAL_VALUE}
  | LOGICAL_READS = {INT_VALUE | DECIMAL_VALUE}
```

参数解释

参数	描述
ENABLE SQL	开启限流，后接限流特征及特征值
FOR PRIORITY	设定优先级，目的是只在部分session生效
RT	按请求执行时间限流
CPU	按CPU占用限流，暂未支持
IO	按IO次数限流，暂未支持

参数	描述
NETWORK	按传输的网络流量大小限流，暂未支持
QUEUE_TIME	按队列等待时间限流
LOGICAL_READS	按逻辑读次数限流，暂未支持
DISABLE SQL	关闭限流

示例

- 对prioriy<=100的session上队列等待时间超过0.1s的请求限流。

```
alter system enable sql throttle for priority <= 100 using queue_time=0.1
```

UNIT

描述

该语句负责 UNIT 资源迁移。

格式

```
alter_system_unit_stmt:
  ALTER SYSTEM MIGRATE
  UNIT [=] unit_id DESTINATION [=] ip_port

unit_id:
  INT_VALUE
```

参数解释

参数	描述
unit_id	UNIT 编号。
ip_port	将 UNIT 迁移到的目标 Server 地址。

示例

- 将 UNIT 1001的资源迁移到 11.11.111.111:19510。

```
OceanBase(root@oceanbase)>alter system migrate unit = 1001 destination = '11.11.111.111:19510';
Query OK, 0 rows affected (0.05 sec)
```

ZONE

描述

该语句负责维护 Zone 的状态，如 Zone 的增、删、启、停等操作。

格式

```
alter_system_zone_stmt:
    ADD ZONE zone_name
    [zone_option_list]
  | {ALTER | CHANGE | MODIFY} ZONE zone_name
    [SET] zone_option_list
  | {DELETE | START | STOP | FORCE STOP} ZONE zone_name

zone_option_list:
  zone_option [, zone_option ...]

zone_option:
    region
  | idc
  | ZONE_TYPE {READONLY | READWRITE}

idc:
  STR_VALUE
```

参数解释

参数	描述
ADD ZONE	新增Zone
{ALTER | CHANGE | MODIFY} ZONE	修改Zone的Region属性
DELETE ZONE	删除Zone，删除前需要保证Zone中没有可用的Server
START | STOP	主动上下线Zone

示例

- 删除Zone。

```
OceanBase(root@oceanbase)>alter system delete zone 'z1';
ERROR 4668 (HY000): The zone is not empty and can not be deleted. You should delete the servers of the zon
e. There are 1 servers alive and 0 not alive.
```

CLUSTER

描述

集群管理相关操作，可以新增、删除、修改集群级相关属性。

格式

```
ALTER SYSTEM  cluster_action cluster_name CLUSTER_ID INTNUM;
cluster_action:
  ADD CLUSTER
  | REMOVE CLUSTER
  | ENABLE CLUSTER SYNCHRONIZATION
  | DISABLE CLUSTER SYNCHRONIZATION
```

参数解释

参数	描述
ADD CLUSTER	新增一个备库。
REMOVE CLUSTER	删除一个已经存在的备库。
ENABLE CLUSTER SYNCHRONIZATION	允许一个备库的同步。
DISABLE CLUSTER SYNCHRONIZATION	禁掉一个备库的同步。

示例

- 新增备库

```
ALTER SYSTEM ADD CLUSTER 'ob1.test' cluster_id = 1;
```

SWITCHOVER

描述

集群无损、有损切换语法，包括主库切成备库，备库切成主库。

格式

```
ALTER SYSTEM commit_switchover_clause;

commit_switchover_clause:
  COMMIT TO SWITCHOVER TO PRIMARY
  | COMMIT TO SWITCHOVER TO PHYSICAL STANDBY
  | ACTIVATE PHYSICAL STANDBY CLUSTER
  | CONVERT TO PHYSICAL STANDBY
```

参数解释

参数	描述
COMMIT TO SWITCHOVER TO PRIMARY	备库切成主库，在备库上执行，并且旧主库已经切换成主库，集群中没有其他的备库先切换成主库。
COMMIT TO SWITCHOVER TO PHYSICAL STANDBY	主库无损切换成备库，在主库上执行，并且备库已经同步成功。
ACTIVATE PHYSICAL STANDBY CLUSTER	在主库宕机的情况下，把一个集群拉成主库。
CONVERT TO PHYSICAL STANDBY	把一个主库切换成备库。

示例

- 主库切换成备库

```
ALTER SYSTEM COMMIT TO SWITCHOVER TO PHYSICAL STANDBY;
```

- 备库切换成主库

```
ALTER SYSTEM COMMIT TO SWITCHOVER TO PRIMARY;
```

- 主库宕机，备库切换成主库

```
ALTER SYSTEM ACTIVATE PHYSICAL STANDBY CLUSTER;
```

- 旧主库重启，切成备库

```
ALTER SYSTEM CONVERT TO PHYSICAL STANDBY;
```

BALANCE TASK

描述

该语句用来清除没有正在调度的负载均衡任务。

格式

```
ALTER SYSTEM REMOVE BALANCE TASK opt_tenant_list opt_zone_list opt_balance_task_type;

opt_tenant_list
  TENANT [=] name,name_list

opt_zone_list
  ZONE [=] zone_name, zone_list

opt_balance_task_type
  ALL
  | MANUAL
  | AUTO
```

参数解释

参数	描述
opt_tenant_list	租户列表，如果不指定，就会清除所有租户的任务。
opt_zone_list	Zone的列表，可不指定。
opt_balance_task_type	待清除的任务类型： • ALL：所有的任务； • AUTO：自动生成的任务； • MANUAL：用户手动发起的任务。

示例

- 清除所有租户所有没有在调度的任务。

```
ALTER SYSTEM REMOVE BALANCE TASK;
```

CANCEL MIGRATE UNIT

描述

该语句用来取消 UNIT 的迁移。

格式

```
ALTER SYSTEM CANCEL MIGRATE UNIT unit_id;
```

参数解释

参数	描述
unit_id	指定迁移的 UNIT 的 ID。

示例

- 取消 UNIT 1001 的迁移。

```
ALTER SYSTEM CANCEL MIGRATE UNIT 1001;
```

RESTORE

描述

该语句用来恢复租户的数据。

格式

```
alter system restore dest_tenant from source_tenant at 'uri' until 'timestamp' with 'restore_option';
```

参数解释

参数	描述
dest_tenant	指恢复的新租户的名字。
source_tenant	指原集群的租户。
uri	指备份的时候设置的backup_dest。
timestamp	恢复的时间戳，需要大于等于最早备份的基线备份的CDB_OB_BACKUP_SET_DETAILS的START_TIME，小于等于日志备份 CDB_OB_BACKUP_ARCHIVELOG_SUMMARY 的 MAX_NEXT_TIME。

参数	描述
restore_option	恢复选项，支持以下几种： • backup_cluster_name：源集群的名字，必选项 • backup_cluster_id：源集群的cluster_id，必选项 • pool_list：用户的资源池，必选项 • locality：租户的locality信息，可选项 • kms_encrypt：为 true 表示需要恢复的时候指定 kms_encrypt_info，可选项

示例

- 恢复租户的数据。

```
alter system restore restored_trade from trade
  at 'oss://antsys-oceanbasebackup/backup_rd/20200323?host=cn-hangzhou-alipay-b.oss-cdn.aliyun-inc.com&access_id=xxx&access_key=xxx'
 until ' 2020-03-23 08:59:45'
 with 'backup_cluster_name=ob20daily.backup&backup_cluster_id=1&pool_list=restore_pool';

alter system restore restored_trade from trade
 at 'file:///data/nfs/physical_backup_test/20200520'
 until '2020-05-21 09:39:54.071670'
 with 'backup_cluster_name=ob20daily.backup&backup_cluster_id=1&pool_list=restore_pool&locality=F@z1,F@z2,F@z3';
```

CHANGE TENANT

描述

该语句用来切换租户。

格式

```
ALTER SYSTEM CHANGE TENANT tenant_name;

ALTER SYSTEM CHANGE TENANT TENANT_ID [=] INTNUM;
```

参数解释

参数	描述
tenant_name	指定要切换到的租户名称。
TENANT_ID	指定要切换到的租户ID。

示例

- 切换至ID为1001的租户。

```
ALTER SYSTEM CHANGE TENANT TENANT_ID = 1001;
```

注意事项

1. 需以系统租户身份登陆，普通租户无法执行该命令。
2. 需直连OBServer执行，断连接后需重新执行。
3. 不能在事务执行过程中执行该命令。
4. 切换到非系统租户后，无法执行DDL操作。

BACKUP

描述

该语句用来触发备份。

格式

```
设置备份的介质路径： alter system set backup_dest = <backup_uri>
开启logarchive的SQL： alter system archivelog
关闭logarchive的SQL: alter system noarchivelog
触发基线备份（只支持集群级别）： alter system backup database;
取消当前备份: alter system cancel backup
```

参数解释

参数	描述
backup_uri	指定备份的路径，目前支持oss和file两种，具体格式参考示例。

示例

- 设置备份的路径。

```
alter system set backup_dest='oss://antsys-oceanbasebackup/backup_dir?host=xxx&access_id=xxx&access_key=xxx';
alter system set backup_dest='file:///data/nfs/physical_backup_dir';
```

注意事项

需以系统租户身份登陆，普通租户无法执行该命令。

5.5.7. ALTER TABLE

描述

该语句用来修改已存在的表的结构，比如：修改表及表属性、新增列、修改列及属性、删除列等。

格式

```
alter_table_stmt:
    ALTER TABLE table_name
    alter_table_action_list;
  | RENAME TABLE rename_table_action_list;

alter_table_action_list:
  alter_table_action [, alter_table_action ...]

alter_table_action:
    ADD [COLUMN] {column_definition | (column_definition_list)}
  | CHANGE [COLUMN] column_name column_definition
  | MODIFY [COLUMN] column_definition
  | ALTER [COLUMN] column_name {SET DEFAULT const_value | DROP DEFAULT}
  | DROP [COLUMN] column_name
  | ADD [CONSTRAINT [constraint_name]] UNIQUE {INDEX | KEY} [index_name] index_desc
  | ADD {INDEX | KEY} [index_name] index_desc
  | ADD FULLTEXT [INDEX | KEY] [index_name] fulltext_index_desc
  | ALTER INDEX index_name [VISIBLE | INVISIBLE]
  | DROP {INDEX | KEY} index_name
  | ADD PARTITION (range_partition_list)
  | DROP PARTITION (partition_name_list)
  | REORGANIZE PARTITION name_list INTO partition_range_or_list
  | TRUNCATE PARTITION name_list
  | [SET] table_option_list
  | RENAME [TO] table_name
  | DROP TABLEGROUP
  | DROP FOREIGN KEY fk_name
```

```
rename_table_action_list:
  rename_table_action [, rename_table_action ...]

rename_table_action:
  table_name TO table_name

column_definition_list:
  column_definition [, column_definition ...]

column_definition:
  column_name data_type
  [DEFAULT const_value] [AUTO_INCREMENT]
  [NULL | NOT NULL] [[PRIMARY] KEY] [UNIQUE [KEY]] comment

index_desc:
 (column_desc_list) [index_type] [index_option_list]

fulltext_index_desc:
 (column_desc_list) CTXCAT(column_desc_list) [index_option_list]

column_desc_list:
  column_desc [, column_desc ...]

column_desc:
   column_name [(length)] [ASC | DESC]

index_type:
  USING BTREE

index_option_list:
  index_option [ index_option ...]

index_option:
    [GLOBAL | LOCAL]
  | block_size
  | compression
  | STORING(column_name_list)
  | comment

table_option_list:
```

```
  table_option [ table_option ...]

table_option:
  | primary_zone
  | replica_num
  | table_tablegroup
  | block_size
  | compression
  | AUTO_INCREMENT [=] INT_VALUE
  | comment
  | DUPLICATE_SCOPE [=] "none|zone|region|cluster"

partition_option:
    PARTITION BY HASH(expression)
    [subpartition_option] PARTITIONS partition_count
  | PARTITION BY KEY([column_name_list])
    [subpartition_option] PARTITIONS partition_count
  | PARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
    [subpartition_option] (range_partition_list)

subpartition_option:
    SUBPARTITION BY HASH(expression)
    SUBPARTITIONS subpartition_count
  | SUBPARTITION BY KEY(column_name_list)
    SUBPARTITIONS subpartition_count
  | SUBPARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
    (range_subpartition_list)

range_partition_list:
  range_partition [, range_partition ...]

range_partition:
  PARTITION partition_name
  VALUES LESS THAN {(expression_list) | MAXVALUE}

range_subpartition_list:
  range_subpartition [, range_subpartition ...]

range_subpartition:
  SUBPARTITION subpartition_name
  VALUES LESS THAN {(expression_list) | MAXVALUE}
```

```
  VALUES LESS THAN {(expression_list) | MAXVALUE}

expression_list:
  expression [, expression ...]

column_name_list:
  column_name [, column_name ...]

partition_name_list:
  partition_name [, partition_name ...]

partition_count | subpartition_count:
  INT_VALUE
```

参数解释

参数	描述
ADD [COLUMN]	增加列，目前不支持增加主键列。
CHANGE [COLUMN]	修改列名和列属性。
MODIFY [COLUMN]	修改列属性。
ALTER [COLUMN]	修改列的默认值。
DROP [COLUMN]	删除列，不允许删除主键列或者包含索引的列。
ADD [UNIQUE INDEX]	增加唯一索引。
ADD [INDEX]	增加普通索引
ALTER [INDEX]	修改索引属性。
ADD [PARTITION]	增加分区。
DROP [PARTITION]	删除分区。

参数	描述
REORGANIZE [PARTITION]	分区重组。
TRUNCATE [PARTITION]	删除分区数据。
RENAME [TO] table_name	表重命名。
DROP [TABLEGROUP]	删除表组。
DROP [FOREIGN KEY]	删除外键。
SET BLOCK_SIZE	设置Partition表BLOCK大小。
SET REPLICA_NUM	设置表的副本数（指表的副本总数）。
SET COMPRESSION	设置表的压缩方式。
SET USE_BLOOM_FILTER	设置是否使用BloomFilter。
SET COMMENT	设置注释信息。
SET PROGRESSIVE_MERGE_NUM	设置渐进合并步数，取值范围是1~64。

示例

- 把表t2的字段d改名为c，并同时修改字段类型

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

- 增加、删除列
  - 增加列前，执行 `DESCRIBE test;` 命令查看表信息，如下图所示：

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

  - 执行以下命令增加c3列

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

- 增加列后，执行 `DESCRIBE test;` 命令查看表信息，如下图所示：

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

- 执行以下命令删除c3列

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

- 删除列后，执行 `DESCRIBE test;` 命令查看表信息，如下图所示：

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

- 设置表格test的副本数，并且增加列c5

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

5.5.8. ALTER TABLEGROUP

描述

该语句用来执行以下操作：

- 对一个表组增加多张表。
- 修改表组的分区规则。
- 修改表组的 locality 和 primary zone。

格式

- 对一个表组增加多张表。

```
ALTER TABLEGROUP tablegroupname ADD [TABLE] tblname [, tblname…]
```

- 修改表组的分区规则。

```
ALTER TABLEGROUP tablegroupname alter_tg_partition_option
alter_tg_partition_option:
DROP PARTITION '(' name_list ')'
| ADD PARTITION opt_range_partition_list
| modify_tg_partition_info
```

- 修改表组的 locality 和 primary zone。

```
ALTER TABLEGROUP tablegroupname alter_tablegroup_actions
alter_tablegroup_actions:
alter_tablegroup_action
| alter_tablegroup_action, alter_tablegroup_action
alter_tablegroup_action:
SET LOCALITY [=] locality_name
|SET PRIMARY_ZONE [=] primary_zone_name
```

参数解释

参数	描述
tablegroupname	指定表组。
tblname	表名。对一个表组增加多张表时，表与表之间以逗号（‘，’）分隔。 当添加多个表时，允许表名重复。如果待添加的表已经属于 *tablegroupname* 的表格组，系统不报错。
modify_tg_partition_info	修改表组的分区规则。
LOCALITY locality_name	指定表组的 locality。
PRIMARY_ZONE primary_zone_name	指定表组的主Zone。

示例

创建表组 tgh 和两张关系表 ttgh 和 ttgh2 ，并将表组的 locality 修改为 F@z1。

```
OceanBase(admin@test)> create tablegroup tgh locality='F,R{ALL_SERVER}@z1' partition by hash partitions
10;
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> create table ttgh(c1 int, c2 int) tablegroup = tgh locality='F,R{ALL_SERVER}@z1';
Query OK, 0 rows affected (0.55 sec)

OceanBase(admin@test)> create table ttgh2(c1 int, c2 int) tablegroup = tgh locality='F,R{ALL_SERVER}@z1';
Query OK, 0 rows affected (0.39 sec)

OceanBase(admin@test)> alter tablegroup tgh set locality ='F@z1';
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> select locality from oceanbase.__all_tablegroup where tablegroup_name ='tgh';
+------------+
| locality   |
+------------+
| FULL{1}@z1 |
+------------+
1 row in set (0.05 sec)

OceanBase(admin@test)> select locality from oceanbase.__all_table where tablegroup_id=(select tablegrou
p_id from oceanbase.__all_tablegroup where tablegroup_name ='tgh');
+------------+
| locality   |
+------------+
| FULL{1}@z1 |
| FULL{1}@z1 |
+------------+
2 rows in set (0.04 sec)
```

5.5.9. ALTER TENANT

描述

该语句用来修改租户信息。

格式

```
ALTER TENANT {tenant_name | ALL}
  [SET] [tenant_option_list] [opt_global_sys_vars_set]

tenant_option_list:
  tenant_option [, tenant_option ...]

tenant_option:
        COMMENT [=]'string'
        |{CHARACTER SET | CHARSET} [=] charsetname
        |COLLATE [=]  collationname
        |REPLICA_NUM [=] num
        |ZONE_LIST [=] (zone [, zone…])
        |PRIMARY_ZONE [=] zone
        |RESOURCE_POOL_LIST [=](poolname [, poolname…])
        |DEFAULT TABLEGROUP [=] {NULL | tablegroupname}
        |{READ ONLY | READ WRITE}
    |LOGONLY_REPLICA_NUM [=] num
    |LOCALITY [=] 'locality description'
    |LOCK|UNLOCK;

opt_global_sys_vars_set:
    VARIABLES system_var_name = expr [,system_var_name = expr] ...
```

参数解释

参数	描述
tenant_name	指定要修改的租户名。
ALTER TENANT ALL	同时修改所有租户。
RESOURCE_POOL_LIST	资源池列表，为创建租户时的必填项，暂时只支持一个RESOURCE POOL。
DEFAULT TABLEGROUP	设置租户默认表组信息，NULL 表示取消ovtn库默认表组。

参数	描述
LOCK|UNLOCK	租户锁定和解锁。对租户进行锁定后，不能在该租户上创建新的连接，已有连接保持不变。通常在客户费用到期未续费的场景下使用，客户费用到期后对租户进行锁定，客户续费后再进行解锁。
COMMENT	修改注释。
CHARACTER SET | CHARSET	修改租户的字符集。
COLLATE	指定校对规则。
REPLICA_NUM	指定副本数。
ZONE_LIST	指定要修改的Zone列表。
PRIMARY_ZONE	指定主Zone。
READ ONLY | READ WRITE	指定租户的只读或读写属性。
LOGONLY_REPLICA_NUM	指定日志副本数。
LOCALITY	描述副本在Zone间的分布情况，如：F@z1,F@z2,F@z3,R@z4 表示z1, z2, z3为全功能副本，z4为只读副本。
system_var_name	修改租户的系统变量值。

示例

- 锁定租户TENANT1。

```
ALTER TENANT TENANT1 LOCK;
```

注意事项

ALTER TENANT语句系统租户有权限执行，同时本租户的管理员能执行。

5.5.10. ALTER USER

描述

该语句主要用于执行以下操作：

- 修改 OceanBase 用户的密码。
- 锁定或者解锁用户，且被锁定的用户不允许登录。

> **说明**
>
> 必须拥有全局 UPDATE USER 的权限，才可以执行本命令。

格式

- 修改用户密码

```
ALTER USER 'username' IDENTIFIED BY 'password';
```

- 锁定用户

```
ALTER USER user [lock_option]

lock_option:{
ACCOUNT LOCK
| ACCOUNT UNLOCK}
```

参数解释

参数	描述
ACCOUNT UNLOCK	锁定用户。
ACCOUNT UNLOCK	解锁用户。

示例

- 修改密码

执行以下命令将用户 sqluser01 的密码修改为 abc123。

```
ALTER USER 'sqluser01' IDENTIFIED BY 'abc123';
```

- 锁定用户

锁定用户 obsqluser01。

```
ALTER USER 'obsqluser01' ACCOUNT LOCK;
```

- 解锁用户

解锁用户 obsqluser01。

```
ALTER USER 'obsqluser01' ACCOUNT UNLOCK;
```

5.5.11. CREATE DATABASE

描述

该语句用于创建数据库，并可以指定数据库的默认属性（如数据库默认字符集，校验规则等）。

格式

```
create_database_stmt:
  CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] database_name [database_option_list]

database_option_list:
  database_option [database_option ...]

database_option:
  [DEFAULT] {CHARACTER SET | CHARSET} [=] charset_name
 | [DEFAULT] COLLATE [=] collation_name
 | REPLICA_NUM [=] int_num
 | PRIMARY_ZONE [=] zone_name
 | {READ ONLY | READ WRITE}
 | DEFAULT TABLEGROUP [=] {NULL | table_group_name}
```

参数解释

参数	描述
database_name	指定要修改属性的数据库名称。如果不指定，会对当前默认数据库进行修改。
CHARSET charset_name	指定要修改的字符集。
COLLATE collation_name	指定校对规则。
REPLICA_NUM int_num	指定副本数。
PRIMARY_ZONE zone_name	指定主Zone。

参数	描述
READ ONLY | READ WRITE	设置数据库级只读或读写属性。
DEFAULT TABLEGROUP table_group_name	设置数据库默认表组信息，NULL表示取消数据库默认表组。

示例

- 创建数据库 test2，并指定字符集为UTF8。

```
OceanBase(admin@test)>create database test2 default CHARACTER SET UTF8;
Query OK, 1 row affected (0.00 sec)
```

- 创建读写属性的数据库test3。

```
OceanBase(admin@test)>create database test3 READ WRITE;
Query OK, 1 row affected (0.03 sec)
```

5.5.12. CREATE INDEX

描述

该语句用来创建索引。索引是创建在表上的，对数据库表中一列或多列的值进行排序的一种结构。其作用主要在于提高查询的速度，降低数据库系统的性能开销。

格式

```
CREATE [UNIQUE] INDEX indexname
            ON tblname (index_col_name,...)
    [index_type] [index_options]
index_type:
    USING BTREE

index_options:
    index_option [index_option…]

index_option:
  GLOBAL | LOCAL
  | COMMENT 'string'
  | COMPRESSION [=] {NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}
  | BLOCK_SIZE [=] size
  | STORING(columname_list)
  | VISIBLE | INVISIBLE

index_col_name:
  colname [(length)] [ASC | DESC]

columname_list:
  colname [, colname…]
```

参数解释

参数	描述
indexname	指定要创建的索引名称。
tblname	指过索引所属的表名。
index_col_name	指定索引的列名，每个列名后都支持ASC（升序），不支持DESC（降序）。默认为升序。 建立索引的排序方式为：首先以index_col_name中第一个列的值排序；该列值相同的记录，按下一列名的值排序；以此类推。
index_type	索引类型，只支持USING BTREE，以B树为索引。

参数	描述
UNIQUE	指定为唯一索引。
index_option	指定索引选项，多个index_option以空格分隔。
GLOBAL | LOCAL	指定该索引是全局索引或局部索引，默认是GLOBAL。
COMMENT	指定注释。
COMPRESSION	指定压缩算法。
BLOCK_SIZE	指定微块大小。
STORING	表示索引表中冗余存储某些列，以提高系统查询性能。

示例

1. 执行以下命令，创建表test。

```
CREATE TABLE test (c1 int primary key, c2 VARCHAR(10));
```

2. 执行以下命令，创建表test的索引。

```
CREATE INDEX test_index ON test (c1, c2 DESC);
```

3. 执行以下命令，查看表test的索引。

```
SHOW INDEX FROM test;
```

5.5.13. CREATE OUTLINE

描述

该语句用来创建 OUTLINE。可以通过两种方式创建，一种是通过 SQL_TEXT(用户执行的带参数的原始语句)，另一种是通过 SQL_ID 创建。

> **说明**
>
> 创建 OUTLINE 需要进入对应的 DataBase 下执行。

格式

- **使用 SQL_TEXT 创建 OUTLINE**

```
CREATE [OR REPLACE] OUTLINE outline_name ON stmt [ TO target_stmt ]
```

- **使用 SQL_ID 创建 OUTLINE**

```
CREATE OUTLINE outline_name ON sql_id USING HINT  hint;
```

参数解释

参数	描述
outline_name	指定要创建的 OUTLINE 名称。
OR REPLACE	指定 OR REPLACE 后，如果要创建的 OUTLINE 已存在，则会替换原有的 OUTLINE。
stmt	一般为一个带有 hint 和原始参数的 DML 语句。
TO target_stmt	如果不指定 TO target_stmt， 则表示如果数据库接受的SQL参数化后与stmt去掉hint参数化文本相同，则将该SQL绑定stmt中hint生成执行计划；如果期望对含有hint的语句进行固定计划，则需要 TO target_stmt 来指明原始的SQL。 **注意** 在使用target_stmt时，严格要求stmt与target_stmt在去掉hint后完全匹配。
sql_id	如果sql_id对应的SQL语句已经有hint，则创建OUTLINE指定的hint会覆盖原始语句中所有hint。
hint	格式为 /*+ xxx */ 。

示例

- 使用 SQL_TEXT 创建 OUTLINE

```
CREATE OUTLINE otl_idx_c2
    ON SELECT/*+ index(t1 idx_c2)*/ * FROM t1 WHERE c2 = 1;
```

- 使用 SQL_ID 创建 OUTLINE

```
CREATE OUTLINE otl_idx_c2
ON "ED570339F2C856BA96008A29EDF04C74"
USING HINT /*+ index(t1 idx_c2)*/;
```

注意事项

使用 SQL_TEXT 方式创建的 OUTLINE 会覆盖 SQL_ID 方式创建的 OUTLINE，SQL_TEXT 方式创建的优先级高。

5.5.14. CREATE RESOURCE POOL

描述

该语句用来创建资源池，用于描述可以分配给租户的资源单元集合。

格式

```
CREATE RESOURCE POOL poolname
 UNIT [=] unitname,
 UNIT_NUM [=] unitnum,
 ZONE_LIST [=] ( ‘zone’ [, ‘zone’ …]);
```

参数解释

参数	描述
UNIT	指定要创建的资源单元名称
NIT_NUM	指定要创建的单个 ZONE 下的单元个数，取值要小于单个 ZONE 中的 OBServer 个数。
ZONE_LIST	指定要创建的资源池所属的集群

示例

- 创建资源池 pool1，属于集群 zone1。

```
OceanBase(admin@test)> CREATE RESOURCE POOL pool1 unit='unit1', unit_num=1, zone_list=('zone1');
Query OK, 0 rows affected (0.01 sec)
```

5.5.15. CREATE RESOURCE UNIT

描述

该语句用来创建资源单元配置，描述了分配硬件资源的规格，用于给租户分配硬件资源。

格式

```
CREATE RESOURCE UNIT unitname
MAX_CPU [=] cpunum,
MAX_MEMORY [=] memsize,
MAX_IOPS [=] iopsnum,
MAX_DISK_SIZE [=] disksize,
MAX_SESSION_NUM [=] sessionnum,
[MIN_CPU [=] cpunum,]
[MIN_MEMORY [=] memsize,]
[MIN_IOPS [=] iopsnum] ;
```

参数解释

参数	描述
MAX_CPU	指定要修改的 CPU 最多数量
MAX_MEMORY	指定要修改的最大内存容量，取值范围为[1073741824, +∞) 单位为字节，即最小值为1G
MAX_IOPS	指定要修改的 IOPS 最多数量，取值范围为[128,+∞)
MAX_DISK_SIZE	指定要修改的最大硬盘容量，取值范围为［536870912,+∞］单位为字节，即最小值为512M
MAX_SESSION_NUM	指定要修改的 session 最多数量，取值范围为[64,+∞)
MIN_CPU	指定要修改的 CPU 最少数量
MIN_MEMORY	指定要修改的最小内存容量
MIN_IOPS	指定要修改的 IOPS 最少数量

示例

- 创建资源单元 unit1 配置为1 个 CPU，1G 内存，128 个 IOPS，10G 硬盘容量，运行64个 session。

```
OceanBase(admin@test)> CREATE RESOURCE UNIT unit1 max_cpu 1, max_memory '1G', max_iops 128,max_
disk_size '10G', max_session_num 64, MIN_CPU=1, MIN_MEMORY= '1G', MIN_IOPS=128;
Query OK, 0 rows affected (0.02 sec)
```

- 创建资源单元 unit1 配置为1 个 CPU，1G 内存，128个 IOPS，10G 硬盘容量，运行64个 session。

```
OceanBase(admin@test)> CREATE RESOURCE UNIT unit1 max_cpu 1, max_memory 1073741824, max_iops 12
8, max_disk_size 10737418240, max_session_num 64, MIN_CPU=1, MIN_MEMORY=1073741824, MIN_IOPS=12
8;
Query OK, 0 rows affected (0.01 sec)
```

5.5.16. CREATE SYNONYM

描述

该语句用来创建一个同义词。

格式

```
CREATE [ OR REPLACE ] [ PUBLIC ]
SYNONYM [ DATABASE. ]synonym
FOR [ DATABASE. ]object;
```

参数说明

参数	描述
OR REPLACE	表示如果要创建的同义词名称已存在，则使用新的定义重新创建同义词。
PUBLIC	指定PUBLIC来创建公共同义词，所有用户都可以使用。用户必须对基础对象具有相应的权限才能使用该同义词。 在解析对象的引用时，仅当对象没有指定DataBase时，才会使用公共同义词。 如果不指定PUBLIC，则同义词是私有的，只能当前DataBase访问，并且同义词名称在当前DataBase中必须唯一。
[DataBase.]synonym	DataBase指定当前同义词属于哪个DataBase，如果指定了PUBLIC，则对同义词不需要指定DataBase；synonym表示同义词的名称。

参数	描述
[DataBase.]object	表示同义词对应对象的名称。

举例

- 创建一个同义词。

```
OceanBase(admin@test)>create table t1(c1 int);
Query OK, 0 rows affected (0.18 sec)

OceanBase(admin@test)>create synonym s1 for t1;
Query OK, 0 rows affected (0.05 sec)

OceanBase(admin@test)>insert into s1 values(1);
Query OK, 1 row affected (0.02 sec)

OceanBase(admin@test)>select * from s1;
+------+
| c1   |
+------+
|    1 |
+------+
1 row in set (0.01 sec)
```

- 创建一个PUBLIC同义词

```
OceanBase(admin@test)>create public synonym syn_pub for t1;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)>select * from syn_pub;
+------+
| c1   |
+------+
|    1 |
+------+
1 row in set (0.01 sec)
```

注意事项

创建同义词时，需要满足如下权限要求：

- 在当前的DataBase下创建私有的synonym，需要CREATE SYNONYM权限
- 在非当前DataBase下创建私有的synonym，需要CREATE ANY SYNONYM权限
- 创建PUBLIC的synonym，需要CREATE PUBLIC SYNONYM权限
- 对于需要创建synonym的对象不需要存在，也不需要具有访问权限

```
连接sys用户:
OceanBase(ADMIN@TEST)>CREATE USER syn_user IDENTIFIED BY syn_user;
Query OK, 0 rows affected (0.06 sec)

OceanBase(ADMIN@TEST)>grant CREATE on syn_user.* to syn_user;
Query OK, 0 rows affected (0.02 sec)

OceanBase(ADMIN@TEST)>grant SELECT on syn_user.* to syn_user;
Query OK, 0 rows affected (0.03 sec)



连接syn_user
//创建同义词失败
OceanBase(SYN_USER@(none))>create synonym syn_1 for t1;
ERROR-00600: internal error code, arguments: -5036, Access denied; you need (at least one of) the CREATE SY
NONYM privilege(s) for this operation

连接sys用户,给定CREATE SYNONYM权限
OceanBase(ADMIN@TEST)>grant CREATE SYNONYM on *.* to syn_user;
Query OK, 0 rows affected (0.03 sec)

连接syn_user
OceanBase(SYN_USER@(none))>create synonym syn_1 for t1;
Query OK, 0 rows affected (0.05 sec)
```

5.5.17. CREATE TABLE

描述

该语句用来在数据库中创建一张新表。

格式

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table_name
    (table_definition_list) [table_option_list] [partition_option] [AS] select;

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table_name
    LIKE table_name;
```

```
  LIKE table_name)

table_definition_list:
  table_definition [, table_definition ...]

table_definition:
    column_definition
  | [CONSTRAINT [constraint_name]] PRIMARY KEY index_desc
  | [CONSTRAINT [constraint_name]] UNIQUE {INDEX | KEY} [index_name] index_desc
  | {INDEX | KEY} [index_name] index_desc
  | FULLTEXT [INDEX | KEY] [index_name] fulltext_index_desc

column_definition_list:
  column_definition [, column_definition ...]

column_definition:
  column_name data_type
  [DEFAULT const_value] [AUTO_INCREMENT]
  [NULL | NOT NULL] [[PRIMARY] KEY] [UNIQUE [KEY]] comment

index_desc:
 (column_desc_list) [index_type] [index_option_list]

fulltext_index_desc:
 (column_desc_list) CTXCAT(column_desc_list) [index_option_list]

column_desc_list:
  column_desc [, column_desc ...]

column_desc:
   column_name [(length)] [ASC | DESC]

index_type:
  USING BTREE

index_option_list:
  index_option [ index_option ...]

index_option:
    [GLOBAL | LOCAL]
  | block_size
```

```
  | compression
  | STORING(column_name_list)
  | comment

table_option_list:
  table_option [ table_option ...]

table_option:
    [DEFAULT] {CHARSET | CHARACTER SET} [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | primary_zone
  | replica_num
  | table_tablegroup
  | block_size
  | compression
  | AUTO_INCREMENT [=] INT_VALUE
  | comment
  | DUPLICATE_SCOPE [=] "none|zone|region|cluster"
  | LOCALITY [=] "locality description"
  | ROW_FORMAT [=] REDUNDANT|COMPACT|DYNAMIC|COMPRESSED|DEFAULT
  | PCTFREE [=] num

partition_option:
    PARTITION BY HASH(expression)
    [subpartition_option] PARTITIONS partition_count
  | PARTITION BY KEY([column_name_list])
    [subpartition_option] PARTITIONS partition_count
  | PARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
    [subpartition_option] (range_partition_list)
  | PARTITION BY LIST {(expression) | COLUMNS (column_name_list)}
    [subpartition_option] PARTITIONS partition_count

subpartition_option:
    SUBPARTITION BY HASH(expression)
    SUBPARTITIONS subpartition_count
  | SUBPARTITION BY KEY(column_name_list)
    SUBPARTITIONS subpartition_count
  | SUBPARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
    (range_subpartition_list)
  | SUBPARTITION BY LIST(expression)
```

```
range_partition_list:
  range_partition [, range_partition ...]

range_partition:
  PARTITION partition_name
  VALUES LESS THAN {(expression_list) | MAXVALUE}

range_subpartition_list:
  range_subpartition [, range_subpartition ...]

range_subpartition:
  SUBPARTITION subpartition_name
  VALUES LESS THAN {(expression_list) | MAXVALUE}

expression_list:
  expression [, expression ...]

column_name_list:
  column_name [, column_name ...]

partition_name_list:
  partition_name [, partition_name ...]

partition_count | subpartition_count:
  INT_VALUE
```

参数说明

参数	描述
DUPLICATE_SCOPE	用来指定复制表属性，取值如下： • none：表示该表是一个普通表 • zone：表示该表是一个复制表，leader需要将事务复制到本zone的所有F副本及R副本 • region：表示该表是一个复制表，leader需要将事务复制到本region的所有F副本及R副本 • cluster：表示该表是一个复制表，leader需要将事务复掉到cluster的所有F副本及R副本 不指定DUPLICATE_SCOPE的情况下，默认值为none。

参数	描述
ROW_FORMAT	指定表是否开启encoding存储格式 • redundant ◦ 不开启encoding存储格式 • compact ◦ 不开启encoding存储格式 • dynamic ◦ encoding存储格式 • compressed ◦ encoding存储格式 • default ◦ 等价dynamic模式
BLOCK_SIZE	指定表的微块大小
COMPRESSION	指定表的压缩算法，取值如下： 1. none：不使用压缩算法 2. lz4_1.0: 使用lz4压缩算法 3. zstd_1.0: 使用zstd压缩算法 4. snappy_1.0: 使用snappy压缩算法
CHARSET | CHARACTER SET	指定表中列的默认字符集，可使用：utf8, utf8mb4, gbk, utf16, gb18030
COLLATE	指定表中列的默认比较规则，可使用： utf8_bin, utf8_general_ci, utf8_unicode_ci, gbk_bin, gbk_chinese_ci, utf8mb4_general_ci, utf8mb4__general_cs, utf8mb4_bin, utf8mb4_unicode_ci, utf16_general_ci, utf16_bin, utf16_unicode_ci, gb18030_chinese_ci, gb18030_bin
primary_zone	指定主Zone（副本Leader所在 Zone）。
replica_num	指定副本数。
table_tablegroup	指定表所属的talegroup。

参数	描述
AUTO_INCREMENT	指定表中自增列的初始值。
comment	注释。
LOCALITY	描述副本在Zone间的分布情况，如：F@z1,F@z2,F@z3,R@z4 表示z1, z2, z3为全功能副本，z4为只读副本。
PCTFREE	指定宏块保留空间百分比。

举例

- 创建数据库表。

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

- 创建表，并指定垂直分区。其中第一个分区只有列c3，第二个分区包含列c1、c2，没有列出的c4、c5列为第三个分区。

```
CREATE TABLE t1(c1 int,
        c2 int,
        c3 int,
        c4 int,
        c5 int)
        PARTITION BY CLOUMN ( c3, (c1, c2));
```

- 创建一个复制表。

```
CREATE TABLE item() locality = 'F,R{all_server}@hz1, F,R{all_server}@hz2,
F,R{all_server}@hz3' DUPLICATE_SCOPE="cluster"
```

- 创建带索引的表。

```
create table t1 (c1 int primary key, c2 int, c3 int, index i1 (c2));
```

- 创建hash分区，分区数为8的表。

```
create table t1 (c1 int primary key, c2 int) partition by hash(c1) partitions 8;
```

- 创建一级分区为range分区，二级分区为key分区的表。

```
create table t1 (c1 int, c2 int, c3 int)
  partition by range(c1) subpartition by key(c2, c3) subpartitions 5
  (partition p0 values less than(0), partition p1 values less than(100));
```

- 创建一列为gbk, 一列为utf8的表。

```
create table t1 (c1 varchar(10),
          c2 varchar(10) charset gbk collate gbk_bin)
 default charset utf8 collate utf8mb4_general_ci;
```

- 开启encoding并使用zstd压缩，宏块保留空间为5%。

```
create table t1 (c1 int, c2 int, c3 varchar(64))
 compression 'zstd_1.0'
 ROW_FORMAT dynamic
 pctfree 5;
```

5.5.18. CREATE TABLEGROUP

描述

该语句用来创建表组。

> **说明**
>
> 只有租户下的管理员权限才可以创建表组。

格式

```
CREATE TABLEGROUP [IF NOT EXISTS] tablegroupname [opt_tablegroup_option_list] [opt_tg_partition_option]

opt_tablegroup_option_list:
tablegroup_option [tablegroup_option]

tablegroup_option:
LOCALITY [=] locality_name
| PRIMARY_ZONE [=] primary_zone_name

opt_tg_partition_option:
PARTITION BY
 KEY COLUMN_NUM [tg_subpartition_option] PARTITIONS INTNUM
| HASH [tg_subpartition_option] PARTITIONS INTNUM
| RANGE [tg_subpartition_option] {PARTITION partition_name VALUES LESS THAN range_partition_expr, ...}
| RANGE COLUMNS COLUMN_NUM [tg_subpartition_option] {PARTITION partition_name VALUES LESS THAN range_partition_expr, ...}
| LIST [tg_subpartition_option] {PARTITION partition_name VALUES in list_partition_expr, ...}
| LIST COLUMNS COLUMN_NUM [tg_subpartition_option] {PARTITION partition_name VALUES in list_partition_expr, ...}

tg_subpartition_option:
SUBPARTITION BY
 RANGE SUBPARTITION TEMPLATE {SUBPARTITION partition_name VALUES LESS THAN range_partition_expr, ...}
| RANGE COLUMNS COLUMN_NUM SUBPARTITION TEMPLATE {SUBPARTITION partition_name VALUES LESS THAN range_partition_expr, ...}
| HASH [SUBPARTITIONS INTNUM]
| KEY COLUMN_NUM [SUBPARTITIONS INTNUM]
| LIST SUBPARTITION TEMPLATE {SUBPARTITION partition_name VALUES in list_partition_expr, ...}
| LIST COLUMNS COLUMN_NUM SUBPARTITION TEMPLATE {SUBPARTITION partition_name VALUES in list_partition_expr, ...}
```

参数解释

参数	描述

参数	描述
tablegroupname	表组名称，最长64个字符，字符只能有大小写英文字母，数字和下划线，而且必须以字母或下划线开头，并且不能OceanBase的关键字。 如果要创建的表组名称已存在，并且没有指定IF NOT EXISTS，则会出现错误。
opt_tablegroup_option_list	表组的分区方式、locality、primary zone必须和组内的表完全一致。 组内的表信息不能独立变更某一项，只可以通过操作表组进行批量操作。 相同的locality：副本类型、个数、位置要完全一致 相同的primary zone：leader位置及其优先级要完全一致 相同的分区方式： • 分区类型相同（例如，都是hash+range分区）。 • 如果是key分区，要求引用的列数相同，且分区个数相同。 • 如果是hash分区，要求分区个数相同。 • 如果是range columns分区，要求引用的列数相同，且分区数相同，且range分割点相同。 • 如果是range分区，且分区数相同，且range分割点相同。 • 对于二级分区，根据分区类型，要求和1）~4）一致
opt_tg_partition_option	表组的分区规则定义，与CREATE TABLE使用同样的分区方式。 表组没有具体的列定义，所以KEY、RANGE COLUMNS、LIST COLUMNS不需要写出具体的列，只需要指定列个数（COLUMN_NUM）。

示例

- 创建名为 myTableGroup1 的表组。

```
OceanBase(admin@test)> CREATE TABLEGROUP myTableGroup1;
Query OK, 0 rows affected (0.07 sec)

OceanBase(admin@test)> create table myt1 (c1 int, c2 int ) tablegroup = myTableGroup1;
Query OK, 0 rows affected (0.28 sec)

OceanBase(admin@test)> create table myt2 (c1 int, c2 int ) tablegroup = myTableGroup1;
Query OK, 0 rows affected (0.26 sec)
```

- 创建 hash 分区的表组 tgh，同时创建 hash 分区的表ttgh，且分区个数相同。

```
OceanBase(admin@test)> create tablegroup tgh locality='F,R{ALL_SERVER}@z1' partition by hash partitions
10;
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> create table ttgh(c1 int, c2 int) locality='F,R{ALL_SERVER}@z1' partition by hash(c
1) partitions 10;
Query OK, 0 rows affected (0.55 sec)

OceanBase(admin@test)> create table ttgh2(c1 int, c2 int) locality='F,R{ALL_SERVER}@z1' partition by hash(
c2) partitions 10;
Query OK, 0 rows affected (0.39 sec)
```

5.5.19. CREATE TENANT

描述

该语句用来创建租户。

格式

```
CREATE TENANT [IF NOT EXISTS] tenantname
   [tenant_characteristic_list] [opt_set_sys_var]

tenant_characteristic_list:
tenant_characteristic [, tenant_characteristic...]

tenant_characteristic:
COMMENT 'string'
|{CHARACTER SET | CHARSET} [=] charsetname
|COLLATE [=]  collationname
|REPLICA_NUM [=] num
|ZONE_LIST [=] (zone [, zone…])
|PRIMARY_ZONE [=] zone
|DEFAULT TABLEGROUP [=] {NULL | tablegroup}
|RESOURCE_POOL_LIST [=](poolname [, poolname…])
|LOGONLY_REPLICA_NUM [=] num
|LOCALITY [=] 'locality description'

opt_set_sys_var:
{ SET | SET VARIABLES | VARIABLES } system_var_name = expr [,system_var_name = expr] ...
```

参数解释

参数	描述
tenant_name	指定租户名。最长64个字节，只能有大小写英文字母，数字和下划线，而且必须以字母或下划线开头，并且不能是OceanBase的关键字。
IF NOT EXISTS	如果要创建的租户名已存在，并且没有指定IF NOT EXISTS，则会报错。
RESOURCE_POOL_LIST	资源池列表，为创建租户时的必填项，暂时只支持一个 RESOURCE POOL。
DEFAULT TABLEGROUP	设置租户默认表组信息，NULL 表示取消默认表组。如果不指定，默认为 NULL。
COMMENT	修改注释。

参数	描述
CHARACTER SET | CHARSET	修改租户的字符集。
COLLATE	指定校对规则。
REPLICA_NUM	指定副本数。
ZONE_LIST	指定要修改的Zone列表。
PRIMARY_ZONE	指定主Zone。
LOGONLY_REPLICA_NUM	指定日志副本数。
LOCALITY	描述副本在Zone间的分布情况，如：F@z1,F@z2,F@z3,R@z4 表示z1, z2, z3为全功能副本，z4为只读副本。
system_var_name	指定租户系统变量值。其中 ob_compatibility_mode 系统变量用于指定租户的兼容模式(MySQL/Oracle)，只能在创建时指定；如果不指定 ob_compatibility_mode ，默认兼容模式为MySQL。

示例

- 创建租户。

```
CREATE TENANT IF NOT EXISTS t1 charset='utf8mb4', replica_num=1, zone_list=('zone1'), primary_zone='zone1', resource_pool_list=('pool1');
```

- 创建oracle兼容模式租户

```
CREATE TENANT IF NOT EXISTS t1 zone_list=('zone1'), primary_zone='zone1', resource_pool_list=('pool1') SET ob_compatibility_mode='oracle';
```

注意事项

只有用root用户连接到根租户（root@ROOT）才能执行 CREATE TENANT 去创建租户。

5.5.20. CREATE USER

描述

该语句用于创建新的 OceanBase 用户。创建新用户后，可以使用该用户连接 OceanBase。

> **说明**
>
> 必须拥有全局的CREATE USER权限，才可以使用CREATE USER命令。

格式

```
create_user_stmt:
  CREATE USER [IF NOT EXISTS] user_name [IDENTIFIED BY 'password'];

alter_user_stmt:
   ALTER USER user_name ACCOUNT {LOCK | UNLOCK};
  | ALTER USER user_name IDENTIFIED BY 'password';
  | SET PASSWORD [FOR user_name] = PASSWORD('password');
  | RENAME USER rename_user_action_list;

drop_user_stmt:
  DROP USER user_name_list;

rename_user_action_list:
  rename_user_action [, rename_user_action ...]

rename_user_action:
  user_name TO user_name

user_name_list:
  user_name [, user_name ...]

password:
  STR_VALUE

CREATE USER [IF NOT EXISTS] user_specification_list;

user_specification_list:
user_specification [, user_specification ...]

user_specification:
user IDENTIFIED BY  'authstring'
| user IDENTIFIED BY PASSWORD  'hashstring'
```

参数解释

参数	描述
user_name	用户名，新建用户后，“mysql.user”表会新增一行该用户的表项。如果同名用户已经存在，则报错。
IDENTIFIED BY	使用自选的 IDENTIFIED BY 子句，可以为账户给定一个密码。
user_name [, user_name ...]	同时创建多个用户时，用“,”隔开。
user IDENTIFIED BY ‘authstring’	此处密码为明文，存入“mysql.user”表后，服务器端会变为密文存储下来。
user IDENTIFIED BY PASSWORD ‘hashstring’	此处密码为密文。

示例

1. 执行以下命令创建名为 sqluser01 和 sqluser02 的用户，密码均为 123456。

```
CREATE USER 'sqluser01' IDENTIFIED BY '123456', 'sqluser02' IDENTIFIED BY '123456';
```

2. 执行以下命令查看创建的用户。

```
SELECT user FROM mysql.user;
```

执行结果如下所示。

```
mysql> CREATE USER 'sqluser01' IDENTIFIED BY '123456', 'sqluser02' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.12 sec)
mysql> select user from mysql.user;
+-----------+
| user      |
+-----------+
| root      |
| admin     |
| sqluser01 |
| sqluser02 |
+-----------+
4 rows in set (0.00 sec)
```

5.5.21. CREATE VIEW

描述

该语句用于创建视图语句，如果指定了 OR REPLACE 子句，该语句能够替换已有的视图。

视图在数据库中实际上并不是以表的形式存在，每次使用时他们就会派生。视图是作为在 CREATE VIEW 语句中指定的 SELECT 语句的结果而派生出来的。

OceanBase 2.2.50 支持可更新视图。

格式

```
create_view_stmt:
 CREATE [OR REPLACE] VIEW view_name [(column_name_list)] AS select_stmt;

column_name_list:
  column_name [, column_name ...]
```

参数解释

参数	描述
OR REPLACE	表示如果要创建的同义词名称已存在，则使用新的定义重新创建同义词。
view_name	视图名。
select_stmt	是一种 SELECT 语句。它给出了视图的定义。该语句可以从基表或其他视图进行选择。
column_name_list	视图必须具有唯一的列名，不得有重复，就像基表那样。默认情况下，由 SELECT 语句检索的列名将用作视图列名。 要想为视图列定义明确的名称，可使用可选的 column_name_list 子句，列出由逗号隔开的 ID。column_name_list 中的名称数目必须等于SELECT语句检索的列数。 SELECT 语句检索的列可以是对表列的简单引用。也可以是使用函数、常量值、操作符等的表达式。

示例

选择表 t 中的 c1、c2 列来创建视图 v。

```
create or replace view v(vc1, vc2) as select c1, c2 from t;
```

5.5.22. DELETE

描述

该语句用来删除表中符合条件的行，包括单表删除及多表删除两种方式。

格式

```
Single-Table-Delete Syntax:
  DELETE [hint_options] FROM tbl_name
  [PARTITION (partition_name,...)]
  [WHERE where_condition]
  [ORDER BY order_expression_list]
  [LIMIT row_count]

Multiple-Table-Delete Syntax:
  DELETE [hint_options] tbl_name[.*] [, tbl_name[.*]] ...
  FROM table_references
  [WHERE where_condition]
Or:
  DELETE [hint_options] FROM tbl_name[.*] [, tbl_name[.*]] ...
  USING table_references
  [WHERE where_condition]

where_condition:
  expression

order_expression_list:
  order_expression [, order_expression ...]

order_expression:
  expression [ASC | DESC]

limit_row_count:
  INT_VALUE

table_references:
  {tbl_name | joined_table | table_subquery | select_with_parens} [, ...]
```

参数解释

参数	描述
hint_options	指定hint选项。
tbl_name	指定需要删除的表名。
partition_name	需要删除表的对应分区名。
where_condition	删除的表需要满足的过滤条件。
order_expression_list	删除的表的排序键列表。
row_count	删除的表的行数指定，指定的值只能为整数。
table_references	多表删除时指定的待选择的表序列。

示例

示例表及数据基于以下定义：

```
OceanBase(admin@test)>create table t1(c1 int primary key, c2 int);
Query OK, 0 rows affected (0.16 sec)
OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.06 sec)

OceanBase(admin@test)>create table t2(c1 int primary key, c2 int) partition by key(c1) partitions 4;
Query OK, 0 rows affected (0.19 sec)
OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  5 |    5 |
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
+----+------+
4 rows in set (0.02 sec)
```

- 单表删除：删除 “c1=2” 的行，其中 c1 列为表 t1 中的 Primary Key。

```
OceanBase(admin@test)>DELETE FROM t1 WHERE c1 = 2;
Query OK, 1 row affected (0.02 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  3 |    3 |
|  4 |    4 |
+----+------+
3 rows in set (0.01 sec)
```

- 单表删除：删除表 t2 的按照 c2 列排序之后的第一行数据。

```
OceanBase(admin@test)>DELETE FROM t1 ORDER BY c2 LIMIT 1;
Query OK, 1 row affected (0.01 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
3 rows in set (0.00 sec)
```

- 单表删除：执行删除表 t2 的 p2 分区的数据。

```
OceanBase(admin@test)>DELETE FROM t2 PARTITION(p2);
Query OK, 3 rows affected (0.02 sec)

OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  5 |    5 |
+----+------+
1 row in set (0.02 sec)
```

- 多表删除：删除 t1，t2 表中 "t1.c1 = t2.c1" 的数据。

```
OceanBase(admin@test)>DELETE t1, t2 FROM t1, t2 WHERE t1.c1 = t2.c1;
Query OK, 3 rows affected (0.02 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  4 |    4 |
+----+------+
1 row in set (0.01 sec)

OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  5 |    5 |
+----+------+
1 row in set (0.01 sec)
```

- 多表删除：删除 t1，t2 表中 "t1.c1 = t2.c1" 的数据。

```
OceanBase(admin@test)>DELETE FROM t1, t2 USING t1, t2 WHERE t1.c1 = t2.c1;
Query OK, 4 rows affected (0.02 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  4 |    4 |
+----+------+
1 row in set (0.01 sec)

OceanBase(admin@test)>select * from t2;
Empty set (0.01 sec)
```

- 多表删除：删除 t2 表中的 p2 分区中和 t1 表中 "t1.c1 = t2.c1" 的数据。

```
OceanBase(admin@test)>DELETE t2 FROM t1,t2 PARTITION(p2) WHERE t1.c1 = t2.c1;
Query OK, 3 rows affected (0.02 sec)

OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
| 5  |  5   |
+----+------+
1 row in set (0.01 sec)
```

- 对可更新视图v进行删除操作

```
OceanBase(admin@test)>create view v as select * from t1;
Query OK, 0 rows affected (0.07 sec)

OceanBase(admin@test)>delete from v where v.c1 = 1;
Query OK, 1 row affected (0.02 sec)

OceanBase(admin@test)>select * from v;
+----+------+
| c1 | c2   |
+----+------+
| 2  |  2   |
| 3  |  3   |
| 4  |  4   |
+----+------+
3 rows in set (0.01 sec)
```

注意事项

不管是多表删除还是单表删除都不支持直接对子查询进行删除操作，比如：

```
delete from (select * from t1);
```

5.5.23. DROP DATABASE

描述

DROP DATABASE 用于删除数据库。

格式

```
drop_database_stmt:
  DROP DATABASE [IF EXISTS] database_name;
```

参数解释

参数	描述
IF EXISTS	用于防止当数据库不存在时发生错误。
database_name	指定待删除的数据库名。

示例

- 删除数据库 test2。

```
OceanBase(admin@test)>drop database test2;
Query OK, 0 rows affected (0.03 sec)
```

- 删除不存在的数据库 notest。

```
OceanBase(admin@test)>drop database notest;
ERROR 1008 (HY000): Can't drop database 'notest'; database doesn't exist

OceanBase(admin@test)>drop database if exists notest;
Query OK, 0 rows affected, 1 warning (0.00 sec)

OceanBase(admin@test)>show warnings;
+-------+------+-----------------------------------------------------+
| Level | Code | Message                                             |
+-------+------+-----------------------------------------------------+
| Note  | 1008 | Can't drop database 'notest'; database doesn't exist |
+-------+------+-----------------------------------------------------+
1 row in set (0.00 sec)
```

5.5.24. DROP INDEX

描述

该语句用来删除索引。当索引过多时，维护开销增大，因此，需要删除不必要的索引。

删除索引的时候需要等待一段时间才能完全删除。

格式

```
DROP INDEX indexname
        ON tblname;
```

参数解释

参数	描述
indexname	指定索引名称。
tblname	指定表名。

示例

- 删除索引test_index。

```
DROP INDEX test_index ON test;
```

5.5.25. DROP OUTLINE

描述

该语句用于删除 OceanBase 数据库中的 OUTLINE。

格式

```
DROP OUTLINE outline_name;
```

参数解释

参数	描述
outline_name	指定要删除的 OUTLINE 名称。

示例

- 删除 OUTLINE ol_1

```
DROP OUTLINE ol_1;
```

5.5.26. DROP RESOURCE POOL

描述

该语句用来删除资源池。

格式

```
DROP RESOURCE POOL poolname;
```

参数解释

参数	描述
poolname	要删除的资源池名称

示例

- 删除资源池 pool1。

```
OceanBase(admin@test)> DROP RESOURCE POOL pool1;
Query OK, 0 rows affected (0.00 sec)
```

5.5.27. DROP RESOURCE UNIT

描述

该语句用来删除资源单元配置。

格式

```
DROP RESOURCE UNIT unitname
```

参数解释

参数	描述
unitname	要删除的资源单元配置的名称

示例

- 删除资源单元配置unit1。

```
OceanBase(admin@test)> DROP RESOURCE UNIT unit1;
Query OK, 0 rows affected (0.00 sec)
```

5.5.28. DROP TABLE

描述

该语句用于删除 OceanBase 数据库中的表。

格式

```
DROP [TEMPORARY] {TABLE | TABLES} [IF EXISTS]
  table_name [,table_name]...
  [RESTRICT | CASCADE]
```

参数解释

参数	描述
table_name	指定要删除的表名，同时删除多个表时，用“,”隔开。
IF EXISTS	如果指定IF EXISTS，即使要删除的表不存在，也不会报错；如果不指定，则会报错。
TEMPORARY	删除临时表。
RESTRICT | CASCADE	用于其它数据库迁移至OceanBase的场景。

示例

- 删除表 test

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

5.5.29. DROP TABLEGROUP

描述

该语句用来删除表组。

格式

```
DROP TABLEGROUP [IF EXISTS] tablegroupname
```

参数解释

参数	描述
tablegroupname	表组名称。如果要删除的表组名称不存在，并且没有指定 IF EXISTS，则会出现错误。

示例

删除名为 myTableGroup1 的表组。

```
OceanBase(admin@test)> DROP TABLEGROUP myTableGroup1;
```

5.5.30. DROP TENANT

描述

该语句用于删除一个 OceanBase 租户。

格式

```
drop_tenant_stmt:
  DROP TENANT [IF EXISTS] tenant_name;
```

参数解释

参数	描述
tenant_name	指定要删除的租户名。 只能删除处于“锁定”状态下的租户，对非锁定状态下的租户执行 DROP 时会报错。

示例

- 删除租户 TENANT1：

```
DROP TENANT TENANT1;
```

注意事项

只有用root用户连接到根租户（root@ROOT）才能执行 DROP TENANT 去删除租户。

5.5.31. DROP SYNONYM

描述

该语句用来删除一个同义词。

格式

```
DROP [PUBLIC] SYNONYM [ DATABASE. ]synonym;
```

参数说明

- PUBLIC

  指定PUBLIC来删除公共同义词；如果不指定PUBLIC，则删除私有同义词。

- [DATABASE.]synonym

  DATABASE指定当前同义词属于哪个DataBase，如果指定了PUBLIC，则对同义词不需要指定DataBase；synonym表示同义词的名称。

举例

- 删除一个同义词。

```
OceanBase(admin@test)>drop synonym test.s1;
Query OK, 0 rows affected (0.03 sec)
```

- 删除一个PUBLIC同义词

```
OceanBase(admin@test)>drop public synonym syn_pub;
Query OK, 0 rows affected (0.02 sec)
```

注意事项

> **注意**
>
> - 删除私有同义词时，需要满足要删除的synonym在对应DataBase下并要有DROP ANY SYNONYM权限。
> - 删除公共同义词时，需要有DROP PUBLIC SYNONYM权限 。
> - 删除public synonym时，必须指定PUBLIC关键字，且不能指定DataBase。

5.5.32. DROP USER

描述

该语句用于删除一个或多个OceanBase用户。

> **说明**
>
> - 必须拥有全局的 CREATE USER 权限，才可以使用 DROP USER 命令。
> - 不能对 mysql.user 表通过 DELETE 方式进行权限管理。
> - 成功删除用户后，该用户的所有权限也会被一同删除。

格式

```
DROP USER username [, username...];
```

参数解释

参数	描述
username	用户名。同时删除多个用户时，用“,”隔开。

示例

执行以下命令，删除名为 sqluser02 的用户。

```
oceanBase(admin@TEST)>drop user sqluser02;
Query OK, 0 rows affected (0.02 sec)
```

5.5.33. DROP VIEW

描述

该语句用于删除一个或多个视图。

> 说明
> 当前用户必须在每个视图上有 DROP 权限。

格式

```
drop_view_stmt:
  DROP VIEW [IF EXISTS] view_name_list [CASCADE | RESTRICT];

view_name_list:
  view_name [, view_name_list]
```

参数解释

参数	描述
IF EXISTS	使用IF EXISTS关键字可以防止因不存在的视图而出错。

参数	描述
view_name_list	如果view_name_list中包含一部分存在和一部分不存在的视图，执行可能报错但是存在的视图依然会被删除。
CASCADE、RESTRICT	CASCADE 和 RESTRICT 会被解析和忽略。

示例

分别执行以下命令，删除 v1 和 v2 两个视图。如果 v1或 v2 至少有一个视图不存在，则第一条命令会报错。

```
drop view v1, v2;

drop view if exists v1, v2;
```

5.5.34. EXPLAIN

描述

该语句用于解释 SQL 语句的执行计划，可以是 SELECT、DELETE、INSERT、REPLACE 或 UPDATE 语句。

格式

```
获取表或列的信息：
{EXPLAIN | DESCRIBE | DESC} tbl_name [col_name | wild]

获取SQL计划信息：
{EXPLAIN | DESCRIBE | DESC}
[BASIC | OUTLINE | EXTENDED | EXTENDED_NOADDR | PARTITIONS | FORMAT = {TRADITIONAL| JSON}]
{SELECT statement | DELETE statement | INSERT statement | REPLACE statement| UPDATE statement}
```

参数解释

参数	描述
tbl_name	指定表名。
col_name	指定表的列名。
BASIC	指定输出计划的基础信息，如算子ID、算子名称、所引用的表名。

参数	描述
OUTLINE	指定输出的计划信息包含outline信息。
EXTENDED	EXPLAIN产生附加信息，包括：每个算子的输入列和输出列，访问表的分区信息，当前使用的filter信息，如果当前算子使用了索引，显示所使用的索引列及抽取的query range。
EXTENDED_NOADDR	以简约的方式展示附加信息。
PARTITIONS	显示分区相关信息。
FORMAT = {TRADITIONAL| JSON}	指定EXPALIN的输出格式： • TRADITIONAL：表格输出格式 • JSON：KEY:VALUE输出格式，JSON显示为JSON字符串，包括EXTENDED和PARTITIONS信息。

示例

- 省略explain_type

```
OceanBase(admin@test)>explain select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: ====================================
|ID|OPERATOR  |NAME|EST. ROWS|COST  |
------------------------------------
|0 |HASH JOIN |  |9801000 |5933109|
|1 | TABLE SCAN|t2 |10000  |6219  |
|2 | TABLE SCAN|t1 |100000  |68478 |
====================================

Outputs & filters:
-------------------------------------
 0 - output([t1.c1], [t1.c2], [t2.c1], [t2.c2]), filter(nil),
    equal_conds([t1.c2 = t2.c2]), other_conds(nil)
 1 - output([t2.c2], [t2.c1]), filter(nil),
    access([t2.c2], [t2.c1]), partitions(p0)
 2 - output([t1.c2], [t1.c1]), filter(nil),
    access([t1.c2], [t1.c1]), partitions(p0)
```

- **EXTENDED**

```
OceanBase(admin@test)>explain extended_noaddr select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: ======================================
|ID|OPERATOR  |NAME|EST. ROWS|COST  |
--------------------------------------
|0 |HASH JOIN |  |9801000 |5933109|
|1 | TABLE SCAN|t2 |10000  |6219  |
|2 | TABLE SCAN|t1 |100000  |68478 |
======================================

Outputs & filters:
-------------------------------------
 0 - output([t1.c1], [t1.c2], [t2.c1], [t2.c2]), filter(nil),
    equal_conds([t1.c2 = t2.c2]), other_conds(nil)
 1 - output([t2.c2], [t2.c1]), filter(nil),
    access([t2.c2], [t2.c1]), partitions(p0),
    is_index_back=false,
    range_key([t2.c1]), range(4 ; MAX),
    range_cond([t2.c1 > 4])
 2 - output([t1.c2], [t1.c1]), filter(nil),
    access([t1.c2], [t1.c1]), partitions(p0),
    is_index_back=false,
    range_key([t1.__pk_increment], [t1.__pk_cluster_id], [t1.__pk_partition_id]), range(MIN,MIN,MIN ; MAX,MAX,MAX)always true
```

- **TRADITIONAL格式**

```
OceanBase(admin@test)>explain format=TRADITIONAL select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: ====================================
|ID|OPERATOR  |NAME|EST. ROWS|COST   |
------------------------------------
|0 |HASH JOIN |    |9801000  |5933109|
|1 | TABLE SCAN|t2  |10000    |6219   |
|2 | TABLE SCAN|t1  |100000   |68478  |
====================================

Outputs & filters:
-------------------------------------
 0 - output([t1.c1], [t1.c2], [t2.c1], [t2.c2]), filter(nil),
     equal_conds([t1.c2 = t2.c2]), other_conds(nil)
 1 - output([t2.c2], [t2.c1]), filter(nil),
     access([t2.c2], [t2.c1]), partitions(p0)
 2 - output([t1.c2], [t1.c1]), filter(nil),
     access([t1.c2], [t1.c1]), partitions(p0)
```

- JSON格式

```
OceanBase(admin@test)>explain format=JSON select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: {
 "ID":2,
 "OPERATOR":"JOIN",
 "NAME":"JOIN",
 "EST.ROWS":9800999,
 "COST":5933108,
 "output": [
  "t1.c1",
  "t1.c2",
  "t2.c1",
  "t2.c2"
 ],
 "TABLE SCAN": {
  "ID":0,
  "OPERATOR":"TABLE SCAN",
  "NAME":"TABLE SCAN",
  "EST.ROWS":10000,
  "COST":6218,
  "output": [
   "t2.c2",
   "t2.c1"
  ]
 },
 "TABLE SCAN": {
  "ID":1,
  "OPERATOR":"TABLE SCAN",
  "NAME":"TABLE SCAN",
  "EST.ROWS":100000,
  "COST":68477,
  "output": [
   "t1.c2",
   "t1.c1"
  ]
 }
}
```

EXPLAIN 的每个输出行提供一个表的相关信息，并且每个行包括下面的列：

列名	说明
ID	计划执行序列号。
OPERATOR	执行算子。
NAME	算子所引用的表。
EST.ROWS	估计执行到当前算子输出的行数。
COST	执行到当前算子的CPU时间。

5.5.35. FLASHBACK DATABASE

描述

用于从回收站中恢复被删除的DATABASE。

前置条件

回收站需要处于开启状态，可以通过

`show variables like 'recyclebin';`

来查看回收站是否开启。

```
OceanBase(admin@test)> show variables like 'recyclebin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| recyclebin    | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

果回收站处于关闭状态，可以通过

`set recyclebin = on;`

来开启。回收站中的表没有被实际删除，仍然会占用资源，如果需要彻底删除，可以执行

`purge recyclebin;`

格式

```
FLASHBACK DATABASE object_name TO BEFORE DROP [RENAME TO db_name];
```

参数解释

参数	解释
object_name	指定要恢复的object名称，不支持指定DATABASE名称。恢复DATABASE时，也会将DATABASE下面的表、索引等都恢复出来。
RENAME to	恢复时将DATABASE重命名。

示例

- 从回收站中恢复删除的DATABASE。

```
OceanBase(admin@test)> create database da;
Query OK, 1 row affected (0.03 sec)

OceanBase(admin@test)> drop database da;
Query OK, 0 rows affected (0.04 sec)

OceanBase(admin@test)> show recyclebin;
+-------------------------------------------------+--------------+----------+----------------------------+
| OBJECT_NAME                                     | ORIGINAL_NAME | TYPE     | CREATETIME                 |
+-------------------------------------------------+--------------+----------+----------------------------+
| __recycle_$_1_1099511628829_18446744073709551615 | da           | DATABASE | 2017-10-20 17:36:15.838771 |
+-------------------------------------------------+--------------+----------+----------------------------+
1 row in set (0.02 sec)

OceanBase(admin@test)> flashback database __recycle_$_1_1099511628829_18446744073709551615 to bef
ore drop;
Query OK, 0 rows affected (0.03 sec)
```

5.5.36. FLASHBACK TABLE

描述

用于从回收站中恢复被删除的TABLE。

前置条件

回收站需要处于开启状态，可以通过

`show variables like 'recyclebin';`

来查看回收站是否开启。

```
OceanBase(admin@test)> show variables like 'recyclebin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| recyclebin    | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

如果回收站处于关闭状态，可以通过

`set recyclebin = on;`

来开启。回收站中的表没有被实际删除，仍然会占用资源，如果需要彻底删除，可以执行

`purge recyclebin;`

格式

```
FLASHBACK TABLE object_name TO BEFORE DROP [RENAME to db_name.table_name];
```

参数解释

参数	解释
object_name	指定要恢复的object名称或表名，只有在表所在的database中才能执行。恢复TABLE时，也会将和TABLE相关的INDEX也恢复出来。
RENAME to	修改表名和表所属的库。

示例

- 从回收站中恢复被删除的表t。

```
OceanBase(admin@test)> create table t(id int primary key, k int);
Query OK, 0 rows affected (0.04 sec)

OceanBase(admin@test)> insert into t values(1,1);
Query OK, 1 row affected (0.00 sec)

OceanBase(admin@test)> select * from t;
+----+------+
| id | k    |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.00 sec)

OceanBase(admin@test)> drop table t;
Query OK, 0 rows affected (0.01 sec)

OceanBase(admin@test)> select * from t;
ERROR 1146 (42S02): Table 'test.t' does not exist
OceanBase(admin@test)> show recyclebin;
+--------------------------------+---------------+-------+----------------------------+
| OBJECT_NAME                    | ORIGINAL_NAME | TYPE  | CREATETIME                 |
+--------------------------------+---------------+-------+----------------------------+
| __recycle_$_1_1597028971700936 | t             | TABLE | 2020-08-10 11:09:31.701033 |
+--------------------------------+---------------+-------+----------------------------+
1 row in set (0.00 sec)

OceanBase(admin@test)> flashback table t to before drop;
Query OK, 0 rows affected (0.01 sec)

OceanBase(admin@test)> select * from t;
+----+------+
| id | k    |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.00 sec)
```

5.5.37. GRANT

描述

该语句用于系统管理员授予 User 某些权限。

> **说明**
>
> - 当前用户必须拥有被授予的权限（例如，user1 把表 t1 的 SELECT 权限授予 user2，则 user1 必须拥有表 t1 的 SELECT 的权限），并且拥有 GRANT OPTION 权限，才能授予成功。
> - 用户授权后，该用户只有重新连接OceanBase，权限才能生效。

格式

```
GRANT priv_type
        ON priv_level
        TO user_specification [, user_specification]...
  [WITH with_option ...]

privilege_type:
    ALTER
  | CREATE
  | CREATE USER
  | CREATE VIEW
  | DELETE
  | DROP
  | GRANT OPTION
  | INDEX
  | INSERT
  | PROCESS
  | SELECT
  | SHOW DATABASES
  | SHOW VIEW
  | SUPER
  | UPDATE
  | USAGE
  | CREATE SYNONYM

priv_level:
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.rountine_name

user_specification:
user [IDENTIFIED BY [PASSWORD] ‘password’ ]

with_option:
 GRANT OPTION
```

参数解释

参数	描述
priv_type	指定授予的权限类型。具体的权限类型及其说明请参见下方权限类型说明表。 同时把多个权限赋予用户时，权限类型用“,”隔开。
priv_level	指定授予权限的层级。权限可以分为以下几个层级： • 全局层级：适用于所有的数据库。使用 GRANT ALL ON *.*授予全局权限。 • 数据库层级：适用于一个给定数据库中的所有目标。使用 GRANT ALL ON db_name.* 授予数据库权限。 • 表层级：表权限适用于一个给定表中的所有列。使用 GRANT ALL ON db_name.tbl_name 授予表权限。 用“*”代替table_name，表示赋予全局权限，即对数据库中的所有表赋权。
user_specification	给特定用户授予权限。如果用户不存在，可以直接创建用户。 sql_mode='no_auto_create_user'，同时没有identified by 指定密码时，不可以直接创建用户。 同时给多个用户授权时，用户名用“,”隔开。
user IDENTIFIED BY 'password'	此处密码为明文。
user IDENTIFIED BY PASSWORD 'password'	此处密码为密文。
with_option	指定权限是否允许转授

可以授予的权限类型如下表所示。

权限类型说明表

权限	说明
ALL PRIVILEGES	除GRANT OPTION以外所有权限。
ALTER	ALTER TABLE的权限。
CREATE	CREATE TABLE的权限。

权限	说明
CREATE USER	CREATE USER，DROP USER，RENAME USER和REVOKE ALL PRIVILEGES的权限。
CREATE TABLEGROUP	全局CREATE TABLEGROUP的权限。
DELETE	DELETE的权限。
DROP	DROP的权限。
GRANT OPTION	GRANT OPTION的权限。
INSERT	INSERT的权限。
SELECT	SELECT的权限。
UPDATE	UPDATE的权限。
SUPER	SET GLOBAL修改全局系统参数的权限。
SHOW DATABASES	全局 SHOW DATABASES的权限。
INDEX	CREATE INDEX, DROP INDEX的权限
CREATE VIEW	创建、删除视图的权限。
SHOW VIEW	SHOW CREATE VIEW权限。
CREATE SYNONYM	创建同义词的权限。

> **说明**
>
> 目前没有 change effective tenant 的权限控制，故 sys 租户下的用户都可以进行授权。

示例

- 执行以下命令给用户 obsqluser01 赋予所有权限。

```
OceanBase(admin@TEST)>GRANT ALL PRIVILEGES ON *.* TO obsqluser01 with grant option;
Query OK, 0 rows affected (0.03 sec)
```

5.5.38. INSERT

描述

该语句用于添加一个或多个记录到表中。

格式

```
INSERT [IGNORE] [INTO]
    single_table_insert
    [ON DUPLICATE KEY UPDATE update_asgn_list]

single_table_insert:
  {dml_table_name values_clause
   | dml_table_name '(' ')' values_clause
   | dml_table_name '(' column_list ')' values_clause
   | dml_table_name SET update_asgn_list}

dml_table_name:
  tbl_name [PARTITION (partition_name,...)]

values_clause:
   {{VALUES | VALUE} ({expr | DEFAULT},...) [, ...]
    | select_stmt}

column_list
  column_name [, ...]

update_asgn_list:
   column_name = expr [, ...]
```

参数解释

INSERT...ON DUPLICATE KEY UPDATE... 语句执行，affect row的计算：

- 在没有设置client_capabilities中CLIENT_FOUND_ROWS的情况下：
  - 作为新行插入的话，affected_row= 1；
  - 存在冲突行的情况下，如果更新前后数据相同的话， affected_row = 0； 否则affectd_row = 2；
- 如果设置了CLIENT_FOUND_ROWS：

- 作为新行插入的话，affected_row=1；
- 如果更新前后数据相同的话，affectd_row=1；
- 如果更新前后数据不相同的话，affected_row=2；

- CLIENT_FOUND_ROWS的影响在于：不设置CLIENT_FOUND_ROWS的情况下，计算affected_row的值，只计算实际更新了行数，而设置了CLIENT_FOUND_ROWS的话，会把所有touched行数（满足冲突条件的行）都记上，而不管其是否发生了真正的数据修改。

参数	描述
IGNORE	在INSERT语句执行过程中发生的错误将会被忽略。
column_list	用于指定插入数据的列，同时插入多列时，用“,”隔开。
tbl_name	指定要插入的表名。
partition_name	插入表指定的分区名。
update_asgn_list	赋值语句，例如：“c1 = 2”
ON DUPLICATE KEY UPDATE	指定对重复主键或唯一键的处理。如果指定了ON DUPLICATE KEY UPDATE，当要插入的主键或唯一键有重复时，会用配置值替换待插入的值；如果不指定ON DUPLICATE KEY UPDATE，当要插入的主键或唯一键有重复时，插入报错。

示例

示例表及数据基于以下定义：

```
OceanBase(admin@test)>create table t1(c1 int primary key, c2 int) partition by key(c1) partitions 4;
Query OK, 0 rows affected (0.16 sec)

OceanBase(admin@test)>create table t2(c1 int primary key, c2 int);
Query OK, 0 rows affected (0.16 sec)
OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.06 sec)
```

- 向表 t1 中插入一行数数据。

```
OceanBase(admin@test)>insert into t1 values(1,1);
Query OK, 1 row affected (0.01 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.04 sec)
```

- 向表 t1 中插入多行数据。

```
OceanBase(admin@test)>insert t1 values(1,1),(2,default),(2+2,3*4);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 | NULL |
|  4 |   12 |
+----+------+
3 rows in set (0.02 sec)
```

- 向表 t1 指定的 p0 分区插入单行数据。

```
OceanBase(admin@test)>insert into t1 partition(p0) (c1) values(5);
Query OK, 1 row affected (0.02 sec)
OceanBase(admin@test)>select * from t1 partition(p0);
+----+------+
| c1 | c2   |
+----+------+
|  5 | NULL |
+----+------+
1 row in set (0.01 sec)
```

- 将表 t2 的查询结果作为数据插入表 t1。

```
OceanBase(admin@test)>insert into t1 select * from t2;
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.01 sec)
```

- 向表 t1 中插入重复主键值时利用 ON DUPLICATE KEY UPDATE 功能进行值更新

```
OceanBase(admin@test)>insert into t1 values(1,1),(1,2) ON DUPLICATE KEY UPDATE c1=100;
Query OK, 3 rows affected (0.01 sec)
Records: 2  Duplicates: 1  Warnings: 0

OceanBase(admin@test)>select * from t1;
+-----+------+
| c1  | c2   |
+-----+------+
| 100 |    1 |
+-----+------+
1 row in set (0.02 sec)
```

- 对可更新视图v的插入值

```
OceanBase(admin@test)>create view v as select * from t1;
Query OK, 0 rows affected (0.07 sec)
OceanBase(admin@test)>insert into v values(1,1);
Query OK, 1 row affected (0.01 sec)

OceanBase(admin@test)>select * from v;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.02 sec)
```

注意事项

INSERT语句不支持直接对子查询进行插入操作，比如：

```
insert into (select * from t1) values(1, 1);
```

5.5.39. KILL

描述

该语句用来终止一个会话。

> **说明**
>
> 如果您拥有 PROCESS 权限，则您可以查看所有会话。如果您拥有 SUPER 权限，您可以终止所有会话和语句。否则，您只能查看和终止您自己的会话和语句。

格式

```
KILL [CONNECTION | QUERY] 'sessionid'
```

参数解释

参数	描述
KILL CONNECTION	与不含修改符的KILL一样，可终止给定的*threadid*。
KILL QUERY	可终止连接当前正在执行的语句，但是会保持连接的原状。

示例

终止 sessionid 为 3221638213 的会话连接当前正在执行的语句后，再终止该会话。

```
OceanBase(admin@test)>show processlist;
+------------+-------+--------------------+------+---------+------+--------+------------------+
| Id         | User  | Host               | db   | Command | Time | State  | Info             |
+------------+-------+--------------------+------+---------+------+--------+------------------+
| 3221638212 | admin | 1.11.111.127:11161 | test | Query   |    0 | ACTIVE | show processlist |
| 3221638213 | admin | 1.11.111.127:11161 | test | Query   |    0 | ACTIVE | select "abcedfg" |
+------------+-------+--------------------+------+---------+------+--------+------------------+
1 row in set (0.03 sec)

OceanBase(admin@test)>kill query 3221638213;
Query OK, 0 rows affected (0.01 sec)

OceanBase(admin@test)>kill 3221638212;
Query OK, 0 rows affected (0.01 sec)
```

5.5.40. PURGE DATABASE

描述

该语句用于删除放在回收站的数据库。

格式

```
PURGE DATABASE object_name;
```

参数解释

参数	描述
object_name	对象进入回收站后的新名字

示例

- 删除放在回收站的数据库 __recycle_$_1_1597384386029184。

```
OceanBase(admin@test)> create database db1;
Query OK, 1 row affected (0.03 sec)

OceanBase(admin@test)> drop database db1;
Query OK, 0 rows affected (0.04 sec)

OceanBase(admin@test)> show recyclebin;
+--------------------------------+---------------+----------+----------------------------+
| OBJECT_NAME                    | ORIGINAL_NAME | TYPE     | CREATETIME                 |
+--------------------------------+---------------+----------+----------------------------+
| __recycle_$_1_1597384386029184 | db1           | DATABASE | 2020-08-14 13:53:06.029367 |
+--------------------------------+---------------+----------+----------------------------+
1 row in set (0.01 sec)

OceanBase(admin@test)> purge database __recycle_$_1_1597384386029184;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)> show recyclebin;
```

5.5.41. PURGE INDEX

描述

该语句用来删除放在回收站中的索引表。

格式

```
PURGE INDEX object_name;
```

参数解释

参数	描述
object_name	对象进入回收站后的新名字

示例

- 删除放在回收站中的索引表 __recycle_$_1_1597387726700872。

```
OceanBase(admin@test)> create table t1(c1 int);
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> create index idx on t1(c1);
Query OK, 0 rows affected (0.48 sec)

OceanBase(admin@test)> drop table t1;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)> show recyclebin;
+-------------------------------+---------------------------+-------+---------------------------+
| OBJECT_NAME                   | ORIGINAL_NAME             | TYPE  | CREATETIME                |
+-------------------------------+---------------------------+-------+---------------------------+
| __recycle_$_1_1597387726700872 | __idx_1101710651081557_idx | INDEX | 2020-08-14 14:48:46.699145 |
| __recycle_$_1_1597387726712976 | t1                        | TABLE | 2020-08-14 14:48:46.712643 |
+-------------------------------+---------------------------+-------+---------------------------+
5 rows in set (0.01 sec)

OceanBase(admin@test)> purge index __recycle_$_1_1597387726700872;
Query OK, 0 rows affected (0.04 sec)
```

5.5.42. PURGE RECYCLEBIN

描述

该语句用于 root 用户清空回收站。

> 注意
>
> PURGE RECYCLEBIN会清除回收站中的所有东西，只有 root 用户才能够执行这条语句，操作时请谨慎。

格式

```
PURGE RECYCLEBIN;
```

参数解释

无。

示例

- root 用户清空回收站。

```
OceanBase(admin@test)> purge recyclebin;
Query OK, 0 rows affected (0.03 sec)
```

5.5.43. PURGE TABLE

描述

该语句用于删除放在回收站的表。

格式

```
PURGE TABLE object_name;
```

参数解释

参数	描述
object_name	对象进入回收站后的新名字

示例

- 删除放在回收站中的表 __recycle_$_1_1099511628776_1099511677778。

```
OceanBase(admin@test)> create table test(c1 int);
Query OK, 0 rows affected (0.16 sec)

OceanBase(admin@test)> drop table test;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)> show recyclebin;
+-----------------------------------------+--------------+-------+---------------------------+
| OBJECT_NAME                             | ORIGINAL_NAME | TYPE | CREATETIME                |
+-----------------------------------------+--------------+-------+---------------------------+
| __recycle_$_1_1099511628776_1099511677778 | test       | TABLE | 2017-10-20 17:40:22.304025 |
+-----------------------------------------+--------------+-------+---------------------------+
1 row in set (0.02 sec)

OceanBase(admin@test)> purge table __recycle_$_1_1099511628776_1099511677778;
Query OK, 0 rows affected (0.04 sec)
```

5.5.44. RENAME TABLE

描述

该语句用来对一个或多个表进行重命名。

格式

```
RENAME TABLE tblname TO newtblname
  [, tb1name2 TO newtblname ···];
```

参数解释

参数	描述
tblname	原表名。
newtblname	新表名。
tblname TO newtblname [, tb1name2 TO newtblname ...];	多个表重命名时，使用“,”隔开。

注意事项

- 重命名操作自动进行，重命名正在进行时，其他线程不能读取任何表。
- 如果此语句用于对多个表进行重命名，则重命名操作从左到右进行。
- 当您执行 RENAME 时，您不能有被锁定的表，也不能有处于活性状态的事务。还必须拥有原表的 ALTER 和 DROP 权限，以及新表的 CREATE 和 INSERT 权限。
- RENAME TABLE 也可以用于视图，只要确保是在同一个数据库中。

示例

1. 创建表 t1 和 t2。

```
create table t1(c1 int);

create table t2(c1 int);
```

2. 将表 t1 重命名为 t11。

```
rename table t1 to t11;
```

3. 将表 t11 重命名为 t111，表 t2 重命名为 t22。

```
rename table t11 to t111, t2 to t22;
```

4. 将表 t111 重命名为 t1111 后，再将表 t1111重命名为 t1。

```
rename table t111 to t1111, t1111 to t1;
```

5.5.45. RENAME USER

描述

该语句用于修改 OceanBase 登录用户的用户名。

> 说明
>
> 必须拥有全局 CREATE USER 权限，才可以使用本命令。

格式

```
RENAME USER
'oldusername' TO 'newusername'
[,'oldusername' TO 'newusername'...];
```

参数解释

参数	描述
oldusername	旧用户名。
newusername	新用户名。用户名的占用字节小于或等于16。
'oldusername' TO 'newusername'	修改用户名。同时修改多个用户名时，用“,”隔开。 修改前后，新旧用户权限保持一致。

示例

1. 修改前，执行以下命令，查看用户。

```
SELECT user FROM mysql.user;
```

结果如下所示。

```
mysql> SELECT user FROM mysql.user;
+-----------+
| user      |
+-----------+
| root      |
| admin     |
| sqluser01 |
| sqluser02 |
+-----------+
4 rows in set (0.00 sec)
```

2. 执行以下命令，将 sqluser01 用户的用户名修改为 obsqluser01。

```
RENAME USER 'sqluser01' TO 'obsqluser01';
```

3. 修改后，执行以下命令，查看用户。

```
SELECT user FROM mysql.user;
```

结果如下所示。 sqluser01 用户的用户名被修改为 obsqluser01。

```
mysql> RENAME USER 'sqluser01' TO 'obsqluser01';
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT user FROM mysql.user;
+-------------+
| user        |
+-------------+
| root        |
| admin       |
| obsqluser01 |
| sqluser02   |
+-------------+
4 rows in set (0.00 sec)
```

5.5.46. REPLACE

描述

该语句用于替换一个或多个记录到表。如果没有主键或唯一键冲突则插入记录，如果存在冲突则先删除冲突记录再插入新的记录。

格式

```
replace_stmt:
  REPLACE [INTO] table_factor [PARTITION (partition_name_list)] [(column_name_list)]
  {VALUES | VALUE} column_value_lists;

partition_name_list:
    partition_name [, partition_name ...]

column_name_list:
  column_name [, column_name ...]

column_value_lists:
    (column_value_list) [, (column_value_list) ...]

column_value_list:
  column_value [, column_value ...]

column_value:
  {expression | DEFAULT}
```

参数解释

参数	描述
table_factor	替换的表名
column_name_list	指定替换数据的列
partition_name_list	替换表指定的分区名

示例

本示例基于如下表定义。

```
OceanBase(admin@test)>create table test (c1 int primary key, c2 varchar(40));
Query OK, 0 rows affected (0.23 sec)
```

1. 替换 test 表中的行一和行二的值分别为 'hello alibaba' 和 'hello ob'。

```
OceanBase(admin@test)>REPLACE INTO test VALUES (1, 'hello alibaba'),(2, 'hello ob');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

2. 查看 test 表中的行一和行二内容。

```
OceanBase(admin@test)>SELECT * FROM test;
+----+---------------+
| c1 | c2            |
+----+---------------+
|  1 | hello alibaba |
|  2 | hello ob      |
+----+---------------+
2 rows in set (0.00 sec)
```

3. 替换表 test 中的行三和行二的内容分别为 'hello alibaba' 和 'hello oceanbase'。

```
OceanBase(admin@test)>REPLACE INTO test VALUES (3, 'hello alibaba'),(2, 'hello oceanbase');
Query OK, 3 rows affected (0.00 sec)
Records: 2  Duplicates: 1  Warnings: 0
```

4. 查看 test 表中的行一、行二和行三的内容。

```
OceanBase(admin@test)>SELECT * FROM test;
+----+-----------------+
| c1 | c2              |
+----+-----------------+
|  1 | hello alibaba   |
|  2 | hello oceanbase |
|  3 | hello alibaba   |
+----+-----------------+
3 rows in set (0.00 sec)
```

5.5.47. REVOKE

描述

该语句用于系统管理员撤销 User 的某些权限。

使用说明如下：

- 用户必须拥有被撤销的权限（例如，user1 要撤销 user2 对表 t1 的 SELECT 权限，则 user1 必须拥有表 t1 的 SELECT的权限），并且拥有 GRANT OPTION权限。

- 撤销 ALL PRIVILEGES 和 GRANT OPTION 权限时，当前用户必须拥有全局 GRANT OPTION 权限，或者对权限表的UPDATE 及 DELETE 权限。
- 撤销操作不会级联。例如，用户 user1 给 user2 授予了某些权限，撤回 user1 的权限不会同时也撤回 user2 的相应权限。

格式

```
REVOKE priv_type
            ON database.tblname
            FROM 'username';

privilege_type:
    ALTER
  | CREATE
  | CREATE USER
  | CREATE VIEW
  | DELETE
  | DROP
  | GRANT OPTION
  | INDEX
  | INSERT
  | PROCESS
  | SELECT
  | SHOW DATABASES
  | SHOW VIEW
  | SUPER
  | UPDATE
  | USAGE
```

参数解释

参数	描述
priv_type	指定撤销的权限类型。具体的权限类型及其说明请参见下方权限类型说明表。 同时对某个用户撤销多个权限时，权限类型用“,”隔开。
database.tblname	指定数据库中的表。 用“*”代替 database 或 *table_name*，表示撤销全局权限，即撤销对数据库中所有表的操作权限。

参数	描述
username	指定撤销权限的用户。同时撤销多个用户的授权时，用户名用“,”隔开。

可以撤销的权限类型如下表所示。

权限类型说明表

权限	说明
ALL PRIVILEGES	除GRANT OPTION以外所有权限。
ALTER	ALTER TABLE的权限。
CREATE	CREATE TABLE的权限。
CREATE USER	CREATE USER，DROP USER，RENAME USER和REVOKE ALL PRIVILEGES的权限。
CREATE TABLEGROUP	全局CREATE TABLEGROUP的权限。
DELETE	DELETE的权限。
DROP	DROP的权限。
GRANT OPTION	GRANT OPTION的权限。
INSERT	INSERT的权限。
SELECT	SELECT的权限。
UPDATE	UPDATE的权限。
SUPER	SET GLOBAL修改全局系统参数的权限。
SHOW DATABASES	全局 SHOW DATABASES的权限。

权限	说明
INDEX	CREATE INDEX, DROP INDEX的权限。
CREATE VIEW	创建、删除视图的权限。
SHOW VIEW	SHOW CREATE VIEW权限。
CREATE SYNONYM	创建同义词的权限。

> 说明
> 目前没有change effective tenant 的权限控制，故 sys 租户下的用户都可以撤销权限。

示例

执行以下命令撤销用户 obsqluser01 的所有权限。

```
OceanBase(admin@TEST)>REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'obsqluser01';
Query OK, 0 rows affected (0.03 sec)
```

5.5.48. SAVEPOINT

描述

SAVEPOINT语句用来实现事务的部分回滚。

格式

1. 创建savepoint：

```
SAVEPOINT spname
```

2. 回滚到savepoint：

```
ROLLBACK [WORK] to [SAVEPOINT] spname
```

3. 删除savepoint：

```
RELEASE SAVEPOINT spname
```

参数说明

- spname：指定savepoint的名称。savepoint是事务范围内唯一的，一个savepoint会覆盖前一个同名的

savepoint。创建savepoint后可以将事务回滚到指定savepoint，也可以使用 `ROLLBACK` 语句回滚整个事务。

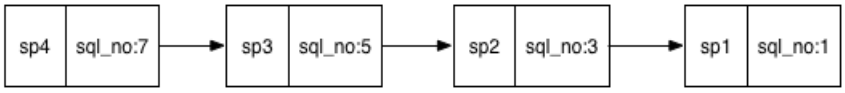
举例

假设一个事务执行了如下语句：

sql_no	语句	分区
1	update...	p1, p4
	savepoint sp1	
2	update...	p2, p4
3	update...	p3, p5
	savepoint sp2	
4	update...	p1, p3, p6
5	update...	p1, p5
	savepoint sp3	
6	select...	
7	update...	p5, p6
	savepoint sp4	

记录savepoint

用户在提交事务之前可以创建savepoint，需要根据savepoint创建的顺序，将事务的savepoint串成链表。以上事务包含了7条sql和4个savepoint，记录savepoint的链表如下图所示，其中每个节点记录了<spname, sql_no>的映射关系：



事务参与者列表

事务为了支持回滚某条sql之后的所有修改，需要将每条语句涉及的参与者以及对应的sql_no记录下来，以上事务执行了7条sql，涉及p1~p6共6个partition：

p1	p2	p3	p4	p5	p6
1,4,5	2	2,3,4	1,2	3,5,7	4,7

savepoint回滚过程

1. 根据savepoint链表查询spname对应的sql_no

假设用户执行 `ROLLBACK to SAVEPOINT sp2` ，根据savepoint链表查询到sp2对应的sql_no为3。

2. 根据事务参与者列表查询sql_no对应的partition

根据事务参与者列表查询到sql_no大于3的语句操作的分区涉及p1、p3、p5、p6。

3. 回滚分区数据

根据第2步查询到的分区，调度程序向这些分区发起回滚请求，回滚当前事务在这些分区上sp2之后的所有修改。其中p1、p3、p5上关于本事务的部分修改被回滚掉，p6上关于本事务的所有修改都被回滚掉。

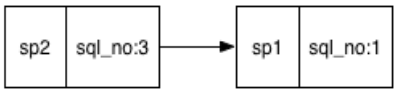
4. 更新事务参与者列表信息

修改事务参与者列表，将sql_no大于3的操作信息从事务参与者列表中删除，由于p6上的所有修改都被回滚掉，因此p6可以从参与者列表中删除。

p1	p2	p3	p4	p5	p6
1	2	2, 3	1, 2	3	

5. 删除无效的savepoint

用户执行 `ROLLBACK to SAVEPOINT sp2` 成功后，系统会删除sp3和sp4的savepoint，不允许再回滚到sp3和sp4。



5.5.49. SCHEMA

`SCHEMA` 同 `DATABASE` ，请参考 CREATE DATABASE、ALTER DATABASE、DROP DATABASE 的详细描述。

5.5.50. SELECT

SELECT 的语法相对比较复杂。本节首先会介绍普通的 SELECT 语法结构，然后介绍集合类 SELECT 的语法结构。

SIMPLE SELECT

描述

该语句用于查询表中的内容。

格式

```
simple_select:
```

```
SELECT [/*+ hint statement */] [ALL | DISTINCT | UNIQUE | SQL_CALC_FOUND_ROWS]
 select_expr_list FROM from_list [WHERE condition]
 [GROUP BY group_expression_list [WITH ROLLUP] [HAVING condition]]
 [ORDER BY order_expression_list]
 [limit_clause]
 [FOR UPDATE]

select_expr:
 table_name.*
 | table_alias_name.*
 | expr [[AS] column_alias_name]

from_list:
   table_reference [, table_reference ...]

table_reference:
    simple_table
  | joined_table

simple_table:
   table_factor [partition_option] [[AS] table_alias_name]
  | (select_stmt) [AS] table_alias_name
  | (table_reference_list)

joined_table:
    table_reference [INNER] JOIN simple_table [join_condition]
  | table_reference outer_join_type JOIN simple_table join_condition

partition_option:
   PARTITION (partition_name_list)

partition_name_list:
   partition_name [, partition_name ...]

outer_join_type:
   {LEFT | RIGHT | FULL} [OUTER]

join_condition:
   ON expression

condition:
```

```
  expression

group_expression_list:
  group_expression [, group_expression ...]

group_expression:
  expression [ASC | DESC]

order_expression_list:
  order_expression [, order_expression ...]

order_expression:
  expression [ASC | DESC]

limit_clause:
 LIMIT {[offset,] row_count |row_count OFFSET offset}
```

参数解释

参数	描述
ALL | DISTINCT | UNIQUE | SQL_CALC_FOUND_ROWS	在数据库表中，可能会包含重复值。 • 指定“DISTINCT”，则在查询结果中相同的行只显示一行； • 指定“ALL”，则列出所有的行； • 指定“SQL_CALC_FOUND_ROWS”，则不输出数据，只返回数据行数； • 不指定时，默认为“ALL”。
select_expr	列出要查询的表达式或列名，用“,”隔开。也可以用“*”表示所有列。
AS othername	为输出字段重新命名。
FROM table_references	指名了从哪个表或哪些表中读取数据（支持多表查询）。
WHERE where_conditions	可选项，WHERE 字句用来设置一个筛选条件，查询结果中仅包含满足条件的数据。where_conditions 为表达式。

参数	描述
GROUP BY group_by_list	用于进行分类汇总。
HAVING search_confitions	HAVING 字句与 WHERE 字句类似，但是 HAVING 字句可以使用累计函数（如 SUM，AVG 等）。
ORDER BY order_list order_list： colname [ASC | DESC] [,colname [ASC | DESC]...]	用来按升序（ASC）或者降序（DESC）显示查询结果。不指定 ASC 或者 DESC 时，默认为 ASC。
[LIMIT {[offset,] row_count |row_count OFFSET offset}]	强制 SELECT 语句返回指定的记录数。 LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。 • 如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)。 • 如果只给定一个参数，它表示返回记录行的最大数目，偏移量为0。
FOR UPDATE	对查询结果所有行上排他锁，以阻止其他事务的并发修改，或阻止在某些事务隔离级别时的并发读取。
PARTITION(partition_list)	指定查询表的分区信息。例如：partition(p0,p1...)

示例

以如下表 a 为例。

表a

id	name	num
1	a	100
2	b	200
3	a	50

- 从表 a 中读取 name 数据。

```
SELECT name FROM a;
```

```
+------+
| name |
+------+
| a    |
| b    |
| a    |
+------+
3 rows in set (0.01 sec)
```

- 在查询结果中对 name 进行去重处理。

```
SELECT DISTINCT name FROM a;
```

```
+------+
| name |
+------+
| a    |
| b    |
+------+
2 rows in set (0.01 sec)
```

- 从表 a 中查询 id，name，num，然后把 num 列除以2输出，输出的列名为 avg。

```
SELECT id, name, num/2 AS avg FROM a;
```

```
+------+------+------+
| id   | name | avg  |
+------+------+------+
|    1 | a    |   50 |
|    2 | b    |  100 |
|    3 | a    |   25 |
+------+------+------+
3 rows in set (0.00 sec)
```

- 从表 a 中根据筛选条件“ name = 'a' ”，输出对应的 id 、name 和 num 。

```
SELECT id, name, num FROM a WHERE name = 'a';
```

```
+------+------+------+
| id   | name | num  |
+------+------+------+
|    1 | a    |  100 |
|    3 | a    |   50 |
+------+------+------+
2 rows in set (0.01 sec)
```

- 从表 a 中查询 id，name，按照 name 分组对 num 求和，并输出。

```
SELECT id, name, SUM(num) FROM a GROUP BY name;
```

```
+------+------+----------+
| id   | name | SUM(num) |
+------+------+----------+
|    1 | a    |      150 |
|    2 | b    |      200 |
+------+------+----------+
2 rows in set (0.00 sec)
```

- 从表 a 中查询 id，name，按照 name 分组对 num 求和，查询 num 总和小于160的行，并输出。

```
SELECT id, name, SUM(num) as sum FROM a GROUP BY name HAVING SUM(num) < 160;
```

```
+------+------+------+
| id   | name | sum  |
+------+------+------+
|    1 | a    |  150 |
+------+------+------+
1 row in set (0.01 sec)
```

- 从表 a 中查询 id，name，num，根据 num 按升序（ASC）输出查询结果。

```
SELECT * FROM a ORDER BY num ASC;
```

```
+------+------+------+
| id   | name | num  |
+------+------+------+
|    3 | a    |   50 |
|    1 | a    |  100 |
|    2 | b    |  200 |
+------+------+------+
3 rows in set (0.00 sec)
```

- 从表 a 中查询 id，name，num，根据 num 按降序（DESC）输出查询结果。

```
SELECT * FROM a ORDER BY num DESC;
```

```
+------+------+------+
| id   | name | num  |
+------+------+------+
|    2 | b    |  200 |
|    1 | a    |  100 |
|    3 | a    |   50 |
+------+------+------+
3 rows in set (0.01 sec)
```

- 从表 a 中查询 id，name，num，使用 LIMIT 强制从第2行开始，返回表 a 的两行结果。

```
SELECT * FROM a LIMIT 1,2;
```

```
+------+------+------+
| id   | name | num  |
+------+------+------+
|    2 | b    |  200 |
|    3 | a    |   50 |
+------+------+------+
2 rows in set (0.00 sec)
```

集合类 SELECT

描述

该语句用于对多个 SELECT 查询的结果进行 UNION，MINUS，INTERSECT。

格式

```
select_clause_set:
  simple_select [ UNION | UNION ALL | EXCEPT | INTERSECT] select_clause_set_left
 [ORDER BY sort_list_columns] [limit_clause]
select_clause_set_right:
  simple_select |
 select_caluse_set
```

参数解释

参数	描述
UNION ALL	合并两个查询的结果
UNION	合并两个查询的结果，并去重
EXCEPT	从左查询结果集中去重出现在右查询中的结果，并去重
INTERSECT	保留左查询结果集中出现在右查询中的结果，并去重

示例

以如下两表的数据为例：

```
create table t1 (c1 int, c2 int);
create table t2 (c1 int, c2 int);
insert into t1 values (1, -1), (2, -2);
insert into t2 values (1, 1), (2, -2), (3, 3);
```

- 计算 T1, T2 的所有的记录

```
SELECT C1, C2 FROM T1 UNION ALL SELECT C1, C2 FROM T2;
+------+------+
| C1   | C2   |
+------+------+
|    1 |   -1 |
|    2 |   -2 |
|    1 |    1 |
|    2 |   -2 |
|    3 |    3 |
+------+------+
```

- 计算 T1, T2 的去重后的所有记录

```
SELECT C1, C2 FROM T1 UNION SELECT C1, C2 FROM T2;
+------+------+
|C1  |C2  |
+------+------+
|  1|  -1|
|  2|  -2|
|  1|  1|
|  3|  3|
+------+------+
```

- 计算 T1 和 T2 的交集

```
SELECT C1, C2 FROM T1 INTERSECT SELECT C1, C2 FROM T2;
+------+------+
|C1  |C2  |
+------+------+
|  2|  -2|
+------+------+
```

- 计算 T1 和 T2 的差集

```
SELECT C1, C2 FROM T1 EXCEPT SELECT C1, C2 FROM T2;
+------+------+
|C1  |C2  |
+------+------+
|  1|  -1|
+------+------+
```

- 取 T1 和 T2 并集中 C2 排序最大的前两行

```
SELECT C1, C2 FROM T1 UNION SELECT C1, C2 FROM T2 ORDER BY C2 DESC LIMIT 2;
+------+------+
|C1  |C2  |
+------+------+
|  3|  3|
|  1|  1|
+------+------+
```

5.5.51. SESSION

描述

该语句用来强制结束 session。

格式

```
session_stmt:
  KILL [CONNECTION] session_id;

session_id:
  INT_VALUE
```

参数解释

参数	描述
session_id	要结束 session 的编号

示例

- 强制结束编号为 3221502221和 3221750376 的 session。

```
OceanBase(admin@test)>show processlist;
+------------+-------+---------------------+------+---------+------+--------+---------------------------------+
| Id         | User  | Host                | db   | Command | Time | State  | Info                            |
+------------+-------+---------------------+------+---------+------+--------+---------------------------------+
| 3221502221 | admin | 100.xx.xx.xx:44775  | test | Sleep   |  10 | SLEEP  | NULL                            |
| 3221502083 | admin | 100.xx.xx.xx:44720  | test | Query   |   0 | ACTIVE | show processlist                |
| 3221502317 | NULL  | 100.xx.xx.xx:41414  | test | Query   |   0 | ACTIVE | REMOTE/DISTRIBUTE PLAN EXECUTING |
| 3221750377 | NULL  | 100.xx.xx.xx:41414  | test | Query   |   0 | ACTIVE | REMOTE/DISTRIBUTE PLAN EXECUTING |
| 3221750376 | admin | 100.xx.xx.xx:43783  | test | Sleep   |   4 | SLEEP  | NULL                            |
+------------+-------+---------------------+------+---------+------+--------+---------------------------------+
5 rows in set (0.04 sec)

OceanBase(admin@test)>kill 3221502221;
Query OK, 0 rows affected (0.00 sec)

OceanBase(admin@test)>kill connection 3221750376;
Query OK, 0 rows affected (0.05 sec)
```

5.5.52. SET PASSWORD

描述

该用于修改 OceanBase 当前登录用户或其他用户的密码。

格式

```
SET PASSWORD [FOR user] = password_option;

password_option: {
PASSWORD（‘authstring’）
|’ hashstring’
}
```

参数解释

参数	描述
FOR user	如果没有 For user 子句，表示修改当前用户的密码。任何成功登录的用户都可以修改当前用户的密码。 如果有 For user 子句，表示修改指定用户的密码。必须拥有全局（MySQL库）的 CREATE USER 权限，才可以修改指定用户的密码。

示例

执行以下命令将 sqluser01 用户的密码修改为 abc123。

```
oceanBase(admin@TEST)>set password for sqluser01 = password('abc123');
Query OK, 0 rows affected (0.02 sec)
```

5.5.53. SHOW GRANTS

描述

该语句用于系统管理员查看 User 的操作权限。

格式

```
SHOW GRANTS [FOR username];
```

参数解释

参数	描述
FOR username	如果不指定用户名，则缺省显示当前用户的权限。当前用户可以查看自己的权限。 如果需要查看其他指定用户的权限，必须拥有对 mysql.user 的 SELECT 权限。

示例

查看看用户 obsqluser01 的权限。

```
oceanBase(admin@TEST)>show grants for obsqluser01;
+-------------------------------------+
| Grants for obsqluser01@%            |
+-------------------------------------+
| GRANT USAGE ON *.* TO 'obsqluser01' |
+-------------------------------------+
1 row in set (0.02 sec)
```

5.5.54. SHOW RECYCLEBIN

描述

该语句用于查看回收站中的内容。

格式

```
SHOW RECYCLEBIN;
```

参数解释

无。

示例

- 查看回收站中的内容。

```
OceanBase(admin@test)> create table t1(c1 int);
Query OK, 0 rows affected (0.24 sec)

OceanBase(admin@test)> drop table t1;
Query OK, 0 rows affected (0.07 sec)

OceanBase(admin@test)> show recyclebin;
+-----------------------------------------+--------------+-------+---------------------------+
| OBJECT_NAME                             | ORIGINAL_NAME | TYPE | CREATETIME                |
+-----------------------------------------+--------------+-------+---------------------------+
| __recycle_$_1_1099511628776_1099511677777 | t1          | TABLE | 2017-10-20 17:27:40.881506 |
+-----------------------------------------+--------------+-------+---------------------------+
1 row in set (0.02 sec)
```

5.5.55. TRANSACTION

描述

该语句用于开启事务。

数据库事务（Database Transaction）是指作为单个逻辑工作单元执行的一系列操作。事务处理可以用来维护数据库的完整性，保证成批的SQL操作全部执行或全部不执行。

显示事务是用户自定义或用户指定的事务。通过 BEGIN TRANSACTION，或 BEGIN 和 BEGIN WORK（被作为 START TRANSACTION的别名受到支持）语句显示开始，以 COMMIT 或 ROLLBACK 语句显示结束。

格式

```
transaction_stmt:
    START TRANSACTION [READ ONLY | READ WRITE];
  | BEGIN [WORK];
  | COMMIT [WORK];
  | ROLLBACK [WORK];
  | SET TRANSACTION {READ ONLY | READ WRITE};
```

参数解释

参数	描述

参数	描述
START TRANSACTION [READ ONLY | READ WRITE]	开启事务语句。一旦开启事务，则随后的 SQL 数据操作语句（即INSERT、UPDATE、DELETE等）直到显式提交时才会生效。 READ ONLY 子句表示事务是只读方式开启，事务内不允许执行修改操作。 READ WRITE 子句表示事务是读写方式开启，默认就是此种模式。
BEGIN	BEGIN 和 BEGIN WORK 被作为 START TRANSACTION 的别名受到支持
COMMIT	提交当前事务。
ROLLBACK	回滚当前事务。
SET TRANSACTION {READ ONLY | READ WRITE}	将当前事务设置成 READ ONLY 或者 READ WRITE 模式。

示例

假设现有表 a 如下所示。

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43
2	b	200	2013-06-21 13:07:21
3	a	50	2013-06-21 13:08:15

1. 依次执行以下命令开始执行事务，将 id 为 3 的的 name 改为 c，并插入一行当前卖出 a 的记录。

```
OceanBase(admin@test)> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

OceanBase(admin@test)> UPDATE a SET name = 'c' WHERE id = 3;
Query OK, 1 rows affected (0.00 sec)

OceanBase(admin@test)> INSERT INTO a VALUES (4, 'a', 30, '2013-06-21 16:09:13');
Query OK, 1 rows affected (0.00 sec)

OceanBase(admin@test)> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

2. 事务提交后，执行命令查看表 a 信息。

```
SELECT * FROM a;
```

结果如下：

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43
2	b	200	2013-06-21 13:07:21
3	c	50	2013-06-21 13:08:15
4	a	30	2013-06-21 16:09:13

注意

在事务还没有 COMMIT 之前，您可以查看下本事务中的操作是否已经生效，比如可以在 COMMIT 前，加一句“SELECT * FROM a;"。本事务 session 的访问能读到最新的结果，本事务 session 之外的访问结果肯定是没有生效，在事务还没有 COMMIT 前，你之前做的操作除当前事务连接之外都是不可见的。如果您想回滚该事务，直接用“ROOLBACK”代替“COMMIT”。

5.5.56. TRUNCATE TABLE

描述

该语句用于完全清空指定表，但是保留表结构，包括表中定义的 Partition 信息。从逻辑上说，该语句与用于删除所有行的DELETE FROM语句相同。执行TRUNCATE语句，必须具有表的删除和创建权限。它属于DDL语句。

TRUNCATE TABLE 语句与 DELETE FROM 语句有以下不同：

- 删减操作会取消并重新创建表，这比一行一行的删除行要快很多。
- TRUNCATE TABLE 语句执行结果显示影响行数始终显示为 0 行。
- 使用 TRUNCATE TABLE 语句，表管理程序不记得最后被使用的 AUTO_INCREMENT 值，但是会从头开始计数。
- TRUNCATE 语句不能在进行事务处理和表锁定的过程中进行，如果使用，将会报错。
- 只要表定义文件是合法的，则可以使用 TRUNCATE TABLE 把表重新创建为一个空表，即使数据或索引文件已经被破坏。

格式

```
TRUNCATE [TABLE] table_name;
```

参数解释

参数	描述
table_name	指定表名。

示例

完全清空表 tb1。

```
TRUNCATE [TABLE] tb1;
```

5.5.57. UPDATE

描述

该语句用于修改表中的字段值。

格式

```
UPDATE [IGNORE] table_references
  SET update_asgn_list
  [WHERE where_condition]
  [ORDER BY order_list]
  [LIMIT row_count];

table_references:
  tbl_name [PARTITION (partition_name,...)] [, ...]

update_asgn_list:
  column_name = expr [, ...]

order_list:
  column_name [ASC|DESC] [, column_name [ASC|DESC]···]
```

参数解释

参数	描述
IGNORE	在 INSERT 语句执行过程中发生的错误将会被忽略。
table_references	指定修改表名，多表修改时，表名直接‘，’作为间隔。
where_condition	指定过滤条件。
row_count	限制的行数。
tbl_name	插入表名。
partition_name	插入表指定的分区名。
column_name	列名。
column_name ASC	按列名升序修改。
column_name DESC	按列名降序修改。

注意事项

不管是多表还是单表更新都不支持直接对子查询进行更新值操作，例如：`update (select * from t1) set c1 = 100;`

示例

1. 创建示例表 t1 和 t2。

```
OceanBase(admin@test)>create table t1(c1 int primary key, c2 int);
Query OK, 0 rows affected (0.16 sec)
OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.06 sec)

OceanBase(admin@test)>create table t2(c1 int primary key, c2 int) partition by key(c1) partitions 4;
Query OK, 0 rows affected (0.19 sec)
OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  5 |    5 |
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
+----+------+
4 rows in set (0.02 sec)
```

2. 将表 t1 中 "t1.c1=1" 对应的那一行数据的 c2 列值修改为 100。

```
OceanBase(admin@test)>update t1 set t1.c2 = 100 where t1.c1 = 1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  100 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.01 sec)
```

3. 将表 t1 中按照 c2 列排序的前两行数据的 c2 列值修改为 100。

```
OceanBase(admin@test)>update t1 set t1.c2 = 100 order by c2 limit 2;
Query OK, 2 rows affected (0.02 sec)
Rows matched: 2  Changed: 2  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  100 |
|  2 |  100 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.01 sec)
```

4. 将表 t2 中 p2 分区的数据中 "t2.c1 > 2" 的对应行数据的 c2 列值修改为 100。

```
OceanBase(admin@test)>update t2 partition(p2) set t2.c2 = 100 where t2.c1 > 2;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  5 |    5 |
|  1 |    1 |
|  2 |    2 |
|  3 |  100 |
+----+------+
4 rows in set (0.06 sec)
```

5. 修改多个表。将 t1 表和 t2 表中满足 "t1.c1 = t2.c1" 对应行的数据 t1 表中的 c2 列值修改为 100，t2 表中的 c2 列值修改为 200。

```
OceanBase(admin@test)>update t1,t2 set t1.c2 = 100, t2.c2 = 200 where t1.c2 = t2.c2;
Query OK, 6 rows affected (0.03 sec)
Rows matched: 6  Changed: 6  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  100 |
|  2 |  100 |
|  3 |  100 |
|  4 |    4 |
+----+------+
4 rows in set (0.00 sec)

OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  5 |    5 |
|  1 |  200 |
|  2 |  200 |
|  3 |  200 |
+----+------+
4 rows in set (0.01 sec)
```

6. 修改多个表。修改 t1 表和 t2 表的p2分区中满足 "t1.c1 = t2.c1" 对应行的数据 t1 表中的 c2 列值修改为 100，t2 表中的 c2 列值修改为 200。

```
OceanBase(admin@test)>update t1,t2 partition(p2) set t1.c2 = 100, t2.c2 = 200 where t1.c2 = t2.c2;
Query OK, 6 rows affected (0.02 sec)
Rows matched: 6  Changed: 6  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  100 |
|  2 |  100 |
|  3 |  100 |
|  4 |    4 |
+----+------+
4 rows in set (0.01 sec)

OceanBase(admin@test)>select * from t2;
+----+------+
| c1 | c2   |
+----+------+
|  5 |    5 |
|  1 |  200 |
|  2 |  200 |
|  3 |  200 |
+----+------+
4 rows in set (0.01 sec)
```

7. 对可更新视图 v 进行更新值。

```
OceanBase(admin@test)>create view v as select * from t1;
Query OK, 0 rows affected (0.07 sec)

OceanBase(admin@test)>update v set v.c2 = 100 where v.c1 = 1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

OceanBase(admin@test)>select * from v;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  100 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.01 sec)
```

6.SQL 参考（Oracle 模式）

6.1. 与 Oracle 兼容性对比

OceanBase 支持 Oracle 中绝大部分的基本 SQL 语法。这意味着在从 Oracle 过渡到 OceanBase 的过程中，您不需要消耗大量的时间去学习新的语法，并且可以流畅的实现从 Oracle 到 OceanBase 的迁移。

基于优化和开发的考虑，有一些功能 OceanBase 暂不支持或者是与 Oracle 的表现有所差异。所以本篇文档中将分节对比 OceanBase 对 Oracle 的兼容。

SQL 语法

- OceanBase 支持 Oracle 中基本 SQL 语法。
- 少数功能性缺失会报语法不支持的错误，例如在层次查询包含多表连接的场景下就会报错。
- OceanBase 暂不支持 Oracle 部分较复杂的 OLAP 语法，如模式匹配、 `PIVOT` / `UNPIVOT` 函数、多态表函数和频繁项目集计算。

SQL 数据类型

- Oracle 中有 24 个数据类型， OceanBase 目前支持 18 种，详细信息请参阅章节 内建数据类型 。
- 基于优化的考虑， `LONG` 和 `LONG RAW` 数据类型过于老旧，所以 OceanBase 不计划支持这两种数据类型。
- OceanBase 中大对象数据类型有 48 M 的大小限制且性能不佳，所以不推荐在复杂场景下使用。详细信息请参阅章节 大对象数据类型 。

字符集和 Collation

- OceanBase 支持 UTF-8、GBK、GB18030 和国家字符集。
- OceanBase 仅支持大小写敏感的 Collation。
- OceanBase 不支持多语言语义的 Collation 比较和排序。

内建函数

Oracle 中支持内建函数 117 个，OceanBase 暂时支持 103 个，详细信息请参阅章节 函数 。

系统视图

- Oracle 中有字典视图 400 多个，OceanBase 目前兼容 17 个：
  - ALL_CONS_COLUMNS
  - ALL_CONSTRAINTS
  - ALL_IND_COLUMNS
  - ALL_INDEX
  - ALL_OBJECTS
  - ALL_PART_KEY_COLUMNS
  - ALL_PART_TABLES
  - ALL_SEQUENCE
  - ALL_SOURCE

  - ALL_SUBPART_KEY_COLUMNS
  - ALL_SYNONYMS
  - ALL_TAB_COLUMNS
  - ALL_TAB_PARTITIONS
  - ALL_TABLES
  - ALL_TYPES
  - ALL_USERS
  - ALL_VIEWS
- Oracle 中有性能视图 700 多个，OceanBase 目前兼容 3 个。
  - V$SYSTEM_EVENT
  - V$SESSION_WAIT
  - V$NLS_PARAMETERS

SQL 功能

OceanBase 支持 Oracle 中核心的 SQL 功能：

- 计划缓存
- 大纲绑定
- 计划管理演进
- 代价优化器
- 代价改写
- 预编译语句
- 全局索引
- 函数索引

外键

- OceanBase 支持外键。
- OceanBase 不支持添加外键约束 `DISABLE` 和 `ENABLE` 。
- OceanBase 不支持 `ALTER TABLR` 语句中添加外键约束。
- OceanBase 不支持级联中的 `SET NULL` 。

触发器

- OceanBase 目前仅支持行级触发器。
- OceanBase 目前只支持在表上创建触发器，不支持在视图上创建触发器。
- OceanBase 不支持对触发器使用 `DISABLE` 和 `ENABLE` 操作。

数据库链接

- OceanBase 暂不支持数据库链接。

同义词

OceanBase 支持对表、视图、同义词和序列等对象创建同义词，并且支持创建公共同义词。

可更新视图

OceanBase 不支持 `WITH CHECK OPTION` 子句。

约束

- OceanBase 支持 `CHECK` 、 `UNIQUE` 和 `NOT NULL` 约束。
- OceanBase 不支持 `UNIQUE` 约束的 `DISABLE` 操作。

Hint

Oracle 中有 Hint 73 个，目前 OceanBase 兼容 25 个。另外，OceanBase 特有 Hint 20 个：

OceanBase 兼容 Oracle 的 Hint	OceanBase 特有的 Hint
USE_BNL	INDEX Hint
NO_USE_BNL Hint	FULL Hint
USE_PX Hint	LEADING Hint
NO_USE_PX Hint	ORDERED Hint
USE_JIT Hint	USE_MERGE Hint
NO_USE_JIT Hint	NO_USE_MERGE Hint
USE_HASH_AGGREGATION Hint	USE_HASH Hint
NO_USE_HASH_AGGREGATION Hint	NO_USE_HASH Hint
USE_LATE_MATERIALIZATION Hint	USE_NL Hint
NO_USE_LATE_MATERIALIZATION Hint	NO_USE_NL Hint
USE_NL_MATERIALIZATION Hint	PARALLEL Hint

OceanBase 兼容 Oracle 的 Hint	OceanBase 特有的 Hint
NO_USE_NL_MATERIALIZATION Hint	PQ_DISTRIBUTE Hint
PLACE_GROUP_BY Hint	NO_REWRITE Hint
NO_PLACE_GROUP_BY Hint	NO_EXPAND Hint
NO_PRED_DEDUCE Hint	USE_CONCAT Hint
READ_CONSISTENCY Hint	MERGE Hint
FROZEN_VERSION Hint	NO_MERGE Hint
QUERY_TIMEOUT Hint	UNNEST Hint
LOG_LEVEL Hint	NO_UNNEST Hint
USE_PLAN_CACHE Hint	QB_NAME Hint
TRANS_PARAM Hint	
TRACING Hint	
STAT Hint TOPK Hint	
TRACE_LOG Hint	

安全相关

透明加密

OceanBase 支持 Oracle 兼容的 TDE 功能， 重做日志文件（ Redo Log）加密暂不支持。

审计

- OceanBase 支持 Oracle 的标准审计，但是暂不支持统一审计。
- OceanBase 审计结果支持存放于文件或者内部审计表中。
- OceanBase 支持语句和对象审计类型，对象审计只支持表、序列和包对象, 其他对象暂不支持。

- OceanBase 暂不支持网络审计和 FGA 细粒度审计。

标签安全

- OceanBase 支持 Oracle 中的标签安全（Label Security）功能。
- OceanBase 暂不支持策略在创建生效后的 `DISABLE` 操作以及 `DISABLE` 后的 `ENABLE` 操作。

传输链路加密SSL

OceanBase 支持客户端与 OceanBase 服务器的传输链路加密。

分区支持

OceanBase 支持一级，二级分区，支持哈希（Hash）、范围（Range）和列表（List）等分区形式。

二级分区支持如下表：

分区/子分区	哈希	范围	列表
哈希	不支持	支持	支持
范围	支持	不支持	支持
列表	支持	支持	不支持

- OceanBase 中分区维护操作命令只支持基本的，例如一级分区 `ADD PARTITION` 操作。
- OceanBase 中暂不支持复杂分区维护操作，例如 `SPLIT`、`MERGE` 和 `EXCHANGE` 分区等。
- OceanBase 中不支持 `TRUNCATE` 分区。
- OceanBase 中二级分区只支持同构形态的，不支持异构形态的二级分区。

并行查询

- OceanBase 支持类 Oracle 的并行查询，OceanBase 中 DOP 需要手工指定，且暂不支持 Auto DOP 功能。
- OceanBase 暂不支持 PDML。

6.2. SQL 概述

结构化查询语言（Structured Query Language）简称 SQL，是一种有特殊目的的编程语言。和当下流行的其他关系数据库一样，所有程序和用户都可以使用 SQL 来访问 OceanBase 数据库中的数据。即便有一些平台、工具允许用户直接通过接口或界面的方式访问数据库，但这些平台、工具底层实际上依旧是使用 SQL 来访问数据库。

SQL 的历史

1970 年 6 月，IBM 公司 San Jose，California 实验室的 E. F. Codd 博士在 ACM（Association for Computing Machinery）期刊上发表了论文《大型共享数据银行的关系模型》（A Relational Model of Data for Large Shared Data Banks）并首次提出了关系模型的概念。

1974 年，同实验室的 D.D.Chamberlin 和 R.F.Boyce 在 IBM 公司研制的关系数据库系统 SystemR 中，研制出了一套规范语言 SEQUEL（Structured English QUEry Language），并在 1976 年 11 月的 IBM Journal of R&D 上公布了新版本的 SQL（称为 SEQUEL/2，1980年改名为 SQL）。

1979 年，Oracle 公司首先提供商用的 SQL，同时 IBM 公司在 DB2 和 SQL/DS 数据库系统中也实现了 SQL。

时至今日，SQL 已经成为了关系数据库管理系统（Relational Database Management System：RDBMS）的标准语言。

SQL 的标准

1986 年 10 月，美国国家标准协会 ANSI 采用 SQL 作为关系数据库管理系统的标准语言，并命名为 ANSI X3.135-1986，后来国际标准化组织（ISO）也采纳 SQL 作为国际标准。

1989 年，ANSI 采纳并使用了在 ANSI X3.135-1989 报告中定义的 SQL 标准语言，并称之为 ANSI SQL 89，该标准替代了之前的 ANSI X3.135-1986 版本。

下面是 SQL 发展的简要历史：

- 1986年，ANSI X3.135-1986，ISO/IEC 9075:1986，SQL-86。
- 1989年，ANSI X3.135-1989，ISO/IEC 9075:1989，SQL-89。
- 1992年，ANSI X3.135-1992，ISO/IEC 9075:1992，SQL-92（SQL2）。
- 1999年，ISO/IEC 9075:1999，SQL:1999（SQL3）。
- 2003年，ISO/IEC 9075:2003，SQL:2003。
- 2008年，ISO/IEC 9075:2008，SQL:2008。
- 2011年，ISO/IEC 9075:2011，SQL:2011。

现在，绝大多数被提及的 SQL 标准，其中涉及的内容其实都是 SQL 92 里最基本、最核心的一部分。OceanBase 目前也遵循的是 SQL 92 标准。

SQL 的运行

SQL 是用来访问关系数据库，如 OceanBase、Oracle 和 MySQL 的接口，所有的 SQL 语句都是对数据库的指令。

通常，SQL 可以分为 5 个部分：

1. 数据查询语言 DQL（Data Query Language）：也称为数据检索语言，用以从表中获得数据，并描述怎样将数据返回给程序输出。DQL 并不改变数据库中存储的数据内容。
2. 数据操作语言 DML（Data Manipulation Language）：用以改变数据库中存储的数据内容，即增加、修改和删除数据。
3. 事务控制语言 TCL（Transaction Control Language）：保证数据库的完整性、一致性，在同一个事务中的 DML 语句要么同时成功，要么同时失败。
4. 数据控制语句 DCL（Data Control Language）：对数据访问权限控制的命令。可以控制特定账号对特定数据库资源的访问权限。
5. 数据定义语言 DDL（Data Definition Language）：对数据库中资源进行定义、修改和删除，如新建表和删除表等。

SQL 的移植性

SQL 是访问数据库的标准语言，所有的主要关系数据库都支持 SQL，因此所有用 SQL 编写的程序都是可移植的。通常进行少量的修改就可以从一个关系数据库移植到另一个关系数据库上。

词汇惯例

- **粗体** 表示与操作或以文本或词汇表定义的术语相关联的图形用户界面元素。
- 保留字、关键字、标识符和参数中的大小写不敏感。为方便阅读与识别，这些字会以大写形式书写。
- 在不同的编程环境中，SQL 语句终止方式不同。本文档中以分号 “;” 来标识一个 SQL 的结尾。
- `行内代码` 表示文档中引用的代码。
- 为了突出重要信息，本文档会加粗“说明”、“注意”和“重要”等文字。
- 本文档中可选参数文本用方括号括起，如 [-n, -quiet]。

6.3. 伪列

伪列（Pseudocolumn）的行为与表中的列相同，但并未存储具体数值。因此，伪列只具备读属性，您不可以对伪列进行插入、更新、删除的等行为。

> **注意**
>
> OceanBase 不支持 `ROWID` 伪列。

层次查询伪列

层次查询伪列仅在层次查询中有效，要在查询中定义层次结构关系，必须使用 `CONNECT BY` 子句。

CONNECT_BY_ISCYCLE 伪列

`CONNECT_BY_ISCYCLE` 伪列用来协助标记循环是从哪一行开始的。如当前行的子节点同时也是其祖先节点之一，则 `CONNECT_BY_ISCYCLE` 返回 1，否则返回 0。

`CONNECT_BY_ISCYCLE` 需要配合 `CONNECT BY` 子句的 `NOCYCLE` 使用，否则查询结果会因树状结果存在循环而报错。

CONNECT_BY_ISLEAF 伪列

`CONNECT_BY_ISLEAF` 伪列用来协助标记层次结构的叶子节点。如当前行无子节点即为树的叶子节点时，返回 1，否则返回 0。

LEVEL 伪列

`LEVEL` 伪列用来协助标记节点的层次。层次结构中，根为第 1 层，根的子结点为第 2 层，之后以此类推。例如，根节点的 `LEVEL` 值会返回 1，根节点的子节点的 `LEVEL` 值会返回 2，之后以此类推。

序列伪列

序列（Sequence）伪列是数据库按照一定规则生成的自增数字序列。因其自增的特性，通常被用作主键和唯一键。序列伪列有两种取值方法：

- `CURRVAL`：返回序列的当前值。
- `NEXTVAL`：返回序列的下一个自增值。

使用序列伪列时，必须在 `CURRVAL`、`NEXTVAL` 前带上序列的名称，并用句点（.）引用。例如，序列的名称为 SEQ_FOO，则可以通过 `SEQ_FOO.CURRVAL` 获取 SEQ_FOO 序列的当前值。同样，可以通过 `SEQ_FOO.NEXTVAL` 获取 SEQ_FOO 序列的下一个自增值。

序列伪列的应用场景

序列伪列 CURRVAL 和 NEXTVAL 的值可以用于以下位置：

- 非子查询、物化视图或者视图中的 SELECT 语句的选择列表中。
- INSERT 语句中子查询的选择列表中。
- INSERT 语句中的 VALUE 子句中。
- UPDATE 语句中的 SET 子句中。

序列伪列 CURRVAL 和 NEXTVAL 的值不能用于以下位置：

- DELETE 、 SELECT 或者 UPDATE 语句的子查询中。
- 视图或者物化视图的查询中。
- 带 DISTINCT 运算符的 SELECT 语句中。
- 带 GROUP BY 子句或者 ORDER BY 子句的 SELECT 语句中。
- 与另一个 SELECT 语句通过 UNION 、 INTERSECT 或者 MINUS 集合运算符进行联合的 SELECT 语句中。
- SELECT 语句的 WHERE 子句中。
- CREATE TABLE 或者 ALTER TABLE 语句中列的 DEFAULT 值。
- CHECK 约束的条件中。

如何使用序列伪列

创建序列时，需要明确其初始值和步长。第一次引用 NEXTVAL 将返回序列的初始值，后续对 NEXTVAL 的引用将按照上一次序列的返回值加上序列定义的步长后返回一个新值。任何时候对 CURRVAL 的引用，都将返回当前序列的值，即最后一次对 NEXTVAL 引用时返回的值。

在会话中引用序列的 CURRVAL 伪列前，都应首先应用序列的 NEXTVAL 伪列来初始化本次会话的序列值。

创建序列时，可以定义其初始值以及其值之间的增量。对 NEXTVAL 的第一次引用将返回序列的初始值。对 NEXTVAL 的后续引用将会使序列值按照定义的增量递增，并返回新值。任何对 CURRVAL 的引用总是返回该序列的当前值，即最后一次对 NEXTVAL 引用时返回的值。对序列的创建的相关内容，请参考文档 CREATE SEQUENCE 章节。

在单条 SQL 语句中引用 NEXTVAL 时，OceanBase 按照以下方式递增序列：

- SELECT 语句的外部查询块每返回一行递增一次。这类查询块可以出现在以下地方：
  - 顶层 SELECT 语句。
  - INSERT... SELECT 语句。对于多表插入， NEXTVAL 必须位于 VALUES 子句中，子查询每返回一行序列就递增一次，即使多个分支引用了 NEXTVAL 。
  - CREATE TABLE ... AS SELECT 语句。
  - CREATE MATERIALIZED VIEW ... AS SELECT 语句。

- UPDATE 语句每更新一行序列就递增一次。
- 每有一条包含 VALUES 子句的 INSERT 语句就递增一次。
- MERGE 语句每合并一行序列递增一次。NEXTVAL 可以出现在 merge_insert_clause 或者 merge_update_clause 子句中，也可两者同时出现。NEXTVAL 会随着每一行的更新和插入而递增，即使序列数值没有用于更新或者插入操作。如果 NEXTVAL 在这些位置中被指定了多次，那么对应每一行都递增一次序列，而且该行中出现的所有 NEXTVAL 都返回相同的值。

当这些位置多次引用一个序列的 NEXTVAL 伪列时，该序列都只递增一次，即为所有被引用的 NEXTVAL 伪列返回当前序列的下一个序列值。

当这些位置同时引用一个序列的 CURRVAL 和 NEXTVAL 伪列时，OceanBase 将递增该序列，即为被引用的 CURRVAL 和 NEXTVAL 伪列都返回当前序列的下一个序列值。

序列可以同时被许多用户访问，不存在等待和锁定。

ROWSCN 伪列

ORA_ROWSCN 伪列将最新更改的系统更改号（SCN：System Change Number）反映到一行，该更改号表示这一行数据修改所在事务的提交时间。

ROWNUM 伪列

ROWNUM 伪列会对查询结果中的每一行进行编号，其值为该行在查询结果集中的具体位置。第一行返回值 1，第二行返回值 2，之后以此类推。

ROWNUM 的用法说明

ROWNUM 可以限制返回的行数，例如以下示例，将返回 employees 表中的 5 条数据：

```
SELECT * FROM employees WHERE rownum <=5;
```

如果在 ROWNUM 后有 ORDER BY 子句，则将对满足 WHERE 条件句的结果进行重排序。如果将 ORDER BY 子句嵌入子查询中，并将 ROWNUM 伪列作为条件放置在顶级查询中，则可以强制 ROWNUM 条件在行排序之后执行。例如，使用以下语句查询年龄最大的 5 名员工信息是得不到预期结果的，该语句只是将查询结果中的前 5 条员工信息进行年龄排序：

```
SELECT * FROM employees WHERE rownum <=5 ORDER BY age DESC;
```

正确的用法应该如下：

```
SELECT * FROM (SELECT * FROM employees ORDER BY age DESC) WHERE rownum <= 5;
```

在 WHERE 子句中，指定 ROWNUM 大于任何一个正整数时总是返回 FALSE，例如以下 SQL 语句将不返回任何信息：

```
SELECT * FROM employees WHERE rownum > 1;
```

因为在获得表的第一行结果时，改行的 ROWNUM 伪列值将被赋值为 1，此时在 WHERE 条件判断时结果为 FALSE，则此行被舍去。在获得第二行结果时，该行的 ROWNUM 伪列值任然被赋值为 1， WHERE 条件判断的结果依旧为 FALSE，此行再次被舍去。以此类推，所以所有行都不满足条件，因此不返回任何数据。

也可以通过 UPDATE 语句将 ROWNUM 数值赋值给表中的某一列，例如：

```
UPDATE employees SET id = rownum;
```

此语句将对表 employees 的 id 列进行 ROWNUM 赋值，即依次对 id 列赋值 1、2、... 直至该表总行数。

> 注意
>
> 在查询中使用 ROWNUM 可能会影响视图优化。

6.4. 基本元素

6.4.1. 内建数据类型

6.4.1.1. 内建数据类型概述

OceanBase 数据库操作的每个值都有一个数据类型。值的数据类型将一组固定的属性与值相关联，这些属性使 OceanBase 将一种数据类型的值与另一种数据类型的值区别对待。OceanBase 数据库提供了许多内建数据类型， 这些内建数据类型也称 OceanBase 基本数据类型。

数据类型

OceanBase 支持如下数据类型，与 Oracle 数据类型保持一致：

- 字符数据类型
- 数值数据类型
- 日期时间类型
- RAW 数据类型
- 大对象数据类型

6.4.1.2. 字符数据类型

6.4.1.2.1. 字符数据类型概述

字符数据类型在数据库字符集或国家字符集中存储字符（字母数字）数据，即单词和自由格式的文本。字符数据类型与其他数据类型相比具有限制性，因此属性较少。

字符数据存储在字符串中，其字节值与创建数据库时指定的字符集之一相对应。OceanBase 支持单字节和多字节字符集。

> **说明**
>
> 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。

数据类型	长度	使用说明	长度说明
CHAR(size [BYTE | CHAR])	定长	索引效率高，程序里面使用 trim 去除多余的空白。	参数 size 必须是一个介于 1~2000 之间的数值，存储大小为 size 个字节。
NCHAR[(size)]	定长	使用 UNICODE 字符集（所有的字符使用两个字节表示）。	参数 size 必须是一个介于 1~2000 之间的数值，存储大小为 size 字节的两倍。
NVARCHAR2(size)	变长	使用 UNICODE 字符集（所有的字符使用两个字节表示）。	参数 size 的值必须介于 1~32767 之间，字节的存储大小是所输入字符个数的两倍。
VARCHAR2(size [BYTE | CHAR])	变长	使用 UNICODE 字符集（所有的字符使用两个字节表示）。	参数 size 必须是一个介于 1~32767 之间的数值，存储大小为输入数据的字节的实际长度，而不是 size 个字节。
VARCHAR(size [BYTE | CHAR])	变长	OceanBase 中 VARCHAR 和 VARCHAR2 没有区别。	参数 size 必须是一个介于 1~32767 之间的数值，存储大小为输入数据的字节的实际长度，而不是 size 个字节。

其中， CHAR 和 VARCHAR2 数据类型需要指定 length 语义，其默认值由系统变量 NLS_LENGTH_SEMANTICS 控制。

6.4.1.2.2. CHAR 数据类型

CHAR 数据类型指定了固定长度的字符串。OceanBase 确保存储在 CHAR 数据类型列中的所有值的长度等于 size 指定的固定长度。如果您插入的值短于指定长度，则 OceanBase 将用空格填充剩余长度。如果您插入的值超过列长度，则 OceanBase 会返回错误。

BYTE 和 CHAR 限定符会覆盖由 NLS_LENGTH_SEMANTICS 参数指定的语义，而该参数默认字节语义。为了确保具有不同字符集的数据库之间，能够进行正确的数据转换， CHAR 数据必须由格式正确的字符串组成。

语法

```
CHAR [(size [BYTE | CHAR])]
```

参数

参数	说明
size	表示指定的固定长度
BYTE	表示以字节（BYTE）为单位提供列的长度
CHAR	表示以字符（CHAR）为单位提供列的长度

CHAR 数据类型的列的默认长度为 1 个字节，允许的最大长度为 2000 个字节。

示例

当创建带有 CHAR 数据类型列的表时，默认情况下，您以字节为单位提供列的长度。 BYTE 与默认值相同。

如果使用 CHAR 限定符。则以字符为单位提供列长。

- **示例 1**

将一个 1 字节的字符串插入 CHAR(10) 类型的列中，在存储该字符串之前，将为其空白填充为 10 个字节。

- **示例 2**

CHAR(10 CHAR) ，表示以字符为单位提供列长。

6.4.1.2.3. NCHAR 数据类型

NCHAR 是 UNICODE 字符数据类型，长度固定。您在创建数据库时，列的最大长度由国家字符集定义确定。创建包含 NCHAR 数据类型列的表时，以字符定义列长度。字符数据类型 NCHAR 的宽度规范是指字符数。允许的最大列大小为 2000 字节。

如果希望用较少的空间来存储中文，可以选择 NCHAR 类型。

使用 NCHAR 来存储数据时，如果存储的数据没有达到指定长度，那么数据库将自动补足空格。指定长度时，采用 CHAR 为计量单位，不可以手动指定其他单位。

> 注意
>
> 您不能将 CHAR 值插入 NCHAR 列，也不能将 NCHAR 值插入 CHAR 列。

语法

```
NCHAR[(size)]
```

参数

参数	说明
size	表示定长字符数据的长度大小，最大大小由国家字符集定义确定，上限为 2000 个字节。默认定长字符数据的长度最小为 1 个字符。

更多信息

UNICODE 字符集

UNICODE 字符集是对字符的一种编码，具体的编码方式有 UTF-8、UTF-16、UTF-32 和压缩转换等，编码方式决定了一个字符的存储大小，中英文在不同的存储方式上使用的空间是不一样的。

三种编码方式的对比

编码方式	编码字节数	BOM	优点	缺点
UTF-8	不定长编码方式，单字节（ASCII 字符）或多字节（非 ASCII 字符）；最小 Code Unit 是 8 位。	无字节序：如果一个文本的开头有字节流 EF BB BF，表示是 UTF-8 编码。	较为理想的 UNICODE 编码方式：与 ASCII 编码完全兼容；无字节序；自同步和纠错能力强，适合网络传输和通信；扩展性好。	变长编码方式不利于程序内部处理。
UTF-16	双字节或者四字节；最小 Code Unit 是 16 位。	有字节序：UTF-16LE（小端序）以 FF FE 代表，UTF-16BE（大端序）以 FE FF 代表。	最早的 Unicode 编码方式，已被应用于大量环境中；适合内存中 Unicode 处理；很多编程语言的 API 中作为 string 类型的编码方式。	无法兼容于 ASCII 编码；增补平面码点编码使用代理对，编码复杂；扩展性差。

UTF-32	固定四字节；最小 Code Unit 是 16 位。	有字节序：UTF-16LE（小端序）以 FF FE 代表，UTF-16BE（大端序）以 FE FF 代表。	固定字节编码读取简单，编译程序内部处理；Unicode 码点和 Code Unit 一一对应关系。	所有字符以固定四字节编码，浪费存储空间和带宽；与 ASCII 编码不兼容；扩展性差；实际使用少。

数据库字符集

- 用来存储 `CHAR` 、 `VARCHAR2` 、 `CLOB` 等类型数据。
- 用来标示诸如表名、列名以及 PL/SQL 变量等。
- 用来存储 SQL 和 PL/SQL 程序单元等。

国家字符集

- 用来存储 `NCHAR` 、 `NVARCHAR2` 、 `NCLOB` 等类型数据。
- 国家字符集实质上是为 OceanBase 选择的附加字符集，主要作用是为了增强 OceanBase 的字符处理能力， `NCHAR` 数据类型使用国家字符集，在使用 `CHAR` 数据类型提供的数据库字符集的同时， `NCHAR` 类型提供了除了数据库字符集之外的另一种字符集选择。

6.4.1.2.4. NVARCHAR2 数据类型

`NVARCHAR2` 是 `UNICODE` 字符数据类型。列的长度可变，最大存储长度为 32767 个字节，最小值为 1 个字节。创建包含 `NVARCHAR2` 数据类型列的表时，您必须为 `NVARCHAR2` 指定可以容纳的最大字符数，默认采用 `CHAR` 为计量单位，不可以手动指定其他单位。

如果不确定存储的数据长度，而且有可能包含中文，可以选择 `NVARCHAR2` 类型。

格式

```
NVARCHAR2(size)
```

参数

参数	说明
size	表示列的长度，可变长度。您必须为 `NVARCHAR2` 指定大小，对于 AL16UTF16 编码，字节数最大为两倍，对于 UTF8 编码，字节数最大为三倍。字节数大小由国家字符集定义确定，上限为 32767 个字节。

6.4.1.2.5. VARCHAR2 数据类型

VARCHAR2 数据类型存储一个可变长度的字符串，最大长度为 32767。当您创建 VARCHAR2 列时，您必须为 VARCHAR2 列指定最大长度。尽管允许存储的实际字符串为零长度字符串（''），但该最大值必须至少为 1 个字节。OceanBase 会将每个值完全按照您指定的方式存储在列中。

> 注意
>
> - 从技术上讲，字符是数据库字符集的代码点。
> - 创建具有该列或属性，且在列或属性定义中未包含任何明确限定词的数据库对象时，长度语义由会话的 NLS_LENGTH_SEMANTICS 参数的值决定。
> - OceanBase 使用未填充的比较语义比较 VARCHAR2 值。
> - 具有不同字符集的数据库之间进行正确的数据转换时，必须确保 VARCHAR2 数据由格式正确的字符串组成。

语法

```
VARCHAR2(size [BYTE | CHAR])
```

参数

参数	说明
size	表示存储的字节数或字符数的长度大小。
BYTE	表示该列将具有字节长度的语义。
CHAR	指示该列将具有字符语义。

示例

您可以使用 CHAR 限定符，以字符为单位提供最大长度。

```
VARCHAR2(10 CHAR)
```

您可以使用 BYTE 限定符，以字节为单位明确给出最大长度。

```
VARCHAR2(10 BYTE)
```

6.4.1.2.6. VARCHAR 数据类型

VARCHAR 数据类型用于存储可变长度的字符串。当您创建 VARCHAR 数据类型的列时，您必须为 VARCHAR 列指定最大长度。尽管允许存储的实际字符串为零长度字符串（''），但该最大值必须至少为 1 个字节。OceanBase 会将每个值完全按照您指定的方式存储在列中。如果插入值超过指定长度，OceanBase 会返回错误。

在 OceanBase 中 VARCHAR 数据类型和 VARCHAR2 数据类型没有区别，一般会用 VARCHAR2 。

语法

```
VARCHAR(size [BYTE | CHAR])
```

参数

参数	说明
size	表示存储的字节数或字符数的长度大小。
BYTE	表示该列将具有字节长度的语义。
CHAR	指示该列将具有字符语义。

6.4.1.3. 数值数据类型

6.4.1.3.1. 数值数据类型概述

OceanBase 为我们提供了四种存储数值的数据类型，它们分别是 NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE 。您可以通过这四种数值类型存储定点数、浮点数和零。在数值计算时，数值类型具有不同的优先级，具体信息请查阅 数值类型的优先级。

数据类	长度（字节）	说明
NUMBER	4~40	NUMBER(p,s) 存储变长、十进制精度的定点数；也可以存储浮点数，此时 NUMBER 没有 p 和 s 。
FLOAT	4~40	FLOAT(p) 是 NUMBER 数据类型的子类型。二进制精度范围为 1~126。 FLOAT 不是浮点数。
BINARY_FLOAT	4	二进制精度浮点数，是一种 32 位单精度浮点数数据类型。

数据类	长度（字节）	说明
BINARY_DOUBLE	8	二进制精度浮点数，是一种 64 位双精度浮点数数据类型。

6.4.1.3.2. NUMBER 数据类型

NUMBER 是变长、精确的数值类型，占 4~40 字节存储空间，其中 4 字节存放 NUMBER 的元数据信息，36 字节存放 NUMBER 的具体数值。可以存储零、浮点数、正的定点数和负的定点数，其绝对值范围 1.0×10^{-130} ~ 1.0×10^{126}（不包括 1.0×10^{126}）。如果您指定的算术表达式的绝对值大于或等于 1.0×10^{-130}，OceanBase 会返回错误。

NUMBER 数据类型具有较好的数据精确度、通用性强、可移植性强，运算效率相对浮点类型偏低。

语法

```
NUMBER [(p[s])]
```

参数

参数	取址范围	说明
p	1~38	表示精度，最大有效十进制数字，其中最高有效数字是最左边的非零数字，最低有效数字是最右边的已知数字。
s	-84~127	表示小数位数，从小数点到最低有效数字的位数。比例尺范围是 -84~127。

说明

- 若 s>0，精确到小数点右边 s 位，四舍五入。然后检验有效位是否小于等于 p 。
- 若 s<0，精确到小数点左边 s 位，四舍五入。然后检验有效位是否小于等于 p + |s| 。
- 若 s=0， 表示整数。

> 注意
>
> - 小数位数正标度是小数点右边到最低有效位数（包括最低有效位数）的有效位数。精度和小数位都用十进制数字表示。
> - 小数位数负数标度是小数点左边的有效位数，但不包括最低有效位数。对于负比例，最低有效数字在小数点的左侧，因为实际数据四舍五入到小数点左侧的指定位数。

示例

- **示例 1：** 使用以下格式指定整数。

`NUMBER(p)` 表示精度为 `p` 且标度为 0 的定点数，等效于 `NUMBER(p,0)` 。

`NUMBER` 表示浮点数，缺少精度和小数位数指示符。

- **示例 2：** 使用不同的精度和小数位数存储数据。为了避免 OceanBase 存储的数据超过精度，我们需要为定点数字列指定小数位数和精度，对输入进行额外的完整性检查。但这不会强制固定定点数字列的长度。若实际存储的数据超过精度，则 OceanBase 会返回错误。如果存储的数据超过小数位数，则 OceanBase 对其进行四舍五入。

实际数据	指定为	存储为
123.89	NUMBER	123.89
123.89	NUMBER(3)	124
123.89	NUMBER(3,2)	超过精度
123.89	NUMBER(4,2)	超过精度
123.89	NUMBER(5,2)	123.89
123.89	NUMBER(6,1)	123.9
123.89	NUMBER(6,-2)	100
.01234	NUMBER(4,5)	.01234
.00012	NUMBER(4,5)	.00012
.000127	NUMBER(4,5)	.00013

实际数据	指定为	存储为
.000012	NUMBER(2,7)	.000012
.0000123	NUMBER(2,7)	.000012
1.2e-4	NUMBER(2,5)	.00012
1.2e-5	NUMBER(2,5)	.00001

6.4.1.3.3. FLOAT 数据类型

FLOAT 数据类型是具有精度（precision）的 NUMBER 数据类型的子类型，需要占 4~40 字节存储空间。它的精度是按二进制有效位数计算的，范围为 1~126，小数位数不可自定义。FLOAT 类型为变长、非精确数值类型。

语法

```
FLOAT [(p)]
```

参数

参数	定义	范围	说明
p	精度	1~126	定义数值精度，按二进制有效位数计算，转换为十进制精度要乘以 0.30103。

> 说明
>
> - 二进制精度转换为十进制精度的换算关系：二进制精度 = int(十进制精度 x 0.30103) 。
> - 十进制精度转换为二进制精度的换算关系：十进制精度 = int(二进制精度 x 3.32193) 。

示例

- **示例 1：** 使用 FLOAT 设置二进制精度为 2，转换为十进制精度 int(2 x 0.30103) = 0.6，结果向下取整，则 FLOAT(2) 的十进制精度为 0。

```
FLOAT(2)
```

- **示例 2**：创建 test 表，并向里面插入数据。其中 col1 列是 NUMBER 类型，col2 列是 FLOAT 类型。NUMBER(5,2) 表示十进制精度的定点数，有效位数为 5，结果保留小数点后 2 位。FLOAT(5) 的二进制精度为 5，转换为十进制精度为 int(5 x 0.30103) = 1.50515，向下取整后十进制精度为 1。如 123.45 用科学计数法表示为 1.2345×10^2，1.2345 小数点后保留 1 位，四舍五入变为 1.2，最后显示为 $1.2 \times 10^2 = 120$。执行以下语句：

```
CREATE TABLE test (col1 NUMBER(5,2), col2 FLOAT(5));
INSERT INTO test VALUES (1.23, 1.23);
INSERT INTO test VALUES (7.89, 7.89);
INSERT INTO test VALUES (12.79, 12.79);
INSERT INTO test VALUES (123.45, 123.45);
```

执行以下语句查看 test 表：

```
SELECT * FROM test;
```

返回结果如下：

```
+-----------+----------+
|col1       | col2 |
+-----------+----------+
|  1.23 |  1.2  |
+-----------+----------+
|  7.89 |  7.9  |
+-----------+----------+
|  12.79 |  13   |
+-----------+----------+
|  123.45|  120  |
+-----------+----------+
```

> **说明**
>
> 当转换 ANSI FLOAT 数据时，您可以使用 OceanBase 数据库内部使用的 FLOAT 数据类型。但是 OceanBase 建议您改用 BINARY_FLOAT 和 BINARY_DOUBLE 浮点数字。

6.4.1.3.4. 浮点数字

浮点数字（FlOAT-Point Numbers）可以有小数点，从第一位到最后一位，或者根本没有小数点。您可以选择在数字后使用指数来增加范围，例如 $1.666\ e^{-20}$。小数位数不适用于浮点数，因为小数点后可以出现的位数不受限制。

> **注意**
>
> 二进制浮点数与 NUMBER 的区别，在于值是由 OceanBase 数据库内部存储的。NUMBER 数据类型使用十进制精度存储，所存储的数据都会精确存储。二进制浮点数（FlOAT-Point Numbers）是使用二进制精度（数字 0 和 1）存储的，这种存储值的方式不能精确地表达使用十进制精度的所有值。

语法

OceanBase 数据库为浮点数提供了两种数字数据类型：

- BINARY_FLOAT，是一种 32 位单精度浮点数数据类型。每个 BINARY_FLOAT 值需要 4 个字节。
- BINARY_DOUBLE，是一种 64 位双精度浮点数数据类型。每个 BINARY_DOUBLE 值需要 8 个字节。

> **说明**
>
> - 在 NUMBER 列中，浮点数具有十进制精度。
> - 在 BINARY_FLOAT 或 BINARY_DOUBLE 列中，浮点数具有二进制精度。
> - 二进制浮点数暂时不支持特殊值 infinity 和 NaN（不是数字）。

取值范围

您可以根据值域范围指定浮点数。

值	BINARY_FLOAT	BINARY_DOUBLE
最大正有限值	3.40282E+38F	1.79769313486231E+308
最小正有限值	1.17549E-38F	2.22507485850720E-308

更多信息

IEEE754 符合性

- IEEE 标准 754-1985（IEEE754）

OceanBase 浮点数据类型的实现基本上符合电气和电子工程师协会（IEEE）的二进制浮点算法标准。

- 浮点数据类型在以下方面符合 IEEE754
  - SQL 函数 SQRT 实现平方根。
  - SQL 函数 REMAINDER 实现余数。
  - 算术运算符一致。
  - 比较运算符符合要求。
  - 转换运算符符合。
  - 支持默认的舍入模式。

- 支持默认的异常处理模式。
- OceanBase 暂时不支持 INF 、 -INF 、 NaN 、 BINARY_FLOAT_NAN 、 BINARY_DOUBLE_NAN 之类的特殊常量。
- SQL 函数 ROUND 、 TRUNC 、 CEIL 和 FLOOR 提供了将二进制浮点数 BINARY_FLOAT 、 BINARY_DOUBLE 舍入为整数值 BINARY_ FLOAT 和 BINARY_DOUBLE 的功能。
- SQL 函数 TO_CHAR 、 TO_NUMBER 、 TO_ NCHAR 、 TO_BINARY_FLOAT 、 TO_BINARY_DOUBLE 和 CAST 提供了将二进制浮点数 BINARY_FLOAT 、 BINARY_DOUBLE 舍入为十进制数和十进制数舍入为 BINARY_FLOAT 、 BINARY_DOUBLE

  的功能。

- 浮点数据类型在以下方面不符合 IEEE754：
  - -0 被强制为 +0。
  - 不支持与 NaN 比较。
  - 不支持非默认的舍入模式。
  - 不支持非默认异常处理模式。

6.4.1.3.5. 数值类型的优先级

不同的数值数据类型在操作时有不同的优先级。在 OceanBase 中， BINARY_DOUBLE 的优先级最高，其次是 BINARY_FLOAT ，最后是 NUMBER 。

在对多个数值进行操作时：

- 若有一个操作数为 BINARY_DOUBLE ，OceanBase 会在执行该操作之前将所有操作数转换为 BINARY_DOUBLE 。
- 若有一个操作数为 BINARY_FLOAT ，OceanBase 会在执行该操作之前将所有操作数转换为 BINARY_FLOAT 。
- 若有一个操作数都不是 BINARY_DOUBLE 和 BINARY_FLOAT ，OceanBase 会在执行该操作之前将所有操作数转换为 NUMBER 。
- 若所需要的转换失败，则该操作将失败。
- 与其他数据类型相比，数值数据类型的优先级低于日期时间与间隔数据类型，高于字符和所有其他数据类型。

6.4.1.4. 日期时间和间隔数据类型

6.4.1.4.1. 日期时间和间隔数据类型概述

与 Oracle 中的数据类型一致，OceanBase 也支持日期时间和间隔数据类型。

日期时间数据类型

日期时间数据类型被用来在数据库中保存日期和时间，此类别中有以下数据类型：

数据类型	说明
DATE	存储日期和时间信息，精确到秒，不带时区。
TIMESTAMP	`DATE` 数据类型的扩展，精确到纳秒，不带时区。
TIMESTAMP WITH TIME ZONE	`DATE` 数据类型的扩展，精确到纳秒，带时区信息。
TIMESTAMP WITH LOCAL TIME ZONE	带有本地时区的 `TIMESTAMP`，存储数据库时区。

间隔数据类型

与日期时间数据类型存储特定的一个时间点不同，间隔数据类型用来存储一段时间。间隔数据类型有效的存储了两个日期时间值之间的差异，此类别中有以下数据类型：

数据类型	说明
INTERVAL YEAR TO MONTH	存储以年和月为单位的时间段。
INTERVAL DAY TO SECOND	存储以天、小时、分钟和秒为单位的时间段。

更多信息

- 夏令时
- 日期时间 DATE 和间隔 INTERVAL 的计算

6.4.1.4.2. DATE 数据类型

`DATE` 数据类型存储日期和时间信息。尽管日期和时间信息可以用字符和数字数据类型表示，但 `DATE` 数据类型具有特殊的关联属性。对于每个 `DATE` 值，OceaBase 存储以下信息：年、月、日、小时、分钟和秒，但是并不包含时区信息。

格式

`DATE` 数据类型的默认输入输出格式由 `NLS_DATE_FORMAT` 决定，运行以下 SQL 语句查看默认格式：

```
SELECT @@NLS_DATE_FORMAT FROM DUAL;
```

返回结果：

```
DD-MON-RR
```

如果您需要自定义数据的格式，可以使用转换函数。在插入数据时，您可以通过函数 `TO_DATE (char,fmt)` 指定数据的输入格式。查询数据时，您可以通过函数 `TO_CHAR (datetime,fmt)` 指定数据的输出格式。这两个转换函数会将字符串转化为参数 `fmt` 中定义的格式。在 `fmt` 未指明的情况下，使用默认格式。

> 注意
>
> `DATE` 数据类型储存小时、分钟和秒等时间信息，但是默认格式中并不包含时间信息。

取值范围

`0001-01-01 00:00:00` ~ `9999-12-31 23:59:59`

示例

- **示例1：** 返回当前的系统日期，由于未指定参数 `fmt`，所以 `TO_CHAR` 按数据类型的默认格式返回数据。

```
SELECT TO_CHAR(sysdate) FROM DUAL;
```

返回结果：

```
+------------------+
| TO_CHAR(SYSDATE) |
+------------------+
| 24-FEB-20        |
+------------------+
```

- **示例2：** 在未指定 `DATE` 字面量的情况下，数据库将返回系统默认值：
  - 年：当前年份，由 `SYSDATE` 返回。
  - 月：当前月份，由 `SYSDATE` 返回。
  - 日：01，默认是当前月的第一天。
  - 小时、分钟和秒均为 0。

本示例将使用 `TO_DATE(string, format)` 插入日期数据。`TO_DATE` 会将 `string` 中的字符串转化为 `format` 中定义的格式。

例如当前查询是 2020 年 2 月发出的：

```
SELECT TO_CHAR(TO_DATE('2020', 'YYYY'),'YYYY-MM-DD HH24:MI:SS') FROM DUAL;
```

返回结果：

```
+-----------------------------------------------------+
| TO_CHAR(TO_DATE('2020', 'YYYY'),'YYYY-MM-DD HH24:MI:SS') |
+-----------------------------------------------------+
|                        2020-02-01 00:00:00|
+-----------------------------------------------------+
```

6.4.1.4.3. TIMESTAMP 数据类型

日期时间数据类型中除了 DATE 数据类型以外还有 TIMESTAMP[(scale)] 时间戳数据类型。它是 DATE 数据类型的扩展，跟 DATE 数据类型一样，它存储了年、月、日、时、分和秒等信息，但是不存储时区信息，它的时间最大可以精确到纳秒。所以它常用于存储时间精确度高和不需要考虑时区变换的数据。

语法

```
TIMESTAMP [(scale)]
```

参数

参数	取值范围	说明
scale	0~9	TIMESTAMP[(scale)] 的范围和精确度取决于 scale 的值，最大值为 9（纳秒，即秒数精确至小数点 9 位），最小值为 0（秒，即秒数精确至小数点 0 位），默认值为 6。

格式

TIMESTAMP 数据类型的默认输入输出格式由 NLS_TIMESTAMP_FORMAT 决定，运行以下 SQL 语句查看默认格式：

```
SELECT @@NLS_TIMESTAMP_FORMAT FROM DUAL;
```

返回结果：

```
DD-MON-RR HH.MI.SSXFF AM
```

如果您需要自定义数据的格式，可以使用转换函数。在插入数据时，您可以通过函数 TO_TIMESTAMP (char,fmt) 指定数据的输入格式。查询数据时，您可以通过函数 TO_CHAR (datetime,fmt) 指定数据的输出格式。这两个转换函数会将字符串转化为参数 fmt 中定义的格式。在 fmt 未指明的情况下，使用默认格式。

取值范围

0001-01-01 00:00:00.000000000 ~ 9999-12-31 23:59:59.999999999

示例

- **示例1**：如下代码所示，在表 Timestamp_Sample 中创建了数据类型为 TIMESTAMP 的两列 timestp1 和 timestp2，并且为 timestp2 指定了时间的精度为 3。向两列同时使用 TO_TIMESTAMP(string, format) 插入日期值 2020-01-01 11:00:00。

```
CREATE TABLE Timestamp_Sample(timestp1 TIMESTAMP, timestp2 TIMESTAMP(3));
INSERT INTO Timestamp_Sample(timestp1,timestp2) VALUES(TO_TIMESTAMP('2020-01-01 11:00:00','YYYY-MM-DD HH24:MI:SS'),TO_TIMESTAMP('2020-01-01 11:00:00','YYYY-MM-DD HH24:MI:SS'));
SELECT * FROM Timestamp_Sample;
```

结果如下，可以看到 timestp1 由于没有指定 scale 的值，所以结果时间精度默认为 6 位，timestp2 在指定后精度为 3 位：

```
+-----------------------------+-------------------------+
| timestamp1                  | timestamp2              |
+-----------------------------+-------------------------+
| 01-JAN-20 11.00.00.000000 AM | 01-JAN-20 11.00.00.000 AM |
+-----------------------------+-------------------------+
```

- **示例2**：以下语句使用函数 TO_CHAR (datetime,fmt) 指定输出格式。

```
SELECT TO_CHAR(TO_TIMESTAMP_TZ('25-FEB-20 11:00:00 AM America/Los_Angeles','DD-MON-RR HH:MI:SSXFF PM TZR'),'YYYY-MM-DD HH:MI:SSXFF PM TZR') Timestamp
FROM DUAL;
```

返回结果：

```
+-----------------------------------------------------+
| Timestamp                                           |
+-----------------------------------------------------+
| 2020-02-25 11:00:00.000000000 AM America/Los_Angeles |
+-----------------------------------------------------+
```

6.4.1.4.4. TIMESTAMP WITH TIME ZONE 数据类型

TIMESTAMP [(scale)] WITH TIME ZONE 是 TIMASTAMP [(scale)] 的变体，与 TIMASTAMP [(scale)] 一样，它存储了年、月、日、时、分和秒等信息，阈值和精确度取决于 scale 的值。但是与 TIMASTAMP [(scale)] 不同的是它还可以存储时区信息，所以常用于存储跨地理区域的日期时间信息。

语法

```
TIMESTAMP [(scale)] WITH TIME ZONE
```

参数

参数	取值范围	说明
scale	0~9	TIMESTAMP [(scale)] WITH TIME ZONE 的阈值和精确度取决于 scale 的值，最大值为 9（纳秒，即秒数精确至小数点 9 位），最小值为 0（秒，即秒数精确至小数点 0 位），默认值为 6。

格式

TIMESTAMP WITH TIME ZONE 数据类型的默认输入输出格式由 NLS_TIMESTAMP_TZ_FORMAT 决定，运行以下 SQL 语句查看默认格式：

```
SELECT @@NLS_TIMESTAMP_TZ_FORMAT FROM DUAL;
```

返回结果：

```
DD-MON-RR HH.MI.SSXFF AM TZR
```

如果您需要自定义数据的格式，可以使用转换函数。在插入数据时，您可以通过函数 TO_TIMESTAMP_TZ (char,fmt) 指定数据的输入格式。查询数据时，您可以通过函数 TO_CHAR (datetime,fmt) 指定数据的输出格式。这两个转换函数会将字符串转化为参数 fmt 中定义的格式。在 fmt 未指明的情况下，使用默认格式。

取值范围

0001-01-01 00:00:00.000000000 ~ 9999-12-31 23:59:59.999999999

示例

以下示例都使用 TO_TIMESTAMP_TZ (char,fmt) 输入时间戳值。

在插入时区时， OceanBase 支持使用时区偏移量和时区区域名称：

- 时区偏移量：与格林尼治标准时间 GMT 的差（小时和分钟）
- 时区区域名称（TZR）及时区缩写（TZD）：国家/城市 时区缩写

使用时区偏移量

执行以下语句，使用时区偏移量插入数值：

```
SELECT TO_TIMESTAMP_TZ('2020-01-01 11:00:00 -05:00','YYYY-MM-DD HH:MI:SS TZH:TZM') FROM DUAL;
```

返回结果：

```
01-JUN-20 11.00.00.000000000 AM AMERICA/LOS_ANGELES
```

使用时区区域名称及时区缩写

执行以下语句，使用时区区域名称及缩写插入数值：

```
SELECT TO_TIMESTAMP_TZ('2020-01-01 11:00:00 America/Los_Angeles PST','YYYY-MM-DD HH:MI:SS TZR TZD
') FROM DUAL;
```

返回结果：

```
01-JUN-20 11.00.00.000000000 AM America/Los_Angeles PST
```

夏令时

OceanBase 支持夏令时且用时区缩写表示夏令时信息，以 America/Los_Angeles 为例，夏令时 PDT 从每年的 3 月的第二个星期日到 11 月的第一个星期日执行，其余时间为 PST。当插入的值中只包含时区区域名称时，OceanBase 会根据插入的时间信息判断当前插入的时区区域是否在夏令时时段，并在返回的结果中会包含时区缩写，以此指明当前时间是夏令时。

您可以执行以下示例代码：

```
SELECT TO_TIMESTAMP_TZ('2020-02-01 11:00:00 America/Los_Angeles','YYYY-MM-DD HH:MI:SS TZR') FROM
DUAL;
SELECT TO_TIMESTAMP_TZ('2020-06-01 11:00:00 America/Los_Angeles','YYYY-MM-DD HH:MI:SS TZR') FROM
DUAL;
```

返回结果：

```
01-JUN-20 11.00.00.000000000 AM America/Los_Angeles PST
01-JUN-20 11.00.00.000000000 AM America/Los_Angeles PDT
```

6.4.1.4.5. TIMESTAMP WITH LOCAL TIME ZONE 数据类型

`TIMESTAMP [(scale)] WITH LOCAL TIME ZONE` 数据类型的时区信息是当前会话发生的时区。与 `TIMASTAMP [(scale)] WITH TIME ZONE` 的区别在于用户不需要自己输入时区信息，OceanBase 直接存储默认的数据库时区 +00:00 （不可更改）。当用户检索数据时，OceanBase 会返回本地会话时区（可更改）。此数据类型常用于始终在两层应用程序中的客户端系统的时区中显示的日期信息。

语法

```
TIMASTAMP [(scale)] WITH TIME ZONE
```

参数

参数	取值范围	说明
scale	0~9	TIMESTAMP [(scale)] 的阈值和精确度取决于 scale 的值，最大值为 9（纳秒），最小值为 0（秒），默认值为 6。

格式

TIMESTAMP WITH LOCAL TIME ZONE 数据类型的默认输入输出格式由 NLS_TIMESTAMP_FORMAT 决定，运行以下 SQL 语句查看日期时间格式：

```
SELECT @@NLS_TIMESTAMP_FORMAT FROM DUAL;
```

返回结果：

```
DD-MON-RR HH.MI.SSXFF AM
```

如果您需要自定义数据的格式，可以使用转换函数。您可以通过函数 TO_CHAR (datetime,fmt) 指定数据的输出格式。这个转换函数会将字符串转化为参数 fmt 中定义的格式。在 fmt 未指明的情况下 TO_CHAR 按数据类型的默认格式返回数据。关于 TIMESTAMP WITH TIME ZONE 数据类型值的输入格式，您可以参阅文档 时间戳字面量。

取值范围

0001-01-01 00:00:00.000000000 ~ 9999-12-31 23:59:59.999999999

示例

TIMESTAMP WITH TIME ZONE 数据类型没有专门的字面量为其赋值，时区信息由 SESSIONTIMEZONE 返回本地对话时区。SESSIONTIMEZONE 的值取自自定义参数 TIME_ZONE。

```
CREATE TABLE LocalTZ ( ltzcol TIMESTAMP WITH LOCAL TIME ZONE);
INSERT INTO LocalTZ VALUES (TIMESTAMP '2020-02-25 11:10:08.123');
ALTER SESSION SET TIME_ZONE='+08:00';
SELECT SESSIONTIMEZONE, ltzcol FROM LocalTZ;
```

返回结果：

```
+-----------------+-----------------------------+
| SESSIONTIMEZONE | ltzcol                      |
+-----------------+-----------------------------+
| +08:00          | 25-FEB-20 11:10:08.123000 AM |
+-----------------+-----------------------------+
```

可通过更改自定义参数 TIME_ZONE 的值而改变 SESSIONTIMEZONE：

```
ALTER SESSION SET TIME_ZONE='+00:00';
SELECT SESSIONTIMEZONE, ltzcol FROM LocalTZ;
```

返回结果：

```
+-----------------+-----------------------------+
| SESSIONTIMEZONE | ltzcol                      |
+-----------------+-----------------------------+
| +00:00          | 25-FEB-20 03:10:08.123000 AM |
+-----------------+-----------------------------+
```

6.4.1.4.6. INTERVAL YEAR TO MONTH 数据类型

与 `DATE` 和 `TIMESTAMP` 数据类型储存特定的时间点日期不同， `INTERVAL YEAR TO MONTH` 使用日期元素 `YEAR` 和 `MONTH` 来储存一段时间，此数据类型对于表示两个日期时间值之间的差异很有用。

语法

```
INTERVAL YEAR [(percision)] TO MONTH
```

参数

参数	值	说明
percision	0~9	代表了 `YEAR` 元素的精度，默认值为 2。也就是说在不指定该参数的情况下，最大可以存储 99 年 11 个月的间隔日期 （最大不能超过 100 年）。如果存储的参数超过了默认精度 2 位，那么这个参数的值不能为空，必须被明确指定。

日期格式

在插入 `INTERVAL YEAR TO MONTH` 数据类型的值时，有以下几种格式，更多关于间隔数据类型值的指定请参阅 间隔字面量：

语法	示例	说明

INTERVAL 'year-month' YEAR(percision) TO MONTH	INTERVAL '120-3' YEAR(3) TO MONTH	间隔 120 年 3 个月。由于 YEAR 元素的值大于默认精度 2，所以必须指定 YEAR 元素的精度为 3.
INTERVAL 'year' YEAR(percision)	INTERVAL '50' YEAR	间隔 50 年。
INTERVAL 'month' MONTH	INTERVAL '500' MONTH	间隔 500 个月或者间隔 41 年 8 个月。

示例

如下代码所示，在表 Interval_Sample 中创建了数据类型为 INTERVAL YEAR TO MONTH 的三列interval1、interval2 和 interval3，并向其中插入数值：

```
CREATE TABLE Interval_Sample (
  interval1 INTERVAL YEAR TO MONTH,
  interval2 INTERVAL YEAR(3) TO MONTH,
  interval3 INTERVAL YEAR TO MONTH
);
INSERT INTO Interval_Sample (interval1, interval2, interval3)
VALUES (INTERVAL '12-3' YEAR TO MONTH, INTERVAL '120-3' YEAR(3) TO MONTH, INTERVAL '40' MONTH);
SELECT * FROM Interval_Sample;
```

返回结果：

```
+-----------+-----------+-----------+
| interval1 | interval2 | interval3 |
+-----------+-----------+-----------+
| +12-03    | +120-03   | +03-04    |
+-----------+-----------+-----------+
```

间隔与其他日期类型的计算

OceanBase 支持数据类型间的转换，所以间隔数据类型的值可以与其他日期值进行数学运算，但是数据库并不支持数据类型间任意的进行加、减、乘、除运算。请参阅文档 日期时间 DATE 和间隔 INTERVAL 的计算 中查看目前支持的日期类型计算矩阵图和 数据类型转换 了解数更多数据类型转换的信息。

- **示例 1**：间隔与间隔的计算，返回的值仍为间隔数据类型。

```
SELECT INTERVAL '2-2' YEAR TO MONTH -INTERVAL '1-1' YEAR  TO MONTH calculate1, INTERVAL '2-2' YEAR TO MONTH + INTERVAL '1-1' YEAR TO MONTH calculate2 FROM DUAL;
```

返回结果：

```
+---------------+---------------+
| calculate1    | calculate2    |
+-------------------------------+
| +000000001-01 | +000000003-03 |
+---------------+---------------+
```

- **示例 2**：间隔与日期时间的计算，返回的值为日期数据类型。`SYSDATE` 返回当前的时间 2020-02-27 16:13:50，以下示例会返回从现在起两个月后的日期值。数据库只支持间隔+日期时间，而间隔-日期时间是无效运算。但是日期时间+间隔和日期时间-间隔都是有效运算。

```
SELECT TO_CHAR(INTERVAL '2' MONTH +SYSDATE,'YYYY-MM-DD HH24:MI:SS') calculate3 FROM DUAL;
```

返回结果：

```
+---------------------+
| calculate3          |
+---------------------+
| 2020-04-27 16:13:50 |
+---------------------+
```

- **示例 3**：间隔和数字的运算，返回的值仍为间隔数据类型。

间隔可以与数字进行乘除运算。以下示例展示了间隔 2 月乘以 2 的计算和间隔 2 天除以 3 的计算。

```
SELECT INTERVAL '2' MONTH*2 calculate4, INTERVAL '2' DAY/3 calculate5 FROM DUAL;
```

返回结果为间隔 4 月和间隔 16 小时：

```
+---------------+------------------------------+
| calculate4    | calculate5                   |
+---------------+------------------------------+
| +000000000-04 | +000000000 16:00:00.000000000 |
+---------------+------------------------------+
```

6.4.1.4.7. INTERVAL DAY TO SECOND 数据类型

`INTERVAL DAY TO SECOND` 可存储以天、小时、分钟和秒为单位的时间段。此数据类型对于表示两个日期时间值之间的精确差异很有用。

语法

```
INTERVAL DAY [(precision)] TO SECOND [(fractional_seconds_precision)]
```

参数

参数	值	说明
precision	0~9	代表了 DAY 元素的精度，默认值为 2。
fractional_seconds_precision	0~9	代表了 SECOND 元素小数部分的精度，默认值为 6。

示例

在插入 INTERVAL DAY TO SECOND 数据类型的值时，有以下几种格式，更多关于间隔数据类型值的指定请参阅 间隔字面量：

语法	示例	说明
INTERVAL 'dd hh:mm:ss' DAY(precision) TO SECOND(fractional_seconds_precision)	INTERVAL '140 5:12:10.2222222' DAY(3) TO SECOND(7)	间隔 140 天 5 小时 12 分钟 10.2222222 秒。
INTERVAL 'dd hh'DAY(precision) TO HOUR	INTERVAL '400 5' DAY(3) TO HOUR	间隔 400 天 5 小时。
INTERVAL 'dd hh:mm'DAY(precision) TO MINUTE	INTERVAL '4 5:12' DAY TO MINUTE	间隔 4 天 5 小时 12 分钟。
INTERVAL 'hh:mm' HOUR TO MINUTE	INTERVAL '11:20' HOUR TO MINUTE	间隔 11 小时 20 分钟。
INTERVAL 'hh:mm:ss' HOUR TO SECOND(fractional_seconds_precision)	INTERVAL '11:12:10.2222222' HOUR TO SECOND(7)	间隔 11 小时 12 分钟 10.2222222 秒。
INTERVAL 'dd' DAY(precision)	INTERVAL '14' DAY	间隔 14 天。

语法	示例	说明
INTERVAL 'hh' HOUR	INTERVAL '160' HOUR	间隔 160 小时。
INTERVAL 'mm' MINUTE	INTERVAL '14' MINUTE	间隔 14 分钟。
INTERVAL 'ss' SECOND(fractional_seconds_precision)	INTERVAL '14.666' SECOND(2, 3)	间隔 14.666 秒。

如下代码所示，在表 Interval_Sample 中创建了数据类型为 INTERVAL DAY TO SECOND 的两列 interval1、interval2 并向这两列中插入数值：

```
CREATE TABLE Interval_Sample (interval1 INTERVAL DAY TO SECOND, interval2 INTERVAL DAY(3) TO SECOND (3));
INSERT INTO Interval_Sample (interval1, interval2) VALUES ( INTERVAL '15 06:10:08' DAY TO SECOND, INTERVAL '150 06:10:08' DAY(3) TO SECOND(3));
SELECT * FROM Interval_Sample;
```

返回结果：

```
+--------------------+--------------------+
| interval1          | interval2          |
+--------------------+--------------------+
| +15 06:10:08.000000 | +150 06:10:08.000 |
+--------------------+--------------------+
```

间隔与其他日期类型的计算

OceanBase 支持数据类型间的转换，所以间隔数据类型的值可以与其他日期值进行数学运算。但是数据库并不支持数据类型间任意的进行加、减、乘、除运算。请参阅文档 日期时间 DATE 和间隔 INTERVAL 的计算 中查看目前支持的日期类型计算矩阵图和参阅文档 数据类型转换 了解数更多数据类型转换的信息。

关于间隔与其他数据类型计算的示例，请参阅文档 INTERVAL YEAR TO MONTH 数据类型 。

6.4.1.4.8. 日期时间 DATE 和间隔 INTERVAL 的计算

您可以在日期（ DATE ）、时间戳（ TIMESTAMP 、 TIMESTAMP WITH TIME ZONE 和 TIMESTAMP WITH LOCAL TIME ZONE ）和间隔（ INTERVAL YEAR TO MONTH 和 INTERVAL DAY TO SECOND ）上执行许多算术运算。

OceanBase 根据以下规则计算结果：

- 您可以在日期和时间戳值（而非间隔值）的算术运算中使用 NUMBER 常数。OceanBase 在内部将时间戳值转换为日期值，并将算术日期时间和间隔表达式中的 NUMBER 常量解释为天数。例如， SYSDATE +1 是明天。 SYSDATE -7 是一周前。 SYSDATE +（10/1440）是十分钟之后。

  > 注意
  >
  > 您不能乘以或除以日期或时间戳值。

- OceanBase 将 BINARY_FLOAT 和 BINARY_DOUBLE 操作数转换为 NUMBER 。
- 每个 DATE 值都包含一个时间成分，许多日期运算的结果都包含一个分数。此分数表示一天的一部分。例如，1.5 天是 36 个小时。OceanBase 内置函数还返回这些分数，以对 DATE 数据执行常见操作。例如， MONTHS_BETWEEN 函数返回两个日期之间的月数。结果的小数部分代表一个 31 天月份的那部分。
- 如果一个操作数是 DATE 值或数字值，但都不包含时区或小数秒部分，则：
  - OceanBase 将其他操作数转换为 DATE 数据。数值乘以间隔例外，这将返回一个间隔。
  - 如果另一个操作数具有时区值，则 OceanBase 在返回的值中使用当前会话时区。
  - 如果另一个操作数具有小数秒数值，则小数秒数值将丢失。
- 当您将时间戳，时间间隔或数字值传递给仅限于 DATE 数据类型使用的内置函数时，OceanBase 将把非 DATE 值转换为 DATE 值。
- 当间隔计算返回日期时间值时，结果必须是实际的日期时间值，否则数据库将返回错误。例如，以下两个语句返回错误：

```
SELECT TO_DATE("31-AUG-2004','DD-MON-YYYY") + TO_YMINTERVAL("0-1") FROM DUAL;
SELECT TO_DATE("29-FEB-2004','DD-MON-YYYY") + TO_YMINTERVAL("1-0") FROM DUAL;
```

第一条语句失败，因为在一个有31天的月份上增加一个月，导致计算结果为 9 月 31 日，这不是有效的日期。第二条语句失败，因为在仅每四年存在的日期上增加一年是无效的。但是，在 2 月 29 日之前加上四年是有效的，计算结果为2008年2月29日：

```
SELECT TO_DATE("29-FEB-2004", "DD-MON-YYYY") + TO_YMINTERVAL("4-0") FROM DUAL;
```

- 下表是日期时间算术运算的矩阵。“—”表示不支持的操作。

数据类型	运算符	日期（DATE）	时间戳（TIMESTAMP）	间隔（INTERVAL）	数字（NUMERIC）
日期（DATE）	+	—	—	日期	日期
	-	数字	间隔	日期	日期
	*	—	—	—	—
	/	—	—	—	—
时间戳（TIMESTAMP）	+	—	—	时间戳	日期
	-	间隔	间隔	时间戳	日期
	*	—	—	—	—
	/	—	—	—	—
间隔（INTERVAL）	+	日期	时间戳	间隔	—
	-	—	—	间隔	—
	*	—	—	—	间隔
	/	—	—	—	间隔
数字（NUMERIC）	+	日期	日期	—	NA
	-	—	—	—	NA
	*	—	—	间隔	NA
	/	—	—	—	NA

6.4.1.5. RAW 数据类型

RAW 是一种可变长度的数据类型，在不同平台上传输时，传送的都是二进制信息，即使字符集不同也不需要转换。OceanBase 中用于保存二进制数据或字节字符串。

RAW 类似于 VARCHAR2，声明方式是 RAW(length)，length 为长度，以字节为单位，作为数据库列最大存储 2000 个字节，作为变量最大存储 2000 字节。

语法

```
RAW(length)
```

参数

参数	说明
length	表示长度，以字节为单位，作为数据库列最大存储 2000 字节的数据，作为变量最大存储 2000 字节的数据。

示例

- **示例 1：** 在 test_raw 表中声明 RAW 型数据，并向表中插入一条数据。

```
CREATE TABLE test_raw (c1 RAW(10));
INSERT INTO test_raw VALUES (utl_raw.cast_to_raw('1234567890'));
```

执行以下语句：

```
SELECT utl_raw.cast_to_varchar2(c1) FROM test_raw;
```

查询结果如下：

```
+------------------------------+
| UTL_RAW.CAST_TO_RAW(C1) |
+------------------------------+
|    1234567890      |
+------------------------------+
```

- **示例 2：** 向 raw_test 表中插入 2 条数据。

```
CREATE TABLE raw_test (id number, raw_date raw(10));
INSERT INTO raw_test VALUES (1, hextoraw('ff'));
INSERT INTO raw_test VALUES (2, utl_raw.cast_to_raw('051'));
```

执行以下语句：

```
SELECT * FROM raw_test;
```

查询结果如下：

```
+-----------+----------+
| ID  |RAW_DATE|
+-----------+----------+
|  1  |  ff |
+-----------+----------+
|  2  | 303531|
+-----------+----------+
```

SQL 函数 `HEXTORAW()` 会把字符串中数据转换为 16 进制数。SQL 函数 `UTL_RAW.CAST_TO_RAW([VARCHAR2])` 会把字符串中每个字符的 ASCII 码存放到 `RAW` 类型的字段中。例如，**051** 转换为 **303531**。

- **示例 3：** OceanBase 将 `RAW` 数据转换为字符数据，每个字符代表 `RAW` 数据的四个连续位的十六进制数（0~9 和 A~F 或 a~f）。

如：二进制位数 11001011，其 `RAW` 数据转换后为字符 **CB**。

OceanBase 将字符数据转换为 `RAW`，它是把每个连续的输入字符解释为二进制数据的四个连续位，通过把这些位进行级联，来构建结果 `RAW` 值。

> **注意**
>
> 如果任一输入字符不是十六进制数（0~9 和 A~F 或 a~f），则将报告错误。如果字符数为奇数，则结果不确定。

更多信息

字符转换

在数据库之间传输数据，或者在数据之间传输数据库字符集和客户端字符集时，OceanBase 会自动在不同数据库字符集之间转换 `CHAR` 和 `VARCHAR2` 数据。而传输 `RAW` 数据时 OceanBase 不执行字符转换。

RAW 的函数

函数	说明
HEXTORAW()	当使用 `hextoraw` 时，会把字符串中数据转换成 16 进制串，字符串中的每两个字符表示了结果 `RAW` 中的一个字节。
RAWTOHEX(rawvalue)	将 `RAW` 类数值 `rawvalue` 转换为一个相应的十六进制表示的字符串。 `rawvalue` 中的每个字节都被转换为一个双字节的字符串。
UTL_RAW_CAST_TO_RAW([VARCHAR2])	保持数据的存储内容不变，仅改变数据类型，将 `VARCHAR2` 转换为 `RAW` 类型。
UTL_RAW_CAST_TO_VARCHAR2([RAW])	保持数据的存储内容不变，仅改变数据类型，将 `RAW` 类型转换为 `VARCHAR2` 类型。
UTL_RAW.BIT_OR()、UTL_RAW.BIT_AND()、UTL_RAW.BIT_XOR()	位操作。

6.4.1.6. 大对象数据类型

6.4.1.6.1. 大对象数据类型概述

大对象数据类型（Large Object），简称 LOB，用来存储大型和非结构化数据，例如文本、图像、视频和空间数据等。包括 `BLOB` 和 `CLOB` 。

您在创建表时，可以选择为 LOB 列或 LOB 对象属性指定与表中指定的不同的表空间和存储特征。

OceanBase 支持以下大对象数据类型：

类型	长度	定义长度上限（字符）	字符集
BLOB	变长	48M	BINARY

类型	长度	定义长度上限（字符）	字符集
CLOB	变长	48M	UTF8MB4

6.4.1.6.2. BLOB 数据类型

BLOB 全称为二进制大型对象（Binary Large Object）。它用于存储数据库中的大型二进制对象，可以将 BLOB 对象视为没有字符集语义的位流。 BLOB 存储的二进制数据，其字节的长度上限为 48M，字符集是 BINARY 。

BLOB 对象具有完整的事务支持。通过 SQL 、 DBMS_LOB 软件包进行的更改将完全参与事务。可以提交和回滚 BLOB 值操作。但是，您不能在一个事务中将 BLOB 定位器保存在 PL/SQL 中，然后在另一事务或会话中使用它。

> 注意
>
> 存储的二进制文件过大，会使数据库的性能下降。

在数据库中，通常像图片、文件、音乐等大文件信息就用 BLOB 字段来存储，它先将大文件转为二进制再存储进去。

如下，创建表 blob_table，并设置 blob_cl 列为 BLOB 数据类型。

```
CREATE TABLE blob_table (blob_cl BLOB);
```

6.4.1.6.3. CLOB 数据类型

CLOB 全称为字符大型对象（Character Large Object）。它用于存储单字节和多字节字符数据。支持固定宽度和可变宽度字符集，且都使用数据库字符集。 CLOB 不支持宽度不等的字符集。可存储字节的长度上限（字符）是 48 M，字符集是 UTF8MB4 。

CLOB 对象具有完整的事务支持。通过 SQL 、 DBMS_LOB 软件包进行的更改将完全参与事务。可以提交和回滚 CLOB 值操作。但是，您不能在一个事务中将 CLOB 定位器保存在 PL/SQL，然后在另一事务或会话中使用它。

由于 VARCHAR2 类型字段长度最大 32767，若需要保存的字段长度大于 32767，可以使用 CLOB 类型。另外，可以使用 CLOB 来保存 CHAR 数据，如 XML 文档就是用 CLOB 数据保存内容。

例如，创建表 temp，设置 temp_clob 列为 CLOB 数据类型。

```
CREATE TABLE temp (temp_clob CLOB);
```

6.4.2. 数据类型比较规则

6.4.2.1. 数据类型比较规则概述

数据类型比较规则规定了 OceanBase 数据库如何比较每种数据类型的值。

OceanBase 数据类型比较规则支持以下数据值：

- 数值
- 日期值
- 字符值

更多信息

- 数据类型优先级
- 数据类型转换
- 数据转换的安全注意事项

6.4.2.2. 数值

数值数据有定点数、浮点数和零。

数值数据比较规则如下：

- 较大的值大于较小的值，如：5.5 > 2.1。
- 负数均小于零，如 -3 < 0, -200 < -1。
- 正数均大于零，如 20 > 0, 100 > 1。

6.4.2.3. 日期值

`DATE` 数据类型存储日期和时间信息。每个 `DATE` 值，OceanBase 存储以下信息：年、月、日、小时、分钟和秒，但是并不包含时区信息。

日期数据比较规则如下：

- 现在的时间大于过去的时间，如 2018 年 5 月 1 日的日期值大于 2012 年 5 月 1 日的日期值。
- 下午的时间大于早上的时间，如 2019 年 2 月 2 日下午 3:30 的日期时间值大于 2019 年 2 月 2 日 上午 10:30 的日期时间值。
- 早上的时间大于昨天的时间，如 2019 年 3 月 5 日上午 2:30 的日期时间值大于 2019 年 3 月 4 日下午 23:30 的日期时间值。

6.4.2.4. 字符值

字符数据是根据字符值大小比较的，而字符值则根据以下两种度量进行比较：

- 二进制和语言比较
- 空白填充或非填充比较语义

二进制和语言比较

二进制比较

在默认的二进制比较中，OceanBase 根据数据库字符集中字符的数字代码的级联值比较字符串。如果一个字符在字符集中的数值大于另一个，则该字符更大。OceanBase 不支持 ASCII 字符集和 EBCDIC 字符集。

语言比较

在语言排序中，SQL 排序和比较都按照 `NLS_SORT` 指定的语言规则。若字符编码的二进制序列与字符集要求的语言序列不匹配，则使用语言比较。若 `NLS_SORT` 参数的设置不是 `BINARY`，且 `NLS_COMP` 参数设置为 `LINGUISTIC`，则使用语言比较。

空白填充和非填充比较语义

空白填充比较语义

使用空白填充语义，若两个值的长度不同，则 OceanBase 首先将空格添加到较短的空格的末尾，以便它们的长度相等。然后，OceanBase 逐个字符地比较值，直到第一个不同的字符为止。在第一个不同位置具有较大字符的值被认为较大。如果两个值没有不同的字符，则认为它们相等。此规则意味着两个值仅在尾随空白数上不同时相等。

> **注意**
>
> 只有当比较中的两个值都是数据类型 `CHAR`、`NCHAR`、文本文字或 `USER` 函数返回的值时，OceanBase 就会使用空白填充的比较语义。

非填充比较语义

使用非填充语义，OceanBase 逐个字符地比较两个值，直到第一个不同的字符为止。该位置上具有较大字符的值被认为较大。如果两个不同长度的值在较短的值之前一直相同，则较长的值被认为较大。如果两个长度相等的值没有不同的字符，则认为这些值相等。

> **注意**
>
> 只要比较中的一个或两个值的数据类型为 `VARCHAR2` 或 `NVARCHAR2`，OceanBase 就会使用非填充比较语义。

示例

使用不同的比较语义比较两个字符值的结果不同。本示例显示了使用空白填充语义和非填充语义比较。

空白填充	非填充
'ac' > 'ab'	'ac' > 'ab'
'ab' > 'a '	'ab' > 'a '
'ab' > 'a'	'ab' > 'a'

空白填充	非填充
'ab' = 'ab'	'ab' = 'ab'
'ac' > 'ab'	'ac' > 'ab'
'a ' = 'a'	'a ' > 'a'

通常，空白填充和非填充比较的结果是相同的。而最后一行的比较示例说明了空白填充和非填充比较语义之间的区别。

6.4.2.5. 数据类型优先级

OceanBase 使用数据类型优先级来确定隐式数据类型转换顺序。

OceanBase 数据类型的转换优先级如下（由高到低）：

1. 日期时间和间隔数据类型
2. `BINARY_DOUBLE` 数据类型
3. `BINARY_FLOAT` 数据类型
4. `NUMBER` 数据类型
5. 字符数据类型
6. 所有其他内置数据类型

6.4.2.6. 数据类型转换

通常，表达式不能包含不同数据类型的值。但是为了使表达式能够进行计算，OceanBase 支持从一个数据类型到另一个数据类型的值的隐式转换和显式转换。

隐式数据类型转换

当转换有意义时，OceanBase 数据库会自动将一个值从一种数据类型转换为另一种数据类型。隐式数据类型的转换规则：

- `INSERT` 和 `UPDATE` 操作时，OceanBase 把变量值转换成列类型。
- `SELECT FROM` 操作时，OceanBase 把列数据类型转换成目标变量类型。
- 字符值和数字值比较时，OceanBase 把字符值转换成数字值。
- 在处理数值时，OceanBase 会调整精度和小数位数。由此产生的数字数据类型与基础表中找到的数字数据类型不同。
- 字符值或数值和浮点数值之间的转换可以是不精确的，因为字符类型和数量使用十进制精度来表示数值，浮点数使用二进制精度。
- 当一个 `CLOB` 值转换为一个字符类型如 `VARCHAR2`，或 `BLOB` 转换为 `RAW` 时。如果要转换的数据大于目标数据类型，那么数据库会返回一个错误。

- 在从时间戳值转换为 `DATE` 值的过程中，时间戳值的小数秒部分被截断，且时间戳值的小数秒部分进行四舍五入。
- 从 `BINARY_FLOAT` 转换为 `BINARY_DOUBLE` 是准确的。
- 如果 `BINARY_DOUBLE` 的精度位数超出了 `BINARY_FLOAT` 支持的位数， `BINARY_DOUBLE` 转换为 `BINARY_FLOAT` 是不精确的。
- 当将字符值与 `DATE` 值进行比较时，OceanBase 将字符数据转换为 `DATE` 。
- 赋值操作时，OceanBase 把等号右边的值转换成左边赋值目标数据类型。
- 连接操作时，OceanBase 把非字符类型转换成字符类型或国家字符类型。

隐式数据类型转换矩阵

下表显示了所有的隐式数据类型转换，您不需要考虑转换的方向或转换的上下文。“-”表示不支持转换。

数据类型	CHAR	VARCHAR2	NCHAR	NVARCHAR2	DATE	DATETIME / INTERVAL	NUMBER	BINARY_FLOAT	BINARY_DOUBLE	RAW	CLOB	BLOB
CHAR	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
VARCHAR2	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-
NCHAR	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-
NVARCHAR2	Yes	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-
DATE	Yes	Yes	Yes	Yes	-	-	-	-	-	-	-	-
DATETIME / INTERVAL	Yes	Yes	Yes	Yes	-	-	-	-	-	-	-	-

NUMBER	Yes	Yes	Yes	Yes	-	-	-	Yes	Yes	-	-	-
BINARY_FLOAT	Yes	Yes	Yes	Yes	-	-	Yes	-	Yes	-	-	-
BINARY_DOUBLE	Yes	Yes	Yes	Yes	-	-	Yes	Yes	-	-	-	-
RAW	Yes	Yes	Yes	Yes	-	Yes [1]	-	-	-	-	Yes	-
CLOB	Yes	Yes	Yes	Yes	-	-	-	-	-	-	-	Yes
BLOB	-	-	-	-	-	-	-	-	-	Yes	-	-

[1] 您不能直接将 `RAW` 转换为 `INTERVAL`，但是可以使用 `UTL_RAW.CAST_TO_VARCHAR2([RAW])` 将 `RAW` 转换为 `VARCHAR2`，然后将所得的 `VARCHAR2` 值转换为 `INTERVAL`。

不同字符类型之间隐式转换的方向

数据类型	TO_CHAR	TO_VARCHAR2	TO_NCHAR	TO_NVARCHAR2
from CHAR	-	VARCHAR2	NCHAR	NVARCHAR2
from VARCHAR2	VARCHAR2	-	NVARCHAR2	NVARCHAR2
from NCHAR	NCHAR	NCHAR	-	NCHAR2
from NVARCHAR2	NVARCHAR2	NVARCHAR2	NVARCHAR2	-

隐式数据类型转换示例

执行以下语句：

```
SELECT 5 * 10 + 'james' FROM DUAL;
```

语句执行失败，且您收到以下报错：

```
invalid number
```

这是由于 OceanBase 使用了隐式数据类型转换，将 `'james'` 转换为数字类型，但是转换失败。

本示例将字符串 '2' 从 `CHAR` 数据类型隐式转换为了数字数据类型 2，计算结果为 52。

执行以下语句：

```
SELECT 5 * 10 + '2' FROM  DUAL;
```

查询结果如下：

```
+---------------------+
|  5 * 10 + '2'  |
+---------------------+
|     52      |
+---------------------+
```

显式数据类型转换

您可以使用 SQL 转换函数转换数据类型，SQL 函数显式转换一个数据类型为另一个数据类型。

显示类型转换矩阵

数据类型	To CHAR、VARCHAR2、NCHAR、NVARCHAR2	To NUMBER	To Datetime/ Interval	To RAW	To CLOB、BLOB	To_BINARY_FLOAT	To_BINARY_DOUBLE
From CHAR、VARCHAR2、NCHAR、NVARCHAR2	TO_CHAR(char .) 、TO_NCHAR(char .)	TO_NUMBER	TO_DATE 、TO_TIMESTAMP、TO_TIMESTAMP_TZ 、TO_YMINTERVAL、TO_DSINTERVAL	HEXTORAW	TO_CLOB	TO_BINARY_FLOAT	TO_BINARY_DOUBLE

From NUMBER	TO_CHAR(number)、TO_NCHAR(number)	--	TO_DATE、NUMTOYM_INTERVAL、NUMTOOLS_INTERVAL	--	--	TO_BINARY_FLOAT	TO_BINARY_DOUBLE
From Datetime/Interval	TO_CHAR(date)、TO_NCHAR(date)	--	--	--	--	--	--
From RAW	RAWTOHEX、RAWTONHEX	--	--	--	TO_BLOB	--	--
From CLOB、BLOB	TO_CHAR、TO_NCHAR	--	--	--	TO_CLOB	--	--
From BINARY_FLOAT	TO_CHAR(char.)、TO_NCHAR(char.)	TO_NUMBER	--	--	--	TO_BINARY_FLOAT	TO_BINARY_DOUBLE
From BINARY_DOUBLE	TO_CHAR(char.)、TO_NCHAR(char.)	TO_NUMBER	--	--	--	TO_BINARY_FLOAT	TO_BINARY_DOUBLE

显式数据类型转换示例

当前的时间通过 `TO_CHAR` 函数显式转换为想要的格式输出。

执行以下语句：

```
SELECT TO_CHAR(SYSDATE, 'YYYY_MM_DD') FROM DUAL;
```

查询结果如下：

```
+------------------------------+
| TO_CHAR(SYSDATE,'YYYY_MM_DD') |
+------------------------------+
| 2020_02_27                   |
+------------------------------+
```

6.4.2.7. 数据转换的安全注意事项

通过隐式转换或不指定格式模型的显式转换将日期时间值转换为文本时，格式模型由一个全局会话参数定义。根据源数据类型，这些参数名称为 `NLS_DATE_FORMAT` 、 `NLS_TIMESTAMP_FORMAT` 或 `NLS_TIMESTAMP_TZ_ FORMAT` 。这些参数的值可以在客户端环境或 `ALTER SESSION` 语句中指定。

当不指定格式模型的显式转换对动态构造的 SQL 语句中的日期时间值进行显式转换时，格式模型对会话参数的选择过程会对数据库安全性产生负面影响。

动态构造的 SQL 语句是指由程序或者存储过程生成的 SQL 语句。执行动态构造的 SQL 语句，需要 OceanBase 内置的 PL 包 `DBMS_SQL` 或与 PL 语句 `EXECUTE IMMEDIATE` 相关，但这些并不是唯一执行动态构造的 SQL 文本的方式。

如下所示， `start_date` 的数据类型为 `DATE` ，使用会话参数 `NLS_DATE_FORMAT` 中指定的格式模型将 `start_date` 的值转换为文本，再将结果传递到 SQL 文本中。日期时间格式模型可以简单地由双引号所包含的文本组成。

```
SELECT last_name FROM employees WHERE hire_date > '' || start_date || '';
```

> **说明**
>
> 为显式转换的格式模型设置全球化参数的用户可以决定上述转换产生了什么文本。
>
> 若 SQL 语句由过程执行，则该过程的执行可能会由于session变量的修改而被SQL注入。当某些过程具有更高权限（例如Definer's Rights Procedure）时，造成的安全方面的影响可能更大。

6.4.3. 字面量

6.4.3.1. 字面量概述

字面量（Literals）是用于表达一个固定值的表示法。许多函数和 SQL 语句都需要指定字面量，字面量也可以作为表达式和条件的一部分。OceanBase 支持以下字面量：

- 文本字面量
- 数值字面量
- 日期时间字面量
- 间隔字面量

6.4.3.2. 文本字面量

文本字面量（Text Literals）是使用单引号 `''` 引起来的字符串，用来在表达式、条件、SQL 函数、SQL 语句中指定字符串的值。

文本字面量本身具有 `CHAR` 和 `VARCHAR2` 数据类型的属性：

- 在表达式和条件中，OceanBase 通过使用空白填充的比较语义进行比较，将文本字面量视为数据类型为 `CHAR` 。
- 指定文本字面量时， `CHAR` 数据类型的值的长度最大是 2000， `VARCHAR2` 数据类型的值的长度最大是 32767。

以下是一些有效的文本字面量，若要在字符串中表现一个单引号，需要在字符串中的单引号前再插入一个单引号：

```
'Jackie''s raincoat'
'Hello'
'09-MAR-98'
'今天天气很好'
```

6.4.3.3. 数值字面量

数值字面量（Numberic Literals）使用数值字面量指定固定数和浮点数的值。

整数字面量

当表达式、条件、SQL 函数和 SQL 语句中出现整数时，需要用整数（Integer）字面量来指定值。

下面是一些有效的整数字面量：

```
8
+186
-15
```

数字和浮点数字面量

当表达式、条件、SQL 函数和 SQL 语句中出现数字时，需要用数字（Number）或浮点字（Floating-Point）面量来指定值。

以下是一些有效的数字 NUMBER 字面量：

```
12
+6.87
0.5
25e-03
-9
```

以下是一些有效的浮点数 Floating-Point 字面量：

```
25f
+6.34F
0.5d
-1D
```

数字字面量最大可以储存精度为 38 位的数字。如果字面量要求的精度比 NUMBER 、 BINARY_FLOAT 或 BINARY_DOUBLE 所提供的精度更高，则 OceanBase 将截断该值。如果字面量的范围超出 NUMBER 、 BINARY_FLOAT 或 BINARY_DOUBLE 支持的范围，则 OceanBase 会抛出错误。

示例

数值字面量中的小数点分隔符始终是点（.）。在期望数值字面量的地方指定了文本字面量，则该文本字面量被转换为数值字面量。

在下面的示例中，计算了 2 乘以数值字面量 2.2 和 2 乘以文本字面量 '3.3'：

```
SELECT 2*2.2, 2*'3.3' FROM DUAL;
```

返回结果：

```
+-------+---------+
| 2*2.2 | 2*'3.3' |
+-------+---------+
|  4.4  |   6.6   |
+-------+---------+
```

6.4.3.4. 日期时间字面量

6.4.3.4.1. 日期字面量

日期（DATE）字面量可以通过字符串指定，或者可以使用 TO_DATE 函数将字符或数字值转换为日期值。日期字面量是唯一接受用 TO_DATE 表达式代替字符串指定值的：

```
TO_DATE('2020-03-25 11:05:00', 'YYYY-MM-DD HH24:MI:SS')
```

使用日期值指定日期字面量时，必须使用公历的日期值。同时也可以如下所示，使用 ANSI 来指定日期字面量，ANSI 日期字面量不包含时间信息，而且必须使用 YYYY-MM-DD 的格式：

```
DATE '2020-03-25'
```

此外，还可以使用数据库默认日期值来指定日期字面量，当在日期表达式中使用默认值时，OceanBase 会自动将默认日期格式的字符值转换为日期值。数据库的默认日期值由初始化参数 NLS_DATE_FORMAT 指定，此示例中默认格式为 DD-MON-RR ：

```
TO_DATE('25-FEB-20', 'DD-MON-RR')
```

如果您指定不带时间成分的日期值，则默认时间为午夜（24 小时制 00:00:00 和 12 小时制 12:00:00）。如果指定的日期值不带日期成分，则默认日期为当前月份的第一天。

OceanBase 表中 `DATE` 列始终同时包含日期和时间字段。因此，如果查询 `DATE` 列，则必须在查询中指定时间字段，或确保 `DATE` 列中的时间字段设置为午夜。否则，数据库可能不会返回您期望的查询结果。比如创建一张具有 id 列和 datecol 日期列的表 Date_Literals：

```
CREATE TABLE Date_Literals (id NUMBER, datecol DATE);
```

在表中插入当前会话的系统日期时间 SYSDATE，此示例使用了 `TRUNC` 函数将时间字段设置为午夜， `TRUNC` 函数会截取 SYSDATE 的日期部分，这样 datecol 列中的时间会自动填充默认的午夜时间：

```
INSERT INTO Date_Literals VALUES (1,SYSDATE);
INSERT INTO Date_Literals VALUES (2,TRUNC(SYSDATE));
```

此时表中数据为：

```
+------+--------------------+
| id  | datecol        |
+------+--------------------+
|  1 | 25-FEB-20 11:28:16 |
|  2 | 25-FEB-20 00:00:00 |
+------+--------------------+
```

当查询中不包含时间信息时，可以在查询中使用大于或小于条件，而不是等于或不等于条件：

```
SELECT * FROM Date_Literals WHERE datecol > TO_DATE('2020-02-24', 'YYYY-MM-DD');
```

返回结果：

```
+------+--------------------+
| id  | datecol        |
+------+--------------------+
|  1 | 25-FEB-20 11:28:16 |
|  2 | 25-FEB-20 00:00:00 |
+------+--------------------+
```

当使用等于条件时，由于查询中不包含时间信息，所以结果只返回了时间信息为午夜值的日期：

```
SELECT * FROM Date_Literals WHERE datecol = TO_DATE('2020-02-25', 'YYYY-MM-DD');
```

返回结果：

```
+------+---------------------+
| id  | datecol        |
+------+---------------------+
|  2 | 25-FEB-20 00:00:00 |
+------+---------------------+
```

反过来，可以过滤掉 datecol 列中的时间字段，只查询日期字段：

```
SELECT * FROM Date_Literals WHERE TRUNC(datecol) = DATE '2020-02-25';
```

返回结果：

```
+------+---------------------+
| id  | detacol        |
+------+---------------------+
|  1 | 25-FEB-20 11:28:16 |
|  2 | 25-FEB-20 00:00:00 |
+------+---------------------+
```

6.4.3.4.2. 时间戳字面量

OceanBase 支持以下三种时间戳字面量：

- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE

TIMESTAMP 字面量

`TIMESTAMP[(scale)]` 数据类型存储了年、月、日、时、分、秒和小数秒值的值。当指定 TIMESTAMP 字面量时，秒字段最大可以指定精度到第9位的纳秒：

```
TIMESTAMP '2020-02-25 11:26:18.316'
```

TIMESTAMP WITH TIME ZONE 字面量

TIMESTAMP WITH TIME ZONE 字面量是包含时区信息的时间戳字面量。

`TIMESTAMP [(scale)] WITH TIME ZONE` 数据类型是 `TIMESTAMP[(scale)]` 数据类型的一种变体，它在 `TIMESTAMP[(scale)]` 数据类型的基础上还存储时区偏移量或者时区区域名称等信息。在指定 TIMESTAMP WITH TIME ZONE 字面量时，需要指定时区信息并且秒字段最大可以指定精度到第9位的纳秒，以下示例使用时区偏移量指定了时区字段的值：

```
TIMESTAMP '2020-02-25 11:26:18.316 +08:00'
```

当两个 TIMESTAMP WITH TIME ZONE 字面量中的值表示 GMT 时区的同一时刻，那么尽管它们的时区字段的值不同，它们也被视为相同的字面量。如以下示例所示，GMT-8 时区的早上 8 点和 GMT-5 时区的早上11点其实是同一时刻：

```
TIMESTAMP '2020-04-25 08:26:18.316 -08:00'
TIMESTAMP '2020-04-25 11:26:18.316 -05:00'
```

同样的，在字面量中我们可以使用时区区域名称替换时区偏移量，如下所示的示例中将 **-08:00** 替换为了 **America/Los_Angeles**：

```
TIMESTAMP '2020-02-01 11:00:00 America/Los_Angeles'
```

由于一些地区有夏令时的转换，为了消除夏令时转换时时间的歧义，在指点字面量的值时可以同时使用 时区区域名称（TZR）和相应的缩写 （TZD）以确保字面量的值为夏令时：

```
TIMESTAMP '2020-06-01 11:00:00 America/Los_Angeles PDT'
```

TIMESTAMP WITH LOCAL TIME ZONE 字面量

`TIMESTAMP [(scale)] WITH LOCAL TIME ZONE` 数据类型是包含本地时区信息的数据类型。OceanBase 中并没有专门的 TIMESTAMP WITH LOCAL TIME ZONE 字面量，是通过其他有效的日期时间字面量来为 `TIMESTAMP [(scale)] WITH LOCAL TIME ZONE` 数据类型赋值的。下表显示了一些可用于将值插入 TIMESTAMP WITH LOCAL TIME ZONE 列的格式，以及查询返回的相应值：

INSERT 语句中指定的值	查询返回的值
'25-FEB-20'	25-FEB-20 00.00.000000
SYSTIMESTAMP	25-FEB-20 14:28:41.264258
TO_TIMESTAMP('25-FEB-2020', 'DD-MON-YYYY')	25-FEB-20 00.00.000000
SYSDATE	25-FEB-20 02.55.29.000000 PM
TO_DATE('25-FEB-20', 'DD-MON-YYYY')	25-FEB-20 12.00.00.000000 AM
TIMESTAMP'2020-02-25 8:00:00 America/Los_Angeles'	25-FEB-20 08.00.00.000000 AM

6.4.3.5. 间隔字面量

间隔字面量用来指定一段时间的值。OceanBase 支持两种类型的间隔字面量：

- INTERVAL YEAR TO MONTH
- INTERVA DAY TO SECOND

前导字段和尾随字段

每种间隔字面量都包含一个前导字段和一个可选的尾随字段。前导字段定义了要测量的日期或时间的基本单位，尾随字段定义了所考虑的基本单位的最小增量。例如，DAY TO MINUTE 用来指定最小单位到月份的间隔字面量，其中前导字段是 `YEAR`，尾随字段是 `MINUTE`。尾随字段是可选的，在指定间隔字面量时可以省去。

在间隔字面量中有以下字段：`YEAR`、`MONTH`、`DAY`、`HOUR`、`MINUTE` 和 `SECOND`。它们的权重从 `YEAR` 开始按顺序递减。当需要指定尾随字段时，字面量中尾随字段的权重一定要低于前导字段，否则是无效的指定。例如，`INTERVAL '1-2' DAY TO YEAR` 是个无效的字面量。

前导字段值的位数范围为 0~9，默认值为 2。`SECOND` 字段指定了秒数，该字段最大可以精确到小数点后 9 位，最小是小数点 0 位，默认值精度是小数点 6 位。字段的值超出指定的范围后，数据库会返回错误。`SECOND` 字段的小数位数如果超出指定精度，会四舍五入到符合指定精度的值。

INTERVAL YEAR TO MONTH 字面量

`INTERVAL YEAR TO MONTH` 字面量用来指定以年和月为单位的一段时间。

以下是一些 `INTERVAL YEAR TO MONTH` 字面量的示例：

示例	说明
INTERVAL '265-2' YEAR(3) TO MONTH	间隔 265 年 2 个月。前导字段 `YEAR` 的精度大于默认的 2 位，需要指定与值位数相符的精度值
INTERVAL '265' YEAR(3)	表示间隔 265 年。
INTERVAL '500' MONTH(3)	表示间隔 500 个月或 41 年 8 个月。
INTERVAL '10' MONTH	表示隔 10 个月。
INTERVAL '123' YEAR	返回错误，值 123 超出了默认精度 2 位。

可以在一个 `INTERVAL YEAR TO MONTH` 字面量之间添加或减去另一个 `INTERVAL YEAR TO MONTH` 字面量。例如：`INTERVAL '6-2' YEAR TO MONTH` + `INTERVAL'21' MONTH` = `INTERVAL '7-11' YEAR TO MONTH`。

INTERVAL DAY TO SECOND 字面量

`INTERVAL DAY TO SECOND` 字面量用来指定以天和具体时间为单位的一段时间。

以下是一些 `INTERVAL DAY TO SECOND` 字面量的示例：

示例	说明
INTERVAL '4 5:12:10.222' DAY TO SECOND(3)	表示间隔 4 天 5 小时 12 分钟 10.222 秒。 `SECOND` 字段小数点默认精度是 6，这里如果不手动指定精度为 3，返回的结果中会用 0 补齐位数。
INTERVAL '4 5:12' DAY TO MINUTE	表示间隔 4 天 5 小时 12 分钟
INTERVAL '400 5' DAY(3) TO HOUR	表示间隔 400 天 5 小时。前导字段 `DAY` 超出默认精度 2 位，这里手动指定精度为 3。
INTERVAL '400' DAY(3)	表示间隔 400 天。
INTERVAL '11:12:10.2222222' HOUR TO SECOND(7)	表示间隔 11 小时 12 分钟 10.2222222 秒。 `SECOND` 字段的值超出默认精度 6 位，这里手动指定了和值相符的精度。
INTERVAL '11:20' HOUR TO MINUTE	表示间隔 11 小时 20 分钟。
INTERVAL '10' HOUR	表示间隔 10 小时
INTERVAL '10:22' MINUTE TO SECOND	表示间隔 10 分钟 22 秒。
INTERVAL '10' MINUTE	表示间隔 10 分钟。
INTERVAL '4' DAY	表示间隔 4 天。
INTERVAL '25' HOUR	表示间隔 25 小时。
INTERVAL '40' MINUTE	表示间隔 40 分钟。
INTERVAL '120' HOUR(3)	表示间隔 120 小时。
INTERVAL '30.12345' SECOND(2,4)	表示间隔 30.1235 秒。秒的小数点位数超出指定精度，所以四舍五入到小数点第 4 位。

可以在一个 INTERVAL DAY TO SECOND 字面量之间添加或减去另一个 INTERVAL DAY TO SECOND 字面量。例如：INTERVAL'20' DAY - INTERVAL'239' HOUR = INTERVAL'10-1' DAY TO SECOND 。

6.4.4. 格式化

6.4.4.1. 格式化概述

格式化指定了存储在数据库中的日期时间数据或数值数据的格式。当您将字符串转换为日期时间或数字时，格式化会告诉 OceanBase 数据库如何转换并存储该字符串。在 SQL 语句中，您可以通过 TO_CHAR 、TO_NUMBER 和 TO_DATE 等函数的参数来指定：

- OceanBase 数据库返回值的格式
- 存储在数据库中值的格式

OceanBase 支持以下几种数据格式化：

数值格式化

数值格式化指定了存储在数据库中的定点数和浮点数的格式。当您需要将 SQL 语句中出现 NUMBER ， BINARY_FLOAT 或 BINARY_ DOUBLE 值转换为 VARCHAR2 数据类型时，您可以使用函数中的数值格式化。数值格式化是由一个或多个数值格式化元素组成，具体信息请查阅 数值格式化。

日期时间格式化

日期时间格式化指定了存储在数据库中日期时间数据的格式。日期时间格式化的总长度不能超过 22 个字符。当您需要将非默认格式的字符值转换为日期时间格式的值时，您可以使用函数中的日期时间格式化。日期时间格式化是由一个或多个日期时间格式化元素组成，具体信息请查阅 日期时间格式化。将字符串值转换为日期值，是有一些转换规则，关于规则请查阅 字符串到日期的转换规则。

RR 日期时间格式化元素类似于 YY 日期时间格式化元素，但它为跨世纪日期值存储提供了额外的灵活性，关于 RR 日式格式化元素，请查阅 RR 日期时间格式化元素。

格式化修饰符

OceanBase 数据库暂不支持格式化修饰符 FX 和 FM 。

6.4.4.2. 数值格式化

数值格式化指定了存储在数据库中的定点数和浮点数的格式。

函数中的数值格式化

以下数值数据类型转换函数使用了数值格式化：

- 当表达式、条件、SQL 函数和 SQL 语句中出现 NUMBER ， BINARY_FLOAT 或 BINARY_ DOUBLE 时，且您需要它们的值转换为 VARCHAR2 数据类型时，需要用 TO_CHAR 函数的参数指定这些数值的格式。
- 当表达式、条件、SQL 函数和 SQL 语句中出现 CHAR 或 VARCHAR2 时，如果您需要将他们的值转换为 NUMBER 数据类型时，需要用 TO_NUMBER 函数的参数（暂不支持 NLS_NUMERIC_CHARACTERS ）指定这些数值的格式。如果您需要将他们的值转换为 BINARY_FLOAT 或 BINARY_DOUBLE 时，需要用 TO_BINARY_FLOAT 和 TO_BINARY_DOUBLE 函数的参数指定数值格式。

数值格式化会将数值四舍五入，并取有效数字位数。如果某个值的小数位数，左边的有效位数比格式中指定的位数高，则用 `#` 代替该值。如果 `NUMBER` 的正值非常大且无法以指定的格式表示，则使用无穷大符号（～）替换该值。如果负 `NUMBER` 值非常小且无法用指定的格式表示，则负无穷大符号将替换值（-～）。

数值格式化的元素

与 Oracle 不同，OceanBase 数值格式化元素仅支持标准的数值格式。下表为 OceanBase 支持的数值格式化元素：

元素	示例	说明
.（小数点）	99.99	返回一个小数，且小数点在指定位置。 **限制条件**：在数字格式化中，您只能指定一个小数点。
0	0999 9990	0999 返回前导零。9990 返回尾随零。
9	9999	返回具有指定位数的值。如果为正，则返回带有有前导空格的数；如果为负，则返回前导负数。前导零返回 0，除了零值，定点数的小数部分返回零。

如果省略格式参数，OB会将数值转为足够长的 `VARCHAR2` 以保留其所有的有效数字。

示例

执行以下语句：

```
SELECT TO_CHAR(0, '99.99') FROM DUAL;
```

查询结果如下：

```
+--------------------+
| TO_CHAR(0,'99.99') |
+--------------------+
|  .00               |
+--------------------+
```

下表显示了对不同数值的 number 按照格式化元素 'fmt' 查询得到的结果。

```
SELECT TO_CHAR(number, 'fmt') FROM DUAL;
```

number	'fmt'	Result
0	99.99	' .00'
+0.1	99.99	' .10'
-0.2	99.99	' -.20'
0	90.99	' 0.00'
+0.1	90.99	' 0.10'
-0.2	90.99	' -0.20'
0	9999	' 0'
1	9999	' 1'
+123.456	999.999	' 123.456'
-123.456	999.999	'-123.456'

当省略'fmt'参数的时候：

```
SELECT TO_CHAR(123.456) FROM DUAL;
```

显示结果如下：

```
+------------------+
| TO_CHAR(123.456) |
+------------------+
| 123.456          |
+------------------+
```

当 `TO_CHAR` 结果超过40字节，或者传入参数为binary_double/binary_float类型时，结果将被转为科学计数法。

例如：

```
SELECT TO_CHAR(12355555555555555555555555555555555555555555555555555) FROM DUAL;
```

显示结果如下：

```
+-----------------------------------------------------------+
| TO_CHAR(12355555555555555555555555555555555555555555555555555) |
+-----------------------------------------------------------+
| 1.23555555555555555555555555555556E+49                    |
+-----------------------------------------------------------+
```

6.4.4.3. 日期时间格式化

日期时间格式化指定了存储在数据库中日期时间数据的格式。日期时间格式化的总长度不能超过 22 个字符。

函数中的日期时间格式化

日期时间格式化出现在下面的数据类型转换中：

- 将非默认格式的字符值转换为日期时间值时，需要 `TO_DATE` 、 `TO_TIMESTAMP` 和 `TO_TIMESTAMP_TZ` 函数的参数指定日期时间的格式。
- 将日期时间值转换为非默认格式的字符值时，需要指定 `TO_CHAR` 函数的参数。

您可以通过以下方式指定日期时间格式。

- 通过会话参数 `NLS_DATE_FORMAT` 、 `NLS_TIMESTAMP_FORMAT` 或 `NLS_TIMESTAMP_TZ_FORMAT` 显式指定。
- 通过会话参数 `NLS_TERRITORY` 隐式指定。
- `ALTER SESSION` 语句更改会话的默认日期时间格式。

日期时间格式化

日期时间格式化由一个或多个日期时间格式化元素组成。OceanBase 支持的格式化元素请查阅 日期时间格式化元素表。

- 在格式化字符串中，相同的格式化元素不能出现两次，表示类似信息的格式化元素不能组合。例如，您不能在一个格式化字符串中同时使用 `SYYYY` 和 `BC` 元素。
- 所有格式化元素都可以在 `TO_CHAR` 、 `TO_DATE` 、 `TO_TIMESTAMP` 和 `TO_TIMESTAMP_TZ` 函数中使用。
- 日期时间格式化元素 `FF` 、 `TZD` 、 `TZH` 、 `TZM` 和 `TZR` 可以出现在时间戳和间隔格式化中，但不能出现在 `DATE` 格式中。
- 许多日期时间格式元素被空白填充或用零填充至指定的长度。

> **注意**
>
> OceanBase 建议您使用 4 位数的年份元素（ `YYYY` ），较短的年份元素会影响查询优化，因为年份只能在运行时确定。

日期时间格式化元素表

元素	日期时间函数是否支持？	说明
- / , 。 ; : “文字”	是	标点和引用的文本会在结果中复制。
AD A.D.	是	表示公元纪年法，带有或不带有点号。
AMA.M.	是	表示上午，带有或不带有点号。
BCB.C.	是	表示公元前的年份，带有或不带有点号。
D	是	星期几（1-7）。
DAY	是	一天的名称。
DD	是	每月的一天（1-31）。
DDD	是	一年中的某天（1-366）。
DL	是	只能打印类似 "Monday, January, 01, 1996" 的固定格式。
DS	是	只能打印类似 "10-10-1996" 的固定格式。
DY	是	日期的缩写，返回星期值。

元素	日期时间函数是否支持？	说明
FF [1..9]	是	小数秒。使用数字 1~9 来指定返回值的小数秒部分的位数。默认为日期时间数据类型指定的精度。在时间戳和间隔格式中有效，但在 `DATE` 格式中无效。
FX	是	需要字符数据和格式模型之间的精确匹配。
HHHH12	是	小时（1-12）。12 小时制
HH24	是	小时（0-23）。24 小时制
YYYY	是	包含4位数字的年份。
MI	是	分钟（0-59）。
MM	是	月（01-12；一月份表示为 01 ）。
MON	是	月份的缩写。
MONTH	是	月份名称。
PMP.M.	是	表示下午，带有或不带有点号。
Q	是	季度（1、2、3、4； 1月-3月是第 1 季度）。
RR	是	RR 匹配两位数的年份。
RRRR	是	年。接受4位或2位输入。
SS	是	秒（0-59）。

元素	日期时间函数是否支持？	说明
SSSSS	是	午夜后的秒（0-86400）。
TZD	是	夏令时信息。TZD 值是带有夏令时信息的缩写时区字符串。在时间戳和间隔格式中有效，但在 `DATE` 格式中无效。
TZH	是	时区小时。在时间戳记和间隔格式中有效，但在 `DATE` 格式中无效。
TZM	是	时区分钟。在时间戳记和间隔格式中有效，但在 `DATE` 格式中无效。
TZR	是	时区区域信息。在时间戳和间隔格式中有效，但在 `DATE` 格式中无效。
X	是	小数点，永远是 '.' 。
Y,YYY	是	带逗号的年。
YYYYSYYYY	是	4位数字的年份。 S 代表用一个负号表示公元前的日期。
YYYYYY	是	年份的后 3、2 或 1 位数字。

说明

日期时间函数指的是 `TO_CHAR` 、 `TO_DATE` 、 `TO_TIMESTAMP` 和 `TO_TIMESTAMP_TZ` 。

注意，以上转化要求输入的字符串日期能够与格式元素相匹配，否则会报错，例如：

```
SELECT TO_DATE( '31 Aug 2020', 'DD MON YYYY' ) FROM DUAL;

+----------------------------------+
| TO_DATE('31AUG2020','DDMONYYYY') |
+----------------------------------+
| 2020-08-31 00:00:00              |
+----------------------------------+
```

当您的格式串漏掉了一些元素时，会得到系统的报错信息：

```
SELECT TO_DATE( '31 Aug 2020', 'DD MON YYY' ) FROM DUAL;

ORA-01830: date format picture ends before converting entire input string
```

日期格式化元素中的大写字母

拼写出来的单词、缩写词或罗马数字中的大写字母在相应的格式元素中也跟着大写。例如，日期格式元素 `DAY` 产生的 `MONDAY` 也大写，`Day` 和 `Monday` 格式一样，`day` 和 `monday` 格式一样。

```
OceanBase(SYS@SYS)>SELECT TO_CHAR(sysdate,'mon') AS nowMonth FROM DUAL;
+----------+
| NOWMONTH |
+----------+
| sep      |
+----------+

OceanBase(SYS@SYS)>SELECT TO_CHAR(sysdate,'MON') AS nowMonth FROM DUAL;
+----------+
| NOWMONTH |
+----------+
| SEP      |
+----------+
```

日期时间格式化中的标点符号和字符字面量

以下字符需要日期格式化，这些字符出现在返回值中的位置与格式化中字符的位置相同：

- 标点符号，例如连字符，斜杠，逗号，句号和冒号。
- 字符字面量，用双引号引起来。

OceanBase 数据库可以灵活的将字符串转换为日期。当您使用 `TO_DATE` 函数时，若输入字符串中的每个数字元素都包含格式化允许的最大位数，则格式字符串将与输入的字符串匹配。

- **示例 1**：格式元素 **MM/YY**，其中 **02** 对应 **MM**，**07** 对应 **YY**。

执行以下语句：

```
SELECT TO_CHAR(TO_DATE('0207','MM/YY'),'MM/YY') FROM DUAL;
```

查询结果如下：

```
+----------------------------------------+
| TO_CHAR(TO_DATE('0207','MM/YY'),'MM/YY') |
+----------------------------------------+
| 02/07                                  |
+----------------------------------------+
```

- **示例 2**：OceanBase 允许非字母数字字符与格式化中的标点字符匹配，**#** 对应 **/**。

执行以下语句：

```
SELECT TO_CHAR (TO_DATE('02#07','MM/YY'), 'MM/YY') FROM DUAL;
```

查询结果如下：

```
+----------------------------------------+
| TO_CHAR(TO_DATE('02#07','MM/YY'),'MM/YY') |
+----------------------------------------+
| 02/07                                  |
+----------------------------------------+
```

日期格式化元素与全球化支持

在 OceanBase 数据库中，日期时间格式化元素的语言可以通过参数 `NLS_DATE_LANGUAGE` 和 `NLS_LANGUAGE` 指定。默认值是 `AMERICAN`，不支持修改，所以不支持全球化。

示例：日期时间格式化的语言参数默认是 `American`，不支持其他语言。

```
SELECT TO_CHAR (SYSDATE, 'DD/MON/YYYY', 'nls_date_language=''Traditional Chinese'' ') FROM DUAL;
```

查询结果报错，语言参数不支持。

```
ERROR-12702: invalid NLS parameter string used in SQL function
```

更多信息

- 字符串到日期的转换规则

6.4.4.4. RR 日期时间格式化元素

RR 日期时间格式化元素类似于 YY 日期时间格式化元素，但它为跨世纪日期值存储提供了额外的灵活性。在 YY 日期时间格式化元素里，您需要指定年份的全部数字。而在 RR 日期时间格式化元素里，您只需指定年份数字的最后两位数，便可以存储日期值。

RR 日期时间格式化元素与 TO_DATE 函数一起使用，返回值的世纪根据指定的两位数字年份和当前年份的最后两位数字而变化。如果 YY 日期时间格式化元素与 TO_DATE 函数一起使用，则返回的年份始终与当前年份具有相同的世纪。

如果指定的两位数字年份是 00~49，当前年份的最后两位数字是 00~49，则返回的年份与当前年份具有相同的世纪。当前年份的后两位数字是 50~99，则返回年份的前两位数字为下一个世纪。

如果指定的两位数字年份是 50~99，当前年份的后两位数字为 00 到 49，则返回年份的前两位数字为当前年份 的上一个世纪。当前年份的最后两位数字是 50~99，则返回的年份与当前年份处于同一个世纪。

如下所示， RR 日期时间格式化元素根据前两位数字不同的年份返回相同的值。假设这些查询是在 1950~1999 年期间发出的，执行以下语句：

```
SELECT TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-RR'), 'YYYY') "Year1" ,
TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR'), 'YYYY') "Year2"  FROM DUAL;
```

查询结果如下：

```
+-------+-------+
| Year1 | Year2 |
+-------+-------+
| 2017  | 1998  |
+-------+-------+
```

假设这些查询是在 2000~2049 年期间发出的，执行以下语句：

```
SELECT TO_CHAR(TO_DATE('27-OCT-98', 'DD-MON-RR'), 'YYYY') "Year1" ,
TO_CHAR(TO_DATE('27-OCT-17', 'DD-MON-RR'), 'YYYY') "Year2"  FROM DUAL;
```

查询结果如下：

```
+-------+-------+
| Year1 | Year2 |
+-------+-------+
| 2017  | 1998  |
+-------+-------+
```

> **注意**
>
> 无论是在 2000 年之前还是之后查询，都将返回相同的值。

6.4.4.5. 字符串到日期的转换规则

将字符串值转换为日期值时，会有下列转换规则：

- 如果指定了数值格式化元素的所有数值（包括前导零点），则可以从日期字符串中省略格式化字符串中包含的标点符号。为两位数格式化元素（如 `MM` 、 `DD` 和 `YY` ）指定 02 而不是 2。
- 您可以从日期字符串中省略在格式化字符串末尾找到的时间字段。
- 您可以在日期字符串中使用任何非字母数值字符来匹配格式化字符串中的标点符号。

6.4.5. 空值

6.4.5.1. 空值概述

空值（Null）指数据库表中无效的、未指定的、未知的或不可预知的值。空值的出现不受 `NOT NULL` 或 `PRIMARY KEY` 主键约束。任何包含 `NULL` 的算术表达式结果都为 `NULL` 。

OceanBase 支持以下 3 种空值类型。

SQL 函数中的空值

SQL 函数中的空值 指的是 SQL 函数的参数为空值，当 SQL 函数的参数为空值时，大多数标量函数都返回 `NULL` ，分析函数会忽略空值。此类别中有以下两种 SQL 函数：

空值	说明
NVL 函数中的空值	在表达式中 `NVL(expr1,expr2)` 中，如果 `expr1` 不是 NULL，返回 `expr1` ，否则返回 `expr2` 。
分析函数中的空值	使用 `AVG` 、 `MAX` 、 `SUM` 或 `COUNT` 等分析函数时，为 NULL 的纪录会被忽略。

比较条件中的空值

比较条件中的空值 指与任何其他条件做比较的 `NULL` 。测试空值只能用比较运算符 `IS NULL` 和 `IS NOT NULL` 。因为 `NULL` 表示缺少数据，所以 `NULL` 和其它值没有可比性，即不能用等于、不等于、大于或小于和其它数值比较，当然也包括空值本身。

条件判断表达式中的空值

条件判断表达式中的空值 指的是条件 `=NULL` 、 `!=NULL` 、 `NULL=` 、 `NULL！=` 中的 `NULL` ，作逻辑判断使用，判断结果不返回任何行，即 `UNKNOWN` 。

6.4.5.2. SQL 函数中的空值

SQL 函数中的空值指的是 SQL 函数的参数存在空值，当 SQL 函数的参数为空值时，大多数标量函数都返回 NULL，分析函数会忽略空值。您可以通过 NVL 函数的返回值确定空值。

NVL 函数中的空值

NVL 函数的表达式为 NVL(expr1,expr2)，如果 expr1 不是 NULL，返回 expr1，否则返回 expr2。

如下所示，给定 expr1 参数为 NULL，查询 NVL(expr1，0) 表达式的返回值。

执行以下语句：

```
SELECT NVL(NULL,0) FROM DUAL;
```

查询结果如下：

```
+-------------+
| NVL(NULL,0) |
+-------------+
|           0 |
+-------------+
```

expr1 是 NULL，则表达式 NVL(expr1，0) 的返回值为 0；expr1 不是 NULL，则表达式的返回值为 NULL。

分析函数中的空值

在使用 AVG，MAX，SUM，COUNT 等分析函数时，为 NULL 的纪录会被忽略。

如下所示，向 tbl_a 表中插入数据并执行以下语句：

```
CREATE TABLE tbl_a (col_a varchar2(1), col_b int );
INSERT INTO tbl_a VALUES (NULL, 3);
INSERT INTO tbl_a VALUES (NULL, NULL);
INSERT INTO tbl_a VALUES (NULL, 1);
```

执行以下语句：

```
SELECT * FROM tb1_a;
```

查询结果如下：

```
+-------+-------+
|COL_A|COL_B|
+-------+-------+
| NULL| 3 |
+-------+-------+
| NULL| NULL|
+-------+-------+
| NULL| 1 |
+-------+-------+
```

查询的结果如下：

```
SELECT AVG(col_b) FROM tbl_a; -- 结果为 2 ，
SELECT MAX(col_b) FROM tbl_a; -- 结果为 3
SELECT SUM(col_b) FROM tbl_a; -- 结果为 4
SELECT COUNT(col_b) FROM tbl_a; -- 结果为 2
SELECT COUNT(col_a) FROM tbl_a; -- 结果为 0
SELECT COUNT(*) FROM tbl_a;  -- 结果为 3
```

`NULL` 的纪录被忽略了。

6.4.5.3. 比较条件中的空值

比较条件中的空值指与任何其他条件做比较的 `NULL` 。测试空值只能用比较运算符 `IS NULL` 和 `IS NOT NULL` 。因为 `NULL` 表示缺少数据，所以 `NULL` 和其它值没有可比性，即不能用等于、不等于、大于或小于和其它数值比较，当然也包括空值本身。

另外，OceanBase 在计算 `DECODE` 函数时认为两个空值是相等的。若两个空值出现在复合键中，则它们也相等。

如下所示，根据 A 值判断比较条件的结果。

条件	A值	结果
A IS NULL	10	FALSE
A IS NOT NULL	10	TRUE
A IS NULL	NULL	TRUE
A IS NOT NULL	NULL	FALSE

6.4.5.4. 条件判断表达式中的空值

条件判断表达式中的空值指的是条件 `= NULL` 、 `!= NULL` 、 `NULL =` 、 `NULL ! =` 中的 `NULL` ，作逻辑判断使用，判断结果不返回任何行，即 `UNKNOWN` 。

在 OceanBase 中，测试空值要用比较运算符 `IS NULL` ，返回结果为 `TRUE` 或 `FALSE` 。但是条件判断表达式中空值的判断结果 `UNKNOWN` 与 `FALSE` 不同， `NOT FALSE` 判断结果为 `TRUE` ， `NOT UNKNOWN` 判断结果仍然为 `UNKNOWN` 。

如下所示，根据 A 值判断条件判断表达式的结果。

条件	A 值	结果
A = NULL	10	UNKNOWN
A != NULL	10	UNKNOWN
A = NULL	NULL	UNKNOWN
A != NULL	NULL	UNKNOWN
A = 10	NULL	UNKNOWN
A != 10	NULL	UNKNOWN

如果在 `SELECT` 语句的 `WHERE` 子句中使用了判断结果为 `UNKNOWN` 的条件，则该查询将不返回任何行。

6.4.6. 注释

6.4.6.1. 注释概述

OceanBase 中用户可以创建三种注释：

- SQL 语句中的注释：被存储为执行 SQL 语句的应用程序代码的一部分。
- Schema 和非 Schema 对象的注释：与对象本身的元数据一起存储在数据字典中。
- Hint：一种在 SQL 语句中将指令传递给 OceanBase 数据库优化器的注释。

6.4.6.2. SQL 语句的注释

注释可以使应用程序更易于阅读和维护。例如，您可以在语句中用注释以描述该语句在应用程序中的用途。除 Hint 外，SQL 语句中的注释不会影响语句的执行。

注释可以出现在语句中的任何关键字、参数或标点符号之间。您可以通过两种方式在语句中添加注释：

- 以斜杠和星号（/*）为开头的注释。斜杠和星号后跟着注释的文本。此文本可以跨越多行，并用星号和斜杠（*/）结束注释。开头和结尾的符号不必与文本用空格或换行符进行分隔。
- 以两个连字符（--）为开头的注释。符号后跟着注释的文本。此文本不能扩展到新行，并以换行符结束注释。

一个 SQL 语句可以同时包含这两种风格的多个注释。注释的文本可以包含数据库字符集中的任何可打印字符。

以下示例展示了多种形式的以斜杠和星号（/*）为开头的注释：

```
SELECT last_name, employee_id, salary + NVL(commission_pct, 0),
        job_id, e.department_id
 /* Select all employees whose compensation is
 greater than that of Pataballa.*/
 FROM employees e, departments d
 /*The DEPARTMENTS table is used to get the department name.*/
 WHERE e.department_id = d.department_id
  AND salary + NVL(commission_pct,0) >  /* Subquery:     */
    (SELECT salary + NVL(commission_pct,0)
     /* total compensation is salary + commission_pct */
     FROM employees
     WHERE last_name = 'Pataballa')
 ORDER BY last_name, employee_id;
```

以下示例中的语句包含多种形式的以两个连字符（- -）为开头的注释：

```
SELECT last_name,                    -- select the name
    employee_id                   -- employee id
    salary + NVL(commission_pct, 0),        -- total compensation
    job_id,                       -- job
    e.department_id                  -- and department
 FROM employees e,                   -- of all employees
    departments d
 WHERE e.department_id = d.department_id
  AND salary + NVL(commission_pct, 0) >       -- whose compensation
                                -- is greater than
    (SELECT salary + NVL(commission_pct,0)    -- the compensation
     FROM employees
     WHERE last_name = 'Pataballa')       -- of Pataballa
 ORDER BY last_name                  -- and order by last name
       employee_id                -- and employee id.
;
```

6.4.6.3. Schema 与非 Schema 对象的注释

可以使用 `COMMENT` 语句将注释与 Schema 对象（表、视图、物化视图、运算符、索引类型）或非 Schema 对象（Edition）关联起来。还可以在表模式对象的列上创建注释。与 Schema 和非 Schema 对象关联的注释存储在数据字典中。

```
语法格式：
COMMENT ON {TABLE table | COLUMN column | INDEXTYPE indextype
  | OPERATOR operator | VIEW view} IS string
```

例如：我们需要对表user表注释：

```
COMMENT ON TABLE test.user is "这是记录用户信息的表";
```

例如：我们需要对user_name列注释：

```
COMMENT ON COLUMN test.user.user_name is "这里记录了用户的姓名";
```

6.4.6.4. Hint

6.4.6.4.1. Hint 概述

Hint 是 SQL 语句中将指令传递给 OceanBase 数据库优化器或服务器的一种注释。通过 Hint 可以使优化器或服务器生成某种特定的计划。 一般情况下，优化器会为用户查询选择最佳的执行计划，不需要用户主动使用 Hint 指定，但在某些场景下，优化器生成的执行计划可能不满足用户的要求，这时就需要用户使用 Hint 来主动指定并生成特殊的执行计划。

Hint 应该少用，仅在您收集了相关表的统计信息并且使用 `EXPLAIN PLAN` 语句在没有 Hint 的情况下评估了优化器计划之后，才谨慎考虑使用。更改数据库条件以及在后续版本中增强查询性能可能会导致您代码中的 Hint 对性能产生重大影响。

Hint 的使用

一个语句块只能有一个注释包含 Hint，并且该注释必须跟随 `SELECT` 、 `UPDATE` 、 `INSERT` 、 `MERGE` 或 `DELETE` 关键字。

以下是 Hint 在语句块注释中的语法格式：

```
/*+[hint text]*/
```

Hint 从语法上看是一种特殊的 SQL 注释, 所不同的是在注释的左标记后增加了一个加号（+）。 如果服务器端无法识别 SQL 语句中的 Hint，那么优化器会选择忽略用户指定的 Hint 而使用默认计划所生成逻辑。另外需要指明的是，Hint 只影响优化器所生成的计划的逻辑，而不影响 SQL 语句的语义。

以下是定义 Hint 时需要注意的一些规则：

- 加号（+）使数据库将注释解释为 Hint 列表。加号必须紧跟在注释左标记符后，不允许有空格。

- 加号（+）和 Hint 文本之间的空格是可选的。如果注释中包含多个 Hint，则 Hint 间至少用一个空格进行分隔。
- Hint 包含拼写错误或语法错误时会被忽略。但是，数据库会考虑在同一注释中其他正确指定的 Hint。
- 不跟随 `DELETE`、`INSERT`、`MERGE`、`SELECT` 或 `UPDATE` 关键字的 Hint 无效。
- Hint 的组合相互冲突时 Hint 无效。但是，数据库会在同一注释中考虑其他不冲突的 Hint。
- 数据库环境使用 PL/SQL 版本 1 时 Hint 无效，例如 Forms 版本 3 触发器。

在 Hint 中定义查询块

您可以通过在许多 Hint 中定义一个可选的查询块名称，以此来指定该 Hint 适用的查询块。使用此语法，允许您在外部查询中指定一个应用于嵌入式视图的 Hint。

查询块参数的语法格式为 `@queryblock`，其中 `queryblock` 是在查询中被指定的查询块的标识符。`queryblock` 标识符可以是系统生成的，也可以是用户自己指定的。当您在查询块中直接指定要应用的 Hint 时，将忽略 `@queryblock`。

- 系统生成的标识符可以通过对查询使用 `EXPLAIN PLAN` 生成，预转换查询块的名称可以通过对使用了 `NO_QUERY_TRANSFORMATION` Hint 的查询运行 `EXPLAIN PLAN` 生成。
- 可以使用 `QB_NAME` 来指定用户自定义的名称。

定义全局 Hint

许多 Hint 既可以应用于特定的表或索引，也可以更全局地应用于视图中的表或一部分索引的列。语法元素 `tablespec` 和 `indexspec` 定义了这些全局 Hint。

以下是 `tablespec` 的语法：

```
[ view.[ view. ]... ]table
```

您必须完全按照在语句中显示的方式来指定要访问的表。如果该语句使用表的别名，则在 Hint 中使用也使用别名而不是表名。但是，即使 Schema 名称出现在语句中，也不要在 Hint 中使用的表名中包含 Schema 名称。

> **注意**
>
> 使用 `tablespec` 子句指定全局 Hint 对于使用 ANSI 连接的查询不起作用，因为优化器在解析期间会生成额外的视图。相反，可以通过 `@queryblock` 来指定该提示所应用到的查询块。

以下是 `indexspec` 的语法：

```
{ index
| ( [ table. ]column [ [ table. ]column ]... )
}
```

在 Hint 的说明部分，当 `tablespec` 后跟着 `indexspec` 时，允许但不要求使用逗号来分隔表名和索引名。也允许（但不是必需）使用逗号分隔多次出现的 `indexspec`。

6.4.6.4.2. Hint 清单

6.4.6.4.2.1. 与访问路径相关的 Hint

INDEX Hint

`INDEX` Hint 指示优化器对指定的表使用索引扫描。您可以将 `INDEX` Hint 用于基于函数、域、B - 树、位图和位图连接的索引。

以下是 `INDEX` Hint 的语法：

```
/*+ INDEX ( [ @ queryblock ] tablespec [ indexspec [ indexspec ]... ] ) */
```

Hint 的行为取决于 `indexspec` 规范：

- 如果 `INDEX` Hint 指定了一个单个可用索引，则数据库将对该索引执行扫描。优化器不考虑全表扫描或表上另一个索引的扫描。
- 如果 `INDEX` Hint 指定了可用索引的列表，那么优化器将考虑扫描列表中每个索引的成本，然后以最低的成本执行索引扫描。如果数据库从这个列表中扫描多个索引并合的访问路径成本最低，数据库将选用这种扫描方案。数据库不考虑对没有在列表中的索引进行全表扫描或扫描。
- 如果 `INDEX` Hint 没有指定具体的索引，那么优化器将考虑表上每个可用索引的扫描成本，然后以最低的成本执行索引扫描。如果数据库扫描多个索引并合的访问路径成本最低，数据库将选用这种扫描方案。优化器不考虑全表扫描。

示例如下：

```
SELECT /*+ INDEX (employees emp_department_ix)*/ employee_id, department_id
 FROM employees
 WHERE department_id > 50;
```

FULL Hint

`FULL` Hint 指示优化器对指定的表执行全表扫描。

以下是 `FULL` Hint 的语法：

```
/*+ FULL ( [ @ queryblock ] tablespec ) */
```

示例如下：

```
SELECT /*+ FULL(e) */ employee_id, last_name
 FROM hr.employees e
 WHERE last_name LIKE :b1;
```

数据库对表 employees 执行一次完整的表扫描来执行这条语句，即使有一个由 WHERE 子句中的条件提供的索引在列 last_name 上。

在 FROM 子句中，表 employees 有个别名 e，因此 Hint 必须根据表的别名而不是名称来引用该表。即使在 FROM 子句中指定了 Schema 名，也不要在 Hint 中引用它们。

6.4.6.4.2.2. 与联接顺序相关的 Hint

LEADING Hint

LEADING Hint 指示优化器在执行计划中使用指定的表集作为前缀，它可以用来指定表的联接顺序。这个 Hint 比 ORDERED Hint 更通用。

以下是 LEADING Hint 的语法：

```
/*+ LEADING ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

LEADING Hint 为确保按照用户指定的顺序联接表，所以会进行严格的检查。如果发现 Hint 指定的 table_name 不存在，则 LEADING Hint 失效。如果发现 Hint 中存在重复表，则 LEADING Hint 失效。如果在优化器联接期间，无法找到对应的表，那么该表及后面的表指定的联接顺序失效，该表前面指定的顺序依然有效。如果由于联接图中的依赖关系，无法首先按照指定的顺序联接指定的表，则 LEADING Hint 失效。如果指定两个或多个相互冲突的 LEADING Hint，则 LEADING Hint 失效。如果您指定了 ORDERED Hint，它将覆盖所有的 LEADING Hint。

示例如下：

```
SELECT /*+ LEADING(e j) */ *
  FROM employees e, departments d, job_history j
  WHERE e.department_id = d.department_id
   AND e.hire_date = j.start_date;
```

ORDERED Hint

ORDERED Hint 指示数据库按照表在 FROM 子句中出现的顺序联接表。建议使用 LEADING Hint，它比 ORDERED Hint 更通用。

以下是 ORDERED Hint 的语法：

```
/*+ ORDERED */
```

当您从需要联接的SQL语句中省略 ORDERED Hint 时，将由优化器将选择联接表的顺序。但是优化器不知道从每个表中要选择的行数，此时您可以使用 ORDERED Hint 来指定联接顺序。这样使您能够比优化器更好地选择内部表和外部表。如果在指定该 ORDERED Hint 后发生了改写，那么就按照改写后的语句中的 FROM 子句的顺序联接表。

6.4.6.4.2.3. 与联接操作相关的 Hint

USE_MERGE Hint

USE_MERGE Hint 指示优化器使用一个 sort-merge 联接将每个指定的表与另一个行资源联接起来。建议在使用 USE_NL 和 USE_MERG Hint 时和 LEADING 与 ORDERED Hint 一起使用。当被引用的表是联接的内部表时，优化器将使用这些提示。如果被引用的表是外部表，则忽略 Hint。

以下是 USE_MERGE Hint 的语法：

```
/*+ USE_MERGE ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

USE_MERGE 指定表作为内部表时候使用 MERGE-JOIN 算法。OceanBase 中使用 MERGE-JOIN 算法时必须有等值条件的 join-condition，因此无等值条件的两个表做联结时， USE_MERGE 失效。

以下是 USE_MERGE 的示例：

```
SELECT /*+ USE_MERGE(employees departments) */ *
 FROM employees, departments
 WHERE employees.department_id = departments.department_id;
```

NO_USE_MERGE Hint

NO_USE_MERGE Hint 指示优化器在使用指定表作为内部表并联接到另一个行资源时排除 USE_MERGE Hint 使用的联接。

以下是 NO_USE_MERGE Hint 的语法：

```
/*+ NO_USE_MERGE ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

以下是 NO_USE_MERGE Hint 的示例：

```
SELECT /*+ NO_USE_MERGE(e d) */ *
 FROM employees e, departments d
 WHERE e.department_id = d.department_id;
```

USE_HASH Hint

USE_HASH Hint 指示优化器使用 HASH-JOIN 算法将每个指定的表与另一个行资源联接起来。

以下是 USE_HASH Hint 的语法：

```
/*+ USE_HASH ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

以下是 USE_HASH Hint 的示例：

```
SELECT /*+ USE_HASH(l h) */ *
 FROM orders h, order_items l
 WHERE l.order_id = h.order_id
  AND l.order_id > 2400;
```

NO_USE_HASH Hint

`NO_USE_HASH` Hint 指示优化器在使用指定表作为内部表并联接到另一个行资源时排除 `USE_HASH` Hint 使用的联接。

以下是 `NO_USE_HASH` Hint 的语法：

```
/*+ NO_USE_HASH ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

以下是 `NO_USE_HASH` Hint 的示例：

```
SELECT /*+ NO_USE_HASH(e d) */ *
 FROM employees e, departments d
 WHERE e.department_id = d.department_id;
```

USE_NL Hint

`USE_NL` Hint 指示优化器使用嵌套循环联接将每个指定的表连接到另一个行资源，并使用指定的表作为内部表，指定表作为内部表时使用 NL-JOIN 算法。建议在使用 `USE_NL` 和 `USE_MERG` Hint 时和 `LEADING` 与 `ORDERED` Hint 一起使用。当被引用的表是联接的内部表时，优化器将使用这些提示。如果被引用的表是外部表，则忽略 Hint。

以下是 `USE_NL` Hint 的语法：

```
/*+ USE_NL ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

在下面的示例中，Hint 强制执行了嵌套循环，通过全表扫描访问了 orders 并且筛选条件 `l.order_id = h.order_id` 应用在了每一行。对于满足筛选条件的每一行，通过索引 order_id 访问 order_items：

```
SELECT /*+ USE_NL(l h) */ h.customer_id, l.unit_price * l.quantity
 FROM orders h, order_items l
 WHERE l.order_id = h.order_id;
```

NO_USE_NL Hint

`NO_USE_NL` Hint 指示优化器在使用指定表作为内部表并联接到另一个行资源时排除 Nest-Loop 联接。

以下是 `NO_USE_NL` Hint 的语法：

```
/*+ NO_USE_NL ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

以下是 NO_USE_NL Hint 的示例：

```
SELECT /*+ NO_USE_NL(e d) */ *
 FROM employees e, departments d
 WHERE e.department_id = d.department_id;
```

USE_BNL Hint

USE_BNL Hint 指示优化器使用块嵌套循环联接将每个指定的表连接到另一个行资源，并使用指定的表作为内部表，指定表作为内部表时使用 BNL-JOIN 算法。建议在使用 USE_BN Hint 时和 LEADING 与 ORDERED Hint 一起使用。当被引用的表是联接的内部表时，优化器将使用这些提示。如果被引用的表是外部表，则忽略 Hint。

以下是 USE_BNL Hint 的语法：

```
/*+ USE_BNL ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

在下面的示例中，Hint 强制执行了块嵌套循环，通过全表扫描访问了 orders 并且筛选条件 l.order_id = h.order_id 应用在了每一行。对于满足筛选条件的每一行，通过索引 order_id 访问 order_items：

```
SELECT /*+ USE_BNL(l h) */ h.customer_id, l.unit_price * l.quantity
 FROM orders h, order_items l
 WHERE l.order_id = h.order_id;
```

NO_USE_BNL Hint

NO_USE_BNL Hint 指示优化器在使用指定表作为内部表并联接到另一个行资源时排除 USE_BNL Hint 使用的联接。

以下是 NO_USE_BNL Hint 的语法：

```
/*+ NO_USE_BNL ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

以下是 NO_USE_BNL Hint 的示例：

```
SELECT /*+ NO_USE_BNL(e d) */ *
 FROM employees e, departments d
 WHERE e.department_id = d.department_id;
```

6.4.6.4.2.4. 与并行执行相关的 Hint

PARALLEL Hint

PARALLEL Hint 是语句级的 Hint，用来指示优化器指定并行操作可使用的并行服务器的数量。此 Hint 将覆盖初始化参数 PARALLEL_DEGREE_ POLICY 的值。该 Hint 适用于语句的 SELECT 、 INSERT 、 MERGE 、 UPDATE 和 DELETE 部分，以及表扫描的部分。如果违反了任何的并行限制，则 PARALLEL Hint 被忽略。

以下是 PARALLEL Hint 的语法：

```
/*+ PARALLEL [ ( DEFAULT | AUTO | MANUAL | integer ) ] */
```

> 注意
>
> 如果还进行了排序或分组操作，那么可以使用的服务器数量是 PARALLEL Hint 中的值的两倍。

PARALLE Hint 中指定参数的值时：

- PARALLEL ：数据库计算并行度，可以为 2 或更大。语句一直并行执行。
- PARALLEL(DEFAULT) ：优化器计算的并行度等于所有参与实例上可用的 CPU 数量乘以初始化参数 PARALLEL_THREADS_PER_CPU 的值。
- PARALLEL(AUTO) ：数据库计算并行度，结果可以大于等于 1，如果计算出的并行度为 1，则该语句按顺序运行。
- PARALLEL(MANUAL) ：优化器被强制使用语句中设置的对象的并行度。
- PARALLEL(integer) ：优化器使用参数 integer 指定的整数值为并行度。

以下示例中数据库计算并行度，并且语句一直并行执行：

```
SELECT /*+ PARALLEL */ last_name
 FROM employees;
```

以下示例中数据库计算并行度，但是并行度为 1，所以该语句串行执行：

```
SELECT /*+ PARALLEL (AUTO) */ last_name
 FROM employees;
```

在以下示例中， PARALLEL Hint 建议优化器使用语句中设定的，当前对表本身有效的并行度 5：

```
CREATE TABLE parallel_table (col1 number, col2 VARCHAR2(10)) PARALLEL 5;
SELECT /*+ PARALLEL (MANUAL) */ col2
 FROM parallel_table;
```

USE_PX Hint

USE_PX Hint 强制指示服务器在执行 SQL 语句时使用 PX 模式，PX 模式允许在执行语句时采用多线程方式。一般 USE_PX Hint 和 PARALLEL Hint 配合使用。

以下是 USE_PX Hint 的语法：

```
/*+ USE_PX */
```

示例如下：

```
SELECT /*+ USE_PX PARALLEL(4)*/ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

NO_USE_PX Hint

NO_USE_PX Hint 强制指示服务器在执行 SQL 语句时避免使用 PX 模式。

以下是 NO_USE_PX Hint 的语法：

```
/*+ NO_USE_PX */
```

示例如下：

```
SELECT /*+ NO_USE_PX*/ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

PQ_DISTRIBUTE Hint

PQ_DISTRIBUTE Hint 指示优化器怎样在程序（查询）服务器和消耗（负载）查询服务器之间分配行。您可以通过该 Hint 控制联接或负载的行分布 PQ_DISTRIBUTE Hint 的语法：

```
/*+ PQ_DISTRIBUTE
 ( [ @ queryblock ] tablespec
  { distribution | outer_distribution inner_distribution }
 ) */
```

控制负载的分配

您可以控制并行语句 INSERT ... SELECT 和并行语句 CREATE TABLE ... AS SELECT 的行分布，以此来确定如何在程序（查询）服务器和消耗（负载）服务器之间进行行分配。使用语法的上分支来指定分发方法。分布方法的值及其语义如下表所示：

分布方法	说明
NONE	没有分配。即将查询和负载操作组合到每个查询服务器中。所有服务器将加载所有分区。这种分配方法的缺失有助于避免在没有偏离的情况下行分配的开销。由于空段或语句中的谓词会过滤掉查询评估的所有行，因此可能会发生偏离。如果由于使用此方法而发生偏斜，则请改用 RANDOM 或 RANDOM_LOCAL 分布。 **注意** 请谨慎使用此分配方法。每个进程加载的 PGA 内存最少需要 512 KB。如果还使用压缩，则每台服务器消耗大约 1.5 MB 的 PGA 内存。
PARTITION	此方法使用 tablespec 的分区信息将行从查询服务器分发到消耗服务器。当不可能或不希望将查询和加载操作组合在一起时和当正在加载的分区数量大于或等于加载服务器的数量并且输入数据将均匀地分布在正在加载的分区之间（即没有偏离）时，请使用此分步方法。
RANDOM	此方法以循环方式将来自程序的行分发给消耗。当输入数据高度倾斜时，使用这种分布方法。
RANDOM_LOCAL	此方法将来自程序的行分布到一组服务器，这些服务器负责维护给定的一组服务器。两个或多个服务器可以加载同一分区，但是没有服务器加载所有分区。当输入数据发生偏移并且由于内存限制而无法合并查询和加载操作时，请使用此分布方法。

例如，在以下直接装入插入操作中，该操作的查询和负载部分被组合到每个查询服务器中：

```
INSERT /*+ APPEND PARALLEL(target_table, 16) PQ_DISTRIBUTE(target_table, NONE) */
 INTO target_table
 SELECT * FROM source_table;
```

在下面的示例中，创建表时优化器使用表 target_table 的分区来分配行：

```
CREATE /*+ PQ_DISTRIBUTE(target_table, PARTITION) */ TABLE target_table
 NOLOGGING PARALLEL 16
 PARTITION BY HASH (l_orderkey) PARTITIONS 512
 AS SELECT * FROM source_table;
```

控制联接的分配

您可以通过指定两种分配方法来控制联接的分配方法，如语法中的下部分支所示，一种外部表的分布方法，一种内部表的分布方法：

- `outside_distribution` 是外部表的分布方法。
- `inner_distribution` 是内部表的分布方法。

分布方法的值是 `HASH` 、 `BROADCAST` 、 `PARTITION` 和 `NONE` 。只有下表中的 6 种分布方法组合是有效的：

分布方法	说明
HASH, HASH	使用联接键上的哈希函数，将每个表的行映射到消耗查询服务器。映射完成后，每个查询服务器都会在一对结果分区之间执行联接。当表的大小可比较并且联接操作是通过哈希联接或排序合并联接实现时，建议使用此分布方法。
BROADCAST, NONE	外部表的所有行都广播到每个程序查询服务器。内部表行是随机分区的。当外部表与内部表相比非常小时，建议使用此分布方法。通常，当内部表大小乘以查询服务器的数量大于外部表大小时，也建议使用此分布方法。
NONE, BROADCAST	内部表的所有行都广播给每个消耗查询服务器。外部表行是随机分区的。当内部表与外部表相比非常小时，建议使用此分布方法。通常，当内部表大小乘以查询服务器的数量小于外部表大小时，也建议使用此分布方法。
PARTITION, NONE	外部表的行使用以内部表的分区进行映射。内部表必须在联接键上进行分区。当外部表的分区数等于或几乎等于查询服务器数的倍数时，建议使用此分布方法。例如，有 14 个分区和 15 个查询服务器。 **注意** 如果内部表未分区或未在分区键上等分联接时，则优化器将忽略此 Hint。

分布方法	说明
NONE, PARTITION	内部表的行使用外部表的分区进行映射。外部表必须在联接键上进行分区。当外部表的分区数等于或几乎等于查询服务器数的倍数时，建议使用此分布方法。例如，有 14 个分区和 15 个查询服务器。 **注意** 如果外部表未在分区键上进行分区或未等分联时接，则优化器将忽略此 Hint
NONE, NONE	每个查询服务器在一对匹配的分区之间执行联接操作，每个表中都有一个。两个表必须在连接键上等分。

例如，给定两个使用哈希联接来联接表 r 和 s，以下查询包含使用哈希分配的 Hint：

```
SELECT /*+ORDERED PQ_DISTRIBUTE(s HASH, HASH) USE_HASH (s)*/ column_list
 FROM r,s
 WHERE r.c=s.c;
```

要广播外部表 r，查询语句为：

```
SELECT /*+ORDERED PQ_DISTRIBUTE(s BROADCAST, NONE) USE_HASH (s) */ column_list
 FROM r,s
 WHERE r.c=s.c;
```

6.4.6.4.2.5. 与查询策略相关的 Hint

USE_JIT Hint

`USE_JIT` Hint 指示服务器在执行 SQL 语句时强制使用 JIT 模式编译执行表达式。

以下是 `USE_JIT` Hint 的语法：

```
/*+ USE_JIT */
```

示例如下：

```
SELECT /*+ USE_JIT*/ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

NO_USE_JIT Hint

`NO_USE_JIT` Hint 指示服务器在执行 SQL 语句时避免使用 JIT 模式编译执行表达式。

以下是 `NO_USE_JIT` Hint 的语法：

```
/*+ NO_USE_JIT*/
```

示例如下：

```
SELECT /*+NO_USE_JIT*/ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

USE_HASH_AGGREGATION Hint

`USE_HASH_AGGREGATION` Hint 指示优化器在生成计划时强制使用 HASH 聚合算法运行该 SQL 语句。

以下是 `USE_HASH_AGGREGATION` Hint 的语法：

```
/*+ USE_HASH_AGGREGATION */
```

示例如下：

```
SELECT /*+ USE_HASH_AGGREGATION */ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

NO_USE_HASH_AGGREGATION Hint

`NO_USE_HASH_AGGREGATION` Hint 指示优化器在执行 SQL 语句时避免使用 HASH 聚合算法运行该语句。

以下是 `NO_USE_HASH_AGGREGATION` Hint 的语法：

```
/*+ NO_USE_HASH_AGGREGATION */
```

示例如下：

```
SELECT /*+ NO_USE_HASH_AGGREGATION */ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

USE_LATE_MATERIALIZATION Hint

`USE_LATE_MATERIALIZATION` Hint 指示优化器延迟物化视图。

以下是 `USE_LATE_MATERIALIZATION` Hint 的语法：

```
/*+ USE_LATE_MATERIALIZATION */
```

示例如下：

```
SELECT /*+ USE_LATE_MATERIALIZATION*/ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

NO_USE_LATE_MATERIALIZATION Hint

`NO_USE_LATE_MATERIALIZATION` Hint 指示优化器禁止延迟物化视图。

以下是 `NO_USE_LATE_MATERIALIZATION` Hint 的语法：

```
/*+ NO_USE_LATE_MATERIALIZATION */
```

示例如下：

```
SELECT /*+ NO_USE_LATE_MATERIALIZATION*/ e.department_id, sum(e.salary)
 FROM employees e
 WHERE e.department_id = 1001;
 GROUP BY e.department_id;
```

USE_NL_MATERIALIZATION Hint

`USE_NL_MATERIALIZATION` Hint 强制指示优化器指定表为内部表（子树）时生成一个物化算子来缓存数据。

以下是 `USE_NL_MATERIALIZATION` Hint 的语法：

```
/*+ USE_NL_MATERIALIZATION ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

示例如下：

```
SELECT /*+ USE_NL_MATERIALIZATION(departments) */ *
 FROM employees, departments
 WHERE employees.department_id = departments.department_id;
```

NO_USE_NL_MATERIALIZATION Hint

`NO_USE_NL_MATERIALIZATION` Hint 强制指示优化器在指定表为内部表（子树）时避免生成一个物化算子来缓存数据。

以下是 `NO_USE_NL_MATERIALIZATION` Hint 的语法：

```
/*+ NO_USE_NL_MATERIALIZATION ( [ @ queryblock ] tablespec [ tablespec ]... ) */
```

示例如下：

```
SELECT /*+ NO_USE_NL_MATERIALIZATION(departments) */ *
 FROM employees, departments
 WHERE employees.department_id = departments.department_id;
```

6.4.6.4.2.6. 与查询转换相关的 Hint

NO_REWRITE Hint

`NO_REWRITE` Hint 指示优化器为禁用查询去重写查询块，并覆盖了参数 `QUERY_REWRITE_ENABLED` 的设置。

以下是 `NO_REWRITE` 的语法：

```
/*+ NO_REWRITE [ ( @ queryblock ) ] */
```

示例如下：

```
SELECT /*+ NO_REWRITE */ sum(s.amount_sold) AS dollars
 FROM sales s, times t
 WHERE s.time_id = t.time_id
 GROUP BY t.calendar_month_desc;
```

NO_EXPAND Hint

`NO_EXPAND` Hint 指示优化器不要对 `WHERE` 子句中具有 `OR` 条件或 `IN` 列表的查询考虑 `OR` 扩展。通常，优化器会使用 `OR` 扩展，当确定使用 `OR` 扩展的成本低于不使用它时。

以下是 `NO_EXPAND` Hint 的语法：

```
/*+ NO_EXPAND [ ( @ queryblock ) ] */
```

示例如下：

```
SELECT /*+ NO_EXPAND */ *
 FROM employees e, departments d
 WHERE e.manager_id = 108
   OR d.department_id = 110;
```

USE_CONCAT Hint

USE_CONCAT Hint 指示优化器使用 UNION ALL 运算符将查询 WHERE 子句中的组合 OR 条件转换为复合查询。 如果没有这个 Hint，则仅当使用串联查询的成本低于比没有串联查询的成本时，才会发生此转换。 USE_CONCAT Hint 将覆盖成本注意事项。

以下是 USE_CONCAT Hint 的语法：

```
/*+ USE_CONCAT [ ( @ queryblock ) ] */
```

示例如下：

```
SELECT /*+ USE_CONCAT */ *
 FROM employees e
 WHERE manager_id = 108
   OR department_id = 110;
```

MERGE Hint

MERGE Hint 使您可以在查询中合并视图。

以下是 MERGE Hint 的语法：

```
/*+ MERGE [ ( @ queryblock ) | ( [ @ queryblock ] tablespec ) ] */
```

如果视图的查询块在 SELECT 列表中包含 GROUP BY 子句或 DISTINCT 运算符时，只有启用了复杂的视图合并后，优化器才能将视图合并到正在访问的语句中。 如果子查询不相关，则也可以使用复杂合并将 IN 子查询合并到访问语句中。

示例如下：

```
SELECT /*+ MERGE(v) */ e1.last_name, e1.salary, v.avg_salary
  FROM employees e1,
     (SELECT department_id, avg(salary) avg_salary
       FROM employees e2
       GROUP BY department_id) v
  WHERE e1.department_id = v.department_id
   AND e1.salary > v.avg_salary
  ORDER BY e1.last_name;
```

当不带参数使用 `MERGE` Hint 时，应将其放在视图查询块中。 当视图名称作为参数使用 `MERGE` Hint 时，应将其放在周边查询中。

NO_MERGE Hint

`NO_MERGE` Hint 指示优化器不要将外部查询和任何内联视图查询合并到单个查询中。

以下是 `NO_MERGE` Hint 的语法：

```
/*+ NO_MERGE [ ( @ queryblock ) | ( [ @ queryblock ] tablespec ) ] */
```

此 Hint 会影响您访问视图的方式。 例如，以下语句导致视图 seattle_dept 不被合并：

```
SELECT /*+ NO_MERGE(seattle_dept) */ e1.last_name, seattle_dept.department_name
  FROM employees e1,
     (SELECT location_id, department_id, department_name
       FROM departments
       WHERE location_id = 1700) seattle_dept
  WHERE e1.department_id = seattle_dept.department_id;
```

在视图查询块中使用 `NO_MERGE` Hint 时，不需要为它指定参数。 在周围的查询中使用 `NO_MERGE` Hint 时，需要将视图名称指定为它的参数。

UNNEST Hint

`UNNEST` Hint 指示优化器不要嵌套而是并将子查询的主体合并到包含该 Hint 的查询块的主体中，从而使优化器在评估访问路径和联接时将它们一起考虑在内。

以下是 `UNNEST` Hint 的语法：

```
/*+ UNNEST [ ( @ queryblock ) ] */
```

在取消嵌套子查询之前，优化器首先验证子查询是否有效，并且必须通过探索和查询优化测试。 使用 `UNNEST` Hint 时优化器仅检查子查询块的有效性。 如果子查询块有效，那么将直接取消嵌套子查询，而无需通过探索和查询优化测试。

示例如下：

```
SELECT AVG(t1.c) FROM t1
WHERE t1.b >=
  (SELECT /*+unnest*/AVG(t2.b)
   FROM t2
   WHERE t1.a = t2.a);
```

NO_UNNEST Hint

`NO_UNNEST` Hint 用来关闭取消嵌套的操作。

以下是 `NO_UNNEST` Hint 的语法：

```
/*+ NO_UNNEST [ ( @ queryblock ) ] */
```

示例如下：

```
SELECT /*+no_unnest(@qb1)*/AVG(t1.c)
FROM t1 WHERE t1.b >=
  (SELECT /*+qb_name(qb1)*/AVG(t2.b)
   FROM t2)
   WHERE t1.a = t2.a);
```

PLACE_GROUP_BY Hint

`PLACE_GROUP_BY` Hint指示优化器采用 GROUP BY 位置替换规则，此时优化器不考虑转换之后的代价增大。

以下是 `PLACE_GROUP_BY` Hint的语法：

```
/*+ PLACE_GROUP_BY [ ( @ queryblock ) ] */
```

示例如下：

```
SELECT /*+place_group_by*/SUM(t1.c),SUM(t2.c) FROM t1, t2
WHERE t1.a = t2.a AND t1.b > 10 AND t2.b > 10
GROUP BY  t1.a;
```

NO_PLACE_GROUP_BY Hint

`NO_PLACE_GROUP_BY` Hint 用来关闭 GROUP BY 位置替换转换。

以下是 `NO_PLACE_GROUP_BY` Hint的语法：

```
/*+ NO_PLACE_GROUP_BY [ ( @ queryblock ) ] */
```

示例如下：

```
SELECT /*+no_place_group_by*/SUM(t1.c),SUM(t2.c) FROM t1, t2
WHERE t1.a = t2.a AND t1.b > 10 AND t2.b > 10
GROUP BY  t1.a;
```

NO_PRED_DEDUCE Hint

`NO_PRED_DEDUCE` Hint 用来指示优化器不是用谓词推导转换规则。

以下是 `NO_PRED_DEDUCE` Hint的语法：

```
/*+ NO_PRED_DEDUCE [ ( @ queryblock ) ] */
```

示例如下：

```
SELECT /*+no_pred_deduce(@qb1)*/ *
FROM  (
 SELECT /*+no_merge qb_name(qb1)*/ t1.a, t2.b
 FROM t1, t2
 WHERE t1.a = t2.a) v, t3
WHERE t3.a = 1 AND t3.a = v.a;
```

6.4.6.4.2.7. 其他 Hint

QB_NAME Hint

使用 `QB_NAME` Hint 来定义查询块的名称。然后，可以在外部查询的 Hint 中使用这个名称，也可以在内联视图的 Hint 中使用这个名称，从而影响查询在被命名查询块中的表上的执行。更多关于查询块名称的信息，请参阅文档 Hint 概述 中的在 Hint 中定义查询块。

以下是 `QB_NAME` Hint 的语法：

```
/*+ QB_NAME ( queryblock ) */
```

如果两个或多个查询块具有相同的名称，或者同一个查询块两次被 Hint 指定了不同的名称，那么优化器将忽略所有引用该查询块的名称和 Hint。未使用 `QB_NAME` Hint 命名的查询块具有由系统生成的惟一名称。这些名称可以显示在计划表中，也可以在查询块中的其他 Hint 中使用。

以下是 `QB_NAME` Hint 的示例：

```
SELECT /*+ QB_NAME(qb) FULL(@qb e) */ employee_id, last_name
 FROM employees e
 WHERE last_name = 'Smith';
```

READ_CONSISTENCY Hint

READ_CONSISTENCY Hint 指示服务器去指定某条 SQL 所读取的表模式为弱一致（指定参数 WEAK ）或强一致性（指定参数 STRONG ）。

以下是 READ_CONSISTENCY Hint 的语法：

```
/*+ READ_CONSISTENCY(WEAK[STRONG]) */
```

示例如下：

```
SELECT /*+ READ_CONSISTENCY(WEAK) */ *
 FROM employees
 WHERE employees.department_id = 1001;
```

FROZEN_VERSION Hint

FROZEN_VERSION Hint 指示服务器读取某个基线数据的版本。

以下是 FROZEN_VERSION Hint 的语法：

```
/*+ FROZEN_VERSION (intnum) */
```

示例如下：

```
SELECT /*+ FROZEN_VERSION(1000) */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

QUERY_TIMEOUT Hint

QUERY_TIMEOUT Hint 指示服务器设定某条 SQL 执行时的超时时间，单位为微妙。

以下是 QUERY_TIMEOUT Hint 的语法：

```
/*+ QUERY_TIMEOUT (intnum) */
```

示例如下，当该查询 1 秒之内未执行完该语句即返回超时错误：

```
SELECT /*+ QUERY_TIMEOUT(1000000) */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

LOG_LEVEL Hint

LOG_LEVEL Hint 指示服务器运行某句 SQL 时采用何种日志级别来执行。

以下是 LOG_LEVEL Hint 的语法：

```
/*+ LOG_LEVEL (['']log_level['']) */
```

以下示例采用 DEBUG 日志级别来执行该 SQL 语句：

```
SELECT /*+ LOG_LEVEL(DEBUG) */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

USE_PLAN_CACHE Hint

USE_PLAN_CACHE Hint 指示服务器执行某条 SQL 时是否要在计划缓存机制下运行，参数 NONE 为不执行计划缓存机制，参数 DEFAULT 表示按照服务器本身的设置来决定是否执行计划缓存机制。

以下是 USE_PLAN_CACHE Hint 的语法：

```
/*+ USE_PLAN_CACHE (NONE[DEFAULT]) */
```

示例如下，以下语句不执行计划缓存机制：

```
SELECT /*+ USE_PLAN_CACHE(NONE) */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

TRANS_PARAM Hint

TRANS_PARAM Hint 指示服务器执行事务时是否要按照参数 param 指定的参数来执行，现在支持的参数只有事务层面的提前解行锁 FORCE_EARLY_LOCK_FREE 参数， FORCE_EARLY_LOCK_FREE 的值为 TRUE 时表示支持，FALSE 表示不支持。注意这里的参数名和参数值要用单引号（' '）引起来，当参数的值为数值型时可以不用引号引起来。

以下是 TRANS_PARAM Hint 的语法：

```
/*+ TRANS_PARAM ['param' , 'param_value'] */
```

示例如下：

```
SELECT /*+ TRANS_PARAM('FORCE_EARLY_LOCK_FREE' 'TRUE') */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

TRACING Hint

TRACING Hint 指示服务器对某些执行计划中的算子采用 TRACING 跟踪。

以下是 TRACING Hint 的语法：

```
/*+ TRACING(TRACING_NUM_LIST)*/
```

示例如下：

```
SELECT /*+ TRACING(1) */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

STAT Hint

STAT Hint 指示对某些执行计划中的算子采用 STAT 显示信息。

以下是 STAT Hint 的语法：

```
/*+ STAT(TRACING_NUM_LIST) */
```

示例如下：

```
SELECT /*+ STAT(1) */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

TOPK Hint

TOPK Hint 指示服务器设置模糊查询的精度和最小行数。其中参数 PRECSION 的值为整型，取值范围为0~100，表示执行模糊查询时的行数百分比，参数 MINIMUM_ROWS 用来指定最小的返回行数。

以下是 TOPK Hint 的语法：

```
/*+ TOPK(PRECISION MINIMUM_ROWS) */
```

示例如下：

```
SELECT /*+ TOPK(1,10) */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

TRACE_LOG Hint

TRACE_LOG Hint 指示服务器收集跟踪日志（Trace log），收集的跟踪日志（Trace log）在运行 SHOW TRACE 命令时展示。

以下是 TRACE_LOG Hint 的语法：

```
/*+ TRACE_LOG */
```

示例如下：

```
SELECT /*+ TRACE_LOG */ *
 FROM employees e
 WHERE e.department_id = 1001;
```

6.4.7. 数据库对象

6.4.7.1. Schema 对象

Schema 是数据的逻辑结构和 Schema 对象的集合。每个 OceanBase 数据库的使用者都拥有一个 Schema，并且Schema 拥有和使用者相同的名字。

可以使用 SQL 来创建和处理用户数据库对象，并且 Schema 对象有以下类型：

- 约束 Constraints
- 数据库链接 Database links
- 数据库触发器 Database triggers
- 索引 Indexes
- 对象表 Object tables
- 对象类型 Object types
- 对象视图 Object views
- 包 Packages
- 序列 Sequences
- 存储函数 Stored functions
- 存储过程 Stored procedures
- 同义词 Synonyms
- 表 Tables
- 视图 Views

6.4.8. 数据库命名规范

6.4.8.1. 数据库对象命名规范概述

一些数据库对象由用户可以命名的部分和用户必须命名的部分组成。例如，表或视图中的列，索引、表分区和表的子分区，表的完整性约束以及存储在包中的对象（包括程序和存储函数）。本章将讨论：

- 数据库对象命名规则
- Schema 对象命名示例
- Schema 对象命名准则

6.4.8.2. 数据库对象命名规则

数据库对象标识符

每个数据库对象都有名字。在 SQL 语句中您可以使用引用标识符和非引用标识符来命名任何数据库对象：

- 引用标识符：以双引号 `""` 为开头和结尾的标识符。如果使用引用标识符命名 Schema 对象，则在引用该对象时必须使用双引号。
- 非引用标识符：标识符中不包含任何标点符号。

但是，数据库名称、全局数据库名称、数据库链接名称、磁盘组名称和可插入数据库（PDB）始终不区分大小写，并以大写形式存储。如果使用引用标识符为它们命名，那么引号将被忽略。

> 注意
>
> OceanBase 并不建议您使用引用标识符来命名数据库对象。这些引用标识符在 SQL PLUS 中兼容，但是在其他数据库对象管理工具中可能不被识别。

标识符使用规则

以下是一些适用于引用标识符和非引用标识符的规则，除非另有说明，否则以下规则对引用标识符和非引用标识符同时适用。

标识符的长度

标识符的长度为 1~128 字节。

如果标识符包含由句点分隔的多个部分，则每个部分最长都可达 128 字节。每个句点分隔符以及旁边的引号都算作一个字节。例如以下字符串：

```
"schema"."table"."column"
```

示例名称中的三部分 schema、table 和 column 各自最长可以是 128 字节，每个引号和句点都是单字节字符，所以示例中标识符的总长度为 392 字节。

保留字做标识符

非引用标识符不能是 OceanBase SQL 保留字。引用标识符可以是保留字，但不建议这样做。根据您计划用于访问数据库对象的工具不同，名称可能进一步受到其他产品特定的保留字限制。

> 注意
>
> 保留字 `ROWID` 是这个规则的一个例外。不能使用大写单词 `ROWID` 作为列的名称，无论是带引号的还是不带引号的。但是，引用标识符作为列名时，不可以使用全大写单词，但是可以包含一个或多个小写字母（例如，“Rowid”或“Rowid”）。

特殊含义单词做标识符

OceanBase 中的 SQL 语言包含了其他具有特殊含义的单词。这些词包括数据类型、Schema 名、函数名、虚拟系统表 `DUAL` 和关键字（SQL 语句中的大写单词，如 `DIMENSION` 、 `SEGMENT` 、 `ALLOCATE` 、 `DISABLE` 等）。这些词不是保留字，但是，OceanBase 在内部以特定的方式使用它们。因此，如果您使用这些词作为对象和对象部件的名称，SQL 语句可能会难以阅读，并可能导致不可预测的结果。尤其是不要使用 SQL 内置函数的名称作为 Schema 对象和用户自定义函数的名称。

ASCII 字符做标识符

在数据库名称、全局数据库名称和数据库链接名称中您应该使用 ASCII 字符集中的字符，因为这些字符拥有跨不同平台和操作系统的最佳兼容性。多租户容器数据库（CDB）中的通用用户和通用角色的名称只能使用 ASCII 字符集中的字符。

密码中的字符

密码中可以包含多位字节的字符，如汉字或中文标点等。

标识符的开头

非引用标识符必须以数据库字符集中的字母字符开头。引用标识符可以以任何字符开头。

标识符中的符号

非引用标识符只能包含来自数据库字符集的字母数字字符和下划线（_）。但是用于数据库链接名称时可以包含句点（.）和符号 @。引用标识符可以包含任何字符、标点符号和空格。但是，引用标识符和非引用标识符都不能包含双引号或空字符（\0）。

对象名称在名称空间中的限制

在同一个名称空间中，表、视图、私有同义词不能拥有相同的名称。

标识符的大小写

非引用标识符不区分大小写，OceanBase 将他们存储为大写。引用标识符区分大小写，所以通过将名称括在双引号里，在同一命名空间的不同对象可以拥有以下类型的名称：

```
"employees"
"Employees"
"EMPLOYEES"
```

由于非引用标识符不区分大小写，所以 OceanBase 认为以下名称是相同的，他们不能作为同一名称空间下不同对象的名字：

```
employees
EMPLOYEES
"EMPLOYEES"
```

大写标识符

存储或比较大写的标识符时，每个字符的大写形式是通过应用数据库字符集的大写规则来确定的。由对话设置 `NLS_SORT` 所设定的特定语言规则不在考虑的范围内。此行为是将 SQL 函数 `UPPER` 应用于标识符，而不是将函数 `NLS_UPPER` 应用于标识符。

数据库字符集的大写规则被应用到自然语言时可能产生不正确的结果。例如，德国小写字母 `ß`，它在数据库字符集大写规则中并没有对应的大写形式， 所以当含有它的标识符被转换为大写形式时它并不会被转变。

数据库字符集大写规则确保标识符在任何语言配置的会话中的样式是相同的。如果希望标识符在某种自然语言中看起来正确，那么可以通过引号来保留它的小写形式，或者在使用该标识符时使用对应语言上正确的大写形式。

列的名称

同一表或视图中的列不能具有相同的名称。但是，不同表或视图中的列可以具有相同的名称。

程序和函数的名称

如果同一包中的程序或函数所包含的参数的数量和数据类型不同，则它们可以具有相同的名称。在同一个包中使用不同的参数创建具有相同名称的多个过程或函数称为重载过程或函数。

6.4.8.3. Schema 对象命名示例

以下是一些有效的 Schema 对象命名示例：

- `last_name`
- `horse`
- `hr.hire_date`
- `"EVEN THIS & THAT!"`
- `a_very_long_and_valid_name`

这些示例都遵循了 数据库对象命名规则 中列出的规则。

6.4.8.4. Schema 对象命名准则

以下是一些有用的用来命名对象及其部分的准则：

- 使用完整的、描述性的、可发音的名称或众所周知的缩写。
- 使用一致的命名规则。
- 使用相同的名称来描述跨表的相同实体或属性。

在命名对象时，要在简单易用和名称具有描述性的目标之间取得平衡。如果无法平衡，请选择更具描述性的名称，因为数据库中的对象可能会在很长一段时间内被使用，从现在开始十年后的用户可能难以理解名称为 pmdd 的表列是用来存储什么信息的，但是 payment_due_date 却更清楚明了。

使用一致的命名规则可以帮助用户理解每个表在应用程序中扮演的角色。例如，以 fin_ 为开头命名所有属于 FINANCE 应用程序的表的名称。

使用相同的名称描述表中的相同内容。例如，雇员和部门表的部门号列其实存储的是一样的值，那么在雇员表和部门表中这一列均被命名为 department_id。

6.4.9. 数据库对象引用方式

6.4.9.1. 数据库对象引用概述

本章将讨论如何在 SQL 语句的上下文中引用 Schema 对象及其部分。本章包括：

- 引用 Schema 对象
- 引用远程数据库中的对象
- 引用分区表和索引
- 引用对象类型属性和方法
- 创建远程数据库链接

6.4.9.2. 引用 Schema 对象

当在 SQL 语句中引用对象时，OceanBase 会考虑 SQL 语句的上下文并在适当的名称空间中找到该对象。找到对象后，OceanBase 执行 SQL 语句对该对象指定的操作。如果在适当的名称空间中找不到命名的对象，则返回错误。

引用用户的 Schema 中的对象

以下示例说明了 OceanBase 如何解析 SQL 语句中对象的引用。

执行以下语句，向名为 **departments** 的表中添加了一行数据：

```
INSERT INTO departments
VALUES (280, 'ENTERTAINMENT_CLERK', 206, 1700);
```

根据 SQL 语句的上下文，**departments** 在数据库中可能是：

- 一张在您自己 Schema 中的表
- 一个在您自己 Schema 中的视图
- 一个表或视图的专有同义词
- 一个公共同义词

OceanBase 始终会先尝试在您自己的 Schema 中的名称空间里解析被引用对象，然后再考虑该 Schema 之外的名称空间。在此示例中，OceanBase 尝试解析如下：

1. 首先，OceanBase 尝试在您自己的包含了表、视图和专用同义词的 Schema 中的名称空间里去定位对象。如果对象是专用同义词，则 OceanBase 会找到该同义词所代表的对象。该对象可以在您自己的 Schema 中，也可以在其他 Schema 中或者在另一个数据库中。该对象也可以是其他的同义词，在这种情况下，OceanBase 会找到同义词所代表的对象。
2. 如果对象在名称空间中被找到，则 OceanBase 尝试对该对象执行 SQL 语句。在此示例中，OceanBase 尝试将一行数据添加到 **departments** 中。如果此对象的类型不是该 SQL 语句所需要的类型，则 OceanBase 返回错误。在此示例中，**departments** 必须是表、视图或者可以是可以解析为表或视图的专用同义词。如果 **departments** 是序列，则 OceanBase 返回错误。
3. 如果该对象到目前为止没有在任何名称空间中被搜索到，则 OceanBase 将搜索包含公共同义词的名称空间。如果对象在包含公共同义词的名称空间中，则 OceanBase 尝试对该对象执行该语句。如果此对象的类型不是该 SQL 语句所需要的类型，则 OceanBase 返回错误，例如在此示例中，如果 **departments** 是一个代表了序列的公共同义词，那么 OceanBase 将返回错误。

如果公共同义词包含任何依赖表或用户定义类型，则不能在与依赖对象相同的 Schema 中创建与同义词同名的对象。

反之，如果同义词没有任何依赖表或用户定义的类型，则可以在与依赖对象相同的 Schema 中创建具有相同名称的对象。OceanBase 会使所有依赖对象无效，并在下一次访问它们时重新验证它们。

引用其他 Schema 中的对象

要引用您的 Schema 之外的 Schema 中的对象，需要在对象名称前添加 Schema 名称：

```
schema.object
```

例如，以下示例展示了在名为 hr 的 Schema 中删除表 employees：

```
DROP TABLE hr.employees;
```

6.4.9.3. 引用远程数据库中的对象

6.4.9.3.1. 创建数据库链接

要引用除本地数据库以外的其他数据库中的对象，需要在对象名称后加上指向该数据库的数据库链接的名称。数据库链接是一个 Schema 对象，它使 OceanBase 连接到远程数据库以访问里面的对象。

可以使用语句 CREATE DATABASE LINK 创建数据库链接，使用该语句时需要指定以下数据库链接的信息：

- 租户名、用户名和密码。
- 网络地址、端口号。

OceanBase 将上述信息存储在数据字典中。

创建链接限制

1. 只支持创建 OceanBase 中的 Oracle 模式租户连接到另一个 Oracle 模式租户的数据库链接，OceanBase 目前不支持创建连接到外部 Oracle 数据库的数据库链接。
2. 一个租户可以反复创建、删除数据库链接，但最多只能同时存在 15 个数据库链接。

数据库连接串格式

```
user_name@tenant_name IDENTIFIED BY password HOST 'ip:port'
```

- user_name：用户名。
- tenant_name：租户名。
- password：密码。
- ip：网络地址。
- port：端口号。

以下是创建数据库链接的示例：

```
CREATE DATABASE LINK my_link CONNECT TO
root@oracle  IDENTIFIED BY abcdef HOST '192.168.0.0:1521';
```

6.4.9.3.2. 引用数据库链接

当您发出包含数据库链接的 SQL 语句时，需要使用存储在数据字典中的完整数据库链接名，更多关于数据库链接名称的信息请查看文档 创建数据库链接。

OceanBase 在连接远程数据库时，将使用数据库连接字符串访问远程数据库。使用数据库字符串、用户名和密码后连接成功，则 OceanBase 使用文档 引用 Schema 对象 中的规则来访问远程数据库上的指定对象，但请注意相关限制：

1. 只能执行只读语句。
2. 只能访问表对象，不支持访问其它对象，如视图、序列等。
3. 访问表对象时必须显式指定数据库名，如：`test.t1@my_link`。
4. 不支持部分计划和算子：
   - 不能在远程数据库执行 `RESCAN` 操作。
   - 不能在远程数据库执行 `NESTED LOOP JOIN`、`SEMI JOIN`、`ANTI JOIN`、`SUBPLAN FILTER` 等算子。
   - 如遇以上问题请尝试运行 `EXPLAIN PLAN` 语句查看原始计划和发送到远端集群执行的 SQL 语句，并通过使用 Hint 调整计划。

6.4.9.4. 引用分区表和索引

表和索引可以分区。进行分区时，这些 Schema 对象由许多称为分区的部分组成，所有这些部分都具有相同的逻辑属性。例如，表中的所有分区共享相同的列和约束定义，而索引中的所有分区共享相同的索引列。

分区扩展名和子分区扩展名使用户可以执行某些分区级和子分区级操作，例如，仅在一个分区或子分区上删除其中的所有行。没有扩展名称时，此类操作将要求您使用判断语句定义范围（`WHERE` 子句）。对于范围分区表和列表分区表，尝试用判断语句描述分区级操作可能会很麻烦，尤其是当范围分区键使用多个列时。对于哈希分区和子分区，使用判断语句更加困难，因为这些分区和子分区是基于系统定义的哈希函数。

分区扩展名使您可以像使用表一样使用分区。此方法的一个优点（对范围分区的表最有用）是，您可以通过对其他用户或角色授予（或撤消）这些视图的特权来构建分区级别的访问控制机制。要将分区用作表，需要通过从单个分区中选择数据来创建视图，然后将该视图用作表。

语法

当在 SQL 语句的语法或轨道图中出现 `partition_extended_name` 或者 `subpartition_extended_name` 元素时，用户可以通过这两个元素指定分区扩展表名和子分区扩展表名。`partition_extended_name` 的语法：

```
PARTITION partition
|
PARTITION FOR ( partition_key_value [, partition_key_value]... )
```

`subpartition_extended_name` 的语法：

```
SUBPARTITION subpartition
|
SUBPARTITION FOR ( subpartition_key_value [, subpartition_key_value]... )
```

DML 语句 INSERT 、 UPDATE 、 DELETE 和 ANALYZE 语句需要在分区或子分区名称的周围加上括号。这个小的区别体现在 partition_extension_clause 元素中。在 partition_extended_name ， subpartition_extended_name 和 partition_ extension_clause 中，可以使用 PARTITION FOR 和 SUBPARTITION FOR 子句在不使用名称的情况下引用分区。它们对任何类型的分区均有效，尤其是间隔分区。将数据插入表中时，间隔分区会根据需要被自动创建。

对于上述元素中各自的 partition_key_value 或 subpartition_key_value 部分，它们为每个分区键列指定一个值。对于多列分区键，需要为每个分区键指定一个值。对于复合分区，对每个分区键指定一个值后，需要继续为每个子分区键指定一个值。所有分区键值均以逗号分隔。对于间隔分区，您只能指定一个 partition_key_value ，并且它必须是有效的 NUMBER 数据类型或日期时间数据类型的值。用户的 SQL 语句将在包含用户指定了值的分区或子分区上运行。

扩展名称的限制

使用分区扩展表名和子分区扩展表名时有以下限制：

- 名称中没有远程表：扩展分区表名和子分区扩展表名不能包含数据库链接或能转换为具有数据库链接的表的同义词。要使用远程分区和远程子分区，需要在远程站点上使用扩展表名语法创建一个视图，然后引用该远程视图。
- 名称中没有同义词：必须使用基表指定分区或子分区扩展名，不能使用同义词，视图或任何其他对象。
- 在 PARTITION FOR 和 SUBPARTITION FOR 子句中，不能指定关键字 DEFAULT 、 MAXVALUE 或绑定变量为 partition_key_value 或 subpartition_key_value 的值。
- 在 PARTITION 和 SUBPARTITION 子句中，不能为分区或子分区名称指定绑定变量。

示例

在以下示例中，sales 是具有分区 sales_q1_2000 的分区表。以下语句创建了单个分区 sales_q1_2000 的视图，然后像使用表一样使用它。本示例从分区中删除了一些行：

```
/*为分区 sales_q1_2000 创建视图 Q1_2000_sales*/
CREATE VIEW Q1_2000_sales
AS
SELECT *
FROM sales PARTITION (SALES_Q1_2000);
/*删除视图 Q1_2000_sales 中符合条件 amount_sold < 0 的值*/
DELETE FROM Q1_2000_sales
WHERE amount_sold < 0;
```

6.4.9.5. 引用对象类型的属性和方法

要在 SQL 语句中引用对象类型属性或方法，必须使用表别名完全限定该引用。以下示例中样本 Schema ob 包含类 cust_address_typ 和表 customers，customers 拥有一个 cust_address_typ 类型的列 cust_address：

```
CREATE TYPE cust_address_typ
 OID '82A4AF6A4CD1656DE034080020E0EE3D'
AS OBJECT
  (street_address   VARCHAR2(40),
   postal_code     VARCHAR2(10),
   city          VARCHAR2(30),
   state_province   VARCHAR2(10),
   country_id      CHAR(2));
/
CREATE TABLE customers
 (customer_id      NUMBER(6),
  cust_first_name   VARCHAR2(20) CONSTRAINT cust_fname_nn NOT NULL,
  cust_last_name    VARCHAR2(20) CONSTRAINT cust_lname_nn NOT NULL,
  cust_address     cust_address_typ,
...
```

在 SQL 语句中，对 postal_code 属性的引用必须使用表别名进行完全限定，如下所示：

```
SELECT c.cust_address.postal_code
FROM customers c;

UPDATE customers c
SET c.cust_address.postal_code = '610000'
WHERE c.cust_address.city = 'chengdu'
AND c.cust_address.state_province = 'SICHUAN';
```

要引用不接受参数的成员方法，必须提供空括号。例如，样本 Schema ob 包含一个基于 catalog_typ 的对象表 category_tab，该表包含成员函数 `getCatalogName` 。为了在 SQL 语句中调用此方法，必须提供空括号，如下所示：

```
SELECT TREAT(VALUE(c) AS catalog_typ).getCatalogName() "Catalog Type"
FROM categories_tab c
WHERE category_id = 10;
```

返回结果：

```
+----------------+
| Catalog Type   |
+----------------+
| online catalog |
+----------------+
```

6.5. 运算符

6.5.1. 运算符概述

运算符一般用于连接运算数或参数之类的单个数据项且返回结果。从语法上讲，运算符出现在运算数之前、之后或两个运算数之间均可。一般用特殊字符或关键字表示运算符，例如，除法运算符用斜杠（/）表示。本章讨论非逻辑（非布尔）运算符，这些运算符本身不能用作查询或子查询的 `WHERE` 或 `HAVING` 条件。

本章将详细介绍以下运算符：

- 算术运算符
- 串联运算符
- 层次查询运算符
- 集合运算符

一元和二元运算符

运算符可分为两大类：

- 一元运算符：一元运算符仅对一个运算数进行运算。一元运算符常用格式：

```
运算符 运算数
```

- 二元运算符：二元运算符顾名思义是对两个运算数进行运算。二元运算符常用格式：

```
运算数1 运算符 运算数2
```

其他具有特殊格式的运算符可接受两个以上的运算数。如果为运算符提供了空运算数（NULL）， 则结果始终为空。唯一不遵循此规则的是串联（||）运算符。

运算符优先级

优先级会影响 OceanBase 数据库在同一表达式中计算不同运算符的顺序。在计算包含多个运算符的表达式时，OceanBase 会先计算优先级更高的运算符，然后再计算优先级较低的运算符，优先级相等时，在表达式中从左到右对相等的运算符求值。

下表列出了 SQL 运算符中从高到低的优先级。同一行上列出的运算符具有相同的优先级。

运算符	运算方式
+、-（作为一元运算符）、PRIOR、CONNECT_BY_ROOT	正号、负号、层次结构中的位置。
*、/	乘法、除法。
+、-（作为二元运算符）、||	加、减、串联。

运算符	运算方式
条件表达式在数据库评估完运算符后再进行评估	详情请参阅文档 SQL 条件概述。

在以下示例中，由于乘法的优先级高于加法，因此数据库首先将 2 乘以 3，然后再将结果加到 1。

```
1+2*3
```

您也可以在表达式中使用括号来限制运算符优先级。OceanBase 将先对括号内的表达式求值，再对括号外部的表达式求值。

SQL 还支持集合运算符（ UNION 、 UNION ALL 、 INTERSECT 和 MINUS ），集合运算符会合并查询返回的行集，而不是单个数据项。所有集合运算符都具有相同的优先级。

6.5.2. 算术运算符

算术运算符用来对一个或两个参数进行求反、加、减、乘和除等操作。其中一些算术运算符还可用于计算日期时间和间隔值。算术运算符的参数必须解析为数字数据类型或任何可以被数据库直接转换为数字数据类型的数据类型。

一元算术运算符返回的数据类型与参数的数据类型相同。对于二进制算术运算符，OceanBase 先确定表达式中数值优先级最高的参数，再将其余参数转换为该参数的数据类型。

下表列出了算术运算符：

运算符	说明
+、-	用于表示正、负时，它们是一元运算符。
+、-	用于表示加、减时，它们是二进制运算符。
*、/	二进制运算符，用于表示乘、除。

算术表达式中不能使用两个连续的负号（--）来表示双重求反或减去一个负值，因为字符 -- 在 SQL 语句中用于指定注释。可以用空格或括号分隔连续的减号。有关 SQL 语句中的注释，详细信息请参阅章节 注释。

以下示例展示了运算符 +、- 用于表示正、负的 SQL 查询：

```
SELECT * FROM order_items WHERE quantity = -1 ORDER BY order_id, line_item_id, product_id;
SELECT * FROM employees WHERE -salary < 0 ORDER BY employee_id;
```

以下示例展示了运算符 +、- 用于表示加、减的 SQL 查询：

```
SELECT hire_date FROM employees WHERE SYSDATE - hire_date > 365 ORDER BY hire_date;
```

以下示例展示了 *、/ 运算符表示乘、除的 SQL 查询：

```
UPDATE employees SET salary = salary * 1.1;
UPDATE employees SET salary = salary / 2;
```

6.5.3. 串联运算符

串联运算符 || 用于连接字符串和 CLOB 数据类型的数据。

连接两个字符串后会得到另一个字符串。如果两个字符串的数据类型均为 CHAR，则返回结果的数据类型也为 CHAR，并且限制为 2000 个字符。如果被合并的任一字符串的数据类型为 VARCHAR2，则返回结果的数据类型也为 VARCHAR2，并且限制为 4000 个字符。如果串联运算符两边的参数中任何一个是 CLOB 数据类型的值，则返回结果的数据类型为临时 CLOB。无论两边参数的数据类型是什么，字符串中的尾部空格都是通过串联保存的。

尽管 OceanBase 将长度为 0 的字符串视为空值（NULL），但是将一个带有值的操作数和另一个 0 长度字符串连接得到的结果是一个带值的操作数，因此 NULL 只能由两个值为 NULL 的字符串串联产生。但是，在将来的 OceanBase 数据库版本中，情况可能不会继续如此。要连接可能为空的表达式，请使用 NVL 函数将表达式直接转换为长度为 0 的字符串。

此示例创建了一个同时包含 CHAR 和 VARCHAR2 类型列的表，然后插入带有和不带有尾部空格的值，并将它们连接起来。

```
CREATE TABLE tab1 (col1 VARCHAR2(6), col2 CHAR(10), col3 VARCHAR2(10), col4 CHAR(6));
INSERT INTO tab1 (col1, col2, col3, col4) VALUES ('abc', 'def  ', 'ghi  ', 'jkl');
SELECT col1 || col2 || col3 || col4 "Concatenation" FROM tab1;
```

返回结果：

```
+----------------------------+
| Concatenation              |
+----------------------------+
| abcdef    ghi  jkl  |
+----------------------------+
```

6.5.4. 层次查询运算符

PRIOR 和 CONNECT_BY_ROOT 运算符仅在 层次查询 中有效。

PRIOR 运算符

在层次查询中，CONNECT BY 条件中应该包含至少一个由 PRIOR 运算符限定的表达式。PRIOR 运算符的结果，通过使用当前行的父行计算紧随后的表达式来得到。

PRIOR 最常用在列的等值比较场景中（ PRIOR 可以在比较符的任意一侧）。理论上，在 CONNECT BY 子句中也可以使用等号（=）以外的其他运算符。但是，由这些其他运算符创建的条件可能会导致循环，在这种情况下，OceanBase 在运行时会检测循环并返回错误。

CONNECT_BY_ROOT 运算符

CONNECT_BY_ROOT 是一元运算符，仅在层次查询中有效。 CONNECT_BY_ROOT 的参数可以是列和大部分的表达式， CONNECT_BY_ROOT 运算符的结果，等于使用根行的数据计算参数的值。

您不能在 START WITH 条件或 CONNECT BY 条件中指定此运算符。

6.5.5. 集合运算符

集合运算符将两个查询结果合并为一个结果。包含集合运算符的查询称为复合查询。

运算符	返回值
UNION	返回任意查询选择的所有不同行。
UNION ALL	返回任意查询选择的所有行，并包括所有重复项。
INTERSECT	返回两个查询都选择的所有不同行。
MINUS	返回第一个查询选择的所有不同行，但其中不包括出现在第二个查询中的行。

关于集合运算符的更多信息和示例，请参阅查询和子查询章节中关于 UNION ALL，INTERSECT，MINUS 运算符的介绍。

6.5.6. 排序规则

排序规则（即 COLLATE 运算符）用于指定字符序，可以用于重新设定字符类型的字符序或为单个表达式指定字符序。

COLLATE 是后缀一元运算符，与其它一元运算符优先级相同，但是在所有前缀一元运算符之后执行。COLLATE 运算符可以应用于 VARCHAR2，CHAR，LONG，NVARCHAR 或 NCHAR 类型的表达式。

COLLATE 运算符后跟一个参数 collation_name，表示需要设置或使用的字符序，字符序名称包含空格时必须在名称外加双引号。

举例如下：

```
obclient> create table t1(c1 varchar(20));
obclient> insert into t1 values('a');
obclient> insert into t1 values('b');
obclient> insert into t1 values(1);
obclient> select * from t1 order by c1 collate utf8mb4_general_ci;
+------+
| C1   |
+------+
| 1    |
| a    |
| b    |
+------+
3 rows in set (0.00 sec)
```

目前支持如下排序规则：

```
obclient> show collation;
+--------------------+---------+-----+---------+----------+---------+
| COLLATION          | CHARSET | ID  | DEFAULT | COMPILED | SORTLEN |
+--------------------+---------+-----+---------+----------+---------+
| utf8mb4_general_ci | utf8mb4 |  45 | Yes     | Yes      |       1 |
| utf8mb4_bin        | utf8mb4 |  46 | NULL    | Yes      |       1 |
| binary             | binary  |  63 | Yes     | Yes      |       1 |
| gbk_chinese_ci     | gbk     |  28 | Yes     | Yes      |       1 |
| gbk_bin            | gbk     |  87 | NULL    | Yes      |       1 |
| utf16_general_ci   | utf16   |  54 | Yes     | Yes      |       1 |
| utf16_bin          | utf16   |  55 | NULL    | Yes      |       1 |
| utf8mb4_unicode_ci | utf8mb4 | 224 | NULL    | Yes      |       1 |
| utf16_unicode_ci   | utf16   | 101 | NULL    | Yes      |       1 |
| gb18030_chinese_ci | gb18030 | 248 | Yes     | Yes      |       1 |
| gb18030_bin        | gb18030 | 249 | NULL    | Yes      |       1 |
+--------------------+---------+-----+---------+----------+---------+
11 rows in set (0.01 sec)
```

6.6. 函数

6.6.1. 函数概述

函数与运算符相似，即给定一些数据元素作为参数的输入并返回结果。然而函数在入参形式上与运算符有比较大的区别。函数允许包含的参数数量不定，一个函数里可以有一到两个，甚至更多的参数。

```
函数名（参数，参数，...）
```

没有任何参数的函数类似于 伪列。但是，伪列通常为结果集中的每一行返回不同的值，而没有任何变量的函数通常为每一行返回相同的值。

关于函数

OceanBase 内嵌的函数可以直接在 SQL 语句中使用。每个函数的入参传入值均有期望的数据类型，如果传入的数据类型不是期望的数据类型，则 OceanBase 会在实际执行 SQL 函数之前尝试将参数传入的数值转换为期望的数据类型。

函数中的空值

对于绝大多数的函数当入参为空值 NULL 的时候，其返回的结果也为 NULL。这种情况下，您可以使用 `NVL` 函数返回一个非空值。例如，一张记录佣金的表的佣金列 **commission_pct** 为空值 NULL，则表达式 `NVL(commission_pct，0)` 返回 0；如果 **commission_pct** 的值不为 NULL，则返回实际的佣金值。

函数分类与列表

在如下的函数分类列表中，每一类函数的参数和最终的函数返回值都有其特定的数据类型。

> **注意**
>
> 在 SQL 语句中对 `LOB` 列使用函数时，OceanBase 数据库将在 SQL 和 PL/SQL 处理期间创建临时 `LOB` 列，并有一定的使用限制，详情信息请参考文档 与 Oracle 兼容性对比。

本章中函数分成了两大类：

- **单行函数**：包括 数字函数、返回字符串的字符串函数、返回数字的字符串函数、 时间日期函数、通用比较函数、转换函数、编码解码函数和空值相关函数。
- **统计函数**：包括聚合函数和分析函数。

单行函数对于被查询的表或者视图每一行均返回一个结果值，这些函数可以使用在 SQL 语句的 `SELECT` 、 `WHERE` 、 `START WITH` 、 `CONNECT BY` 、 `HAVING` 等子句当中。

分析函数与聚合函数，都是对行集组（一组行的集合）进行聚合计算，不同的是，聚合函数每组只能返回一个值（一行），而分析函数每组可以返回多个值（多行）。行集组又称为窗口（Window）。聚合函数通常和 `SELECT` 语句中的 `GROUP BY` 子句一起使用，使用时数据库将查询表或视图的行分为几组，并将聚合函数应用于每组行，同时为每组返回一个结果行。

使用分析函数时需要用特殊的关键字 `OVER` 来指定窗口。更多关于窗后函数的信息，请参阅文档 窗口函数说明。

数字函数

数字函数的变量输入与函数输出结果均为数字类型，绝大部分的数字函数的返回值的数据类型为 `NUMBER` ，可以精确到小数点后 38 位。一些高等代数相关函数 `COS` 、 `COSH` 、 `EXP` 、 `LN` 、 `LOG` 、 `SIN` 、 `SINH` 、 `SQRT` 、 `TAN` 、 `TANH` 等函数的结果精确到小数点后 36 位，其他代数相关函数 `ACOS` 、 `ASIN` 、 `ATAN` 和 `ATAN2` 。其结果返回值精确到小数点后 30 位。

函数分类	函数子分类	函数名	功能描述
单行函数	数字函数	ABS	返回指定数值表达式的绝对值（正值）的数学函数。
单行函数	数字函数	ACOS	返回以弧度表示的角，其余弦为指定的 `NUMBER` 表达式，也称为反余弦。
单行函数	数字函数	ASIN	OceanBase 暂不支持。
单行函数	数字函数	ATAN	OceanBase 暂不支持。
单行函数	数字函数	ATAN2	OceanBase 暂不支持。
单行函数	数字函数	BITAND	运算符按位进行“与”操作。输入和输出类型均为 `INT` 整型，且类型一致。
单行函数	数字函数	CEIL	返回值大于等于数值 `numeric_expression` 的最小整数。
单行函数	数字函数	COS	OceanBase 暂不支持。
单行函数	数字函数	COSH	OceanBase 暂不支持。
单行函数	数字函数	EXP	返回 e 的 numeric_expression 次幂。
单行函数	数字函数	FLOOR	返回小于等于数值 `numeric_expression` 的最大整数。

函数分类	函数子分类	函数名	功能描述
单行函数	数字函数	LN	返回以 e 为底的 `numeric_expression` 的对数。
单行函数	数字函数	LOG	返回以 `x` 为底的 `y` 的对数。
单行函数	数字函数	MOD	返回 `x` 除以 `y` 的余数。
单行函数	数字函数	POWER	返回 `x` 的 `y` 次幂。
单行函数	数字函数	REMAINDER	返回 `x` 除以 `y` 的余数。
单行函数	数字函数	ROUND	返回 `numberic` 四舍五入后的值。
单行函数	数字函数	SIGN	返回数字 `n` 的符号，大于 0 返回 1，小于 0 返回 -1，等于 0 返回 0。
单行函数	数字函数	SIN	OceanBase 暂不支持。
单行函数	数字函数	SINH	OceanBase 暂不支持。
单行函数	数字函数	SQRT	返回 `n` 的平方根。
单行函数	数字函数	TAN	OceanBase 暂不支持。
单行函数	数字函数	TANH	OceanBase 暂不支持。

函数分类	函数子分类	函数名	功能描述
单行函数	数字函数	TRUNC	返回 `numeric` 按精度 `precision` 截取后的值。
单行函数	数字函数	WIDTH_BUCKET	OceanBase 暂不支持。

返回字符串的字符串函数

函数的返回值的最大长度受数据类型的影响，比如：函数的返回值的数据类型是 `VARCHAR2`，但是返回值实际的大小超过了 `VARCHAR2` 数据类型的最大限制，此时 OceanBase 数据库会对结果进行截断处理并返回，但是在客户端上并不会显示提示。

> **注意**
>
> 如果返回值的数据类型是 `CLOB`，当返回值长度超过了最大限制时，OceanBase 不会返回数据且显示错误提示。

函数分类	函数子分类	函数名	功能描述
单行函数	返回字符串的字符串函数	CHR	将 `n` 转换为等价的一个或多个字符返回，且返回值与当前系统的字符集相关。
单行函数	返回字符串的字符串函数	CONCAT	连接两个字符串。
单行函数	返回字符串的字符串函数	INITCAP	返回字符串并将字符串中每个单词的首字母大写，其他字母小写。
单行函数	返回字符串的字符串函数	LOWER	将字符串全部转为小写。

函数分类	函数子分类	函数名	功能描述
单行函数	返回字符串的字符串函数	LPAD	在字符串 `c1` 的左边用字符串 `c2` 填充，直到长度为 `n` 时为止。
单行函数	返回字符串的字符串函数	LTRIM	删除左边出现的字符串。
单行函数	返回字符串的字符串函数	REGEXP_REPLACE	用于正则表达式替换。
单行函数	返回字符串的字符串函数	REGEXP_SUBSTR	OceanBase 暂不支持。
单行函数	返回字符串的字符串函数	REPLACE	将字符表达式值中，部分相同字符串，替换成新的字符串。
单行函数	返回字符串的字符串函数	RPAD	在字符串 `c1` 的右边用字符串 `c2` 填充，直到长度为 `n` 时为止。
单行函数	返回字符串的字符串函数	RTRIM	删除右边出现的字符串，此函数对于格式化查询的输出非常有用。
单行函数	返回字符串的字符串函数	SUBSTR	截取子字符串。其中多字节符（汉字、全角符等）按 1 个字符计算。
单行函数	返回字符串的字符串函数	TRANSLATE	将字符表达式值中，指定字符替换为新字符。多字节符（汉字、全角符等），按 1 个字符计算。
单行函数	返回字符串的字符串函数	TRIM	删除一个字符串的开头或结尾（或两者）的字符。
单行函数	返回字符串的字符串函数	UPPER	将字符串全部转为大写。

返回数字的字符串函数

函数分类	函数子分类	函数名	功能描述
单行函数	返回数字的字符串函数	ASCII	返回字符表达式最左端字符的 ASCII 码值。
单行函数	返回数字的字符串函数	INSTR	在一个字符串中搜索指定的字符，返回发现指定的字符的位置。
单行函数	返回数字的字符串函数	LENGTH	返回字符串的长度。
单行函数	返回数字的字符串函数	REGEXP_COUNT	OceanBase 暂不支持。
单行函数	返回数字的字符串函数	REGEXP_INSTR	OceanBase 暂不支持。

时间日期函数

时间日期函数支持的入参数据类型有三类：日期时间（`DATE`），时间戳（`TIMESTAMP`、`TIMESTAMP WITH TIME ZONE`、`TIMESTAMP WITH LOCAL TIME ZONE`），以及间隔（`INTERVAL DAY TO SECOND`、`INTERVAL YEAR TO MONTH`）

仅支持 `DATE` 数据类型入参的函数列表：`ADD_MONTHS`、`CURRENT_DATE, LAST_DAY`、`NEW_TIME` 和 `NEXT_DAY`。

如果对于以上函数，您尝试给入的是 `TIMESTAMP` 类型的数据，OceanBase 内部会先进行隐式数据类型转换后带入函数进行运算，并返回 `DATE` 类型的返回值。

> **注意**
>
> `MONTHS_BETWEEN` 返回值为 1 个数字。
>
> `ROUND` 和 `TRUNC` 并不能进行隐式转换，必须传输 `DATE` 类型的数值，否则会报错。

剩余的函数对于三种参数数据类型均可以支持，且返回和入参同样的数据类型。

函数分类	函数子分类	函数名	功能描述

函数分类	函数子分类	函数名	功能描述
单行函数	时间日期函数	ADD_MONTHS	返回在日期 `date` 基础上 `n` 个月后的日期值，如果 `n` 的值为负数则返回日期 `date` 基础上 `n` 个月前的日期值（`date` 减去 `n` 个月）。
单行函数	时间日期函数	CURRENT_DATE	返回当前会话时区中的当前日期。
单行函数	时间日期函数	CURRENT_TIMESTAMP	返回 `TIMESTAMP WITH TIME ZONE` 数据类型的当前会话时区中的当前日期，返回值中包含当前的时区信息。
单行函数	时间日期函数	DBTIMEZONE	返回当前数据库实例的时区，在 OceanBase 中数据库时区恒为+00:00，且不支持修改。
单行函数	时间日期函数	EXTRACT (datetime)	从指定的时间字段或表达式中抽取年、月、日、时、分、秒等元素。
单行函数	时间日期函数	FROM_TZ	将一个 `TIMSTAMP` 数据类型的值和时区信息拼成一个 `TIMESTAMP WITH TIME ZONE` 数据类型的时间值。
单行函数	时间日期函数	LAST_DAY	返回日期 `date` 所在月份的最后一天的日期。

函数分类	函数子分类	函数名	功能描述
单行函数	时间日期函数	LOCALTIMESTAMP	返回当前会话时区中的当前日期，返回 `TIMESTAMP` 数据类型的值。
单行函数	时间日期函数	MONTHS_BETWEEN	返回返回参数 `date1` 到 `date2` 之间的月数。
单行函数	时间日期函数	NEW_TIME	OceanBase 暂不支持。
单行函数	时间日期函数	NEXT_DAY	返回日期 `d1` 的下一周中 `c1`（星期值）所在的日期值。
单行函数	时间日期函数	NUMTODSINTERVAL	把参数 `n` 转为以参数 `interval_unit` 为单位的 `INTERVAL DAY TO SECOND` 数据类型的值。
单行函数	时间日期函数	NUMTOYMINTERVAL	把参数 `n` 转为以 `interval_unit` 为单位的 `INTERVAL YEAR TO MONTH` 数据类型的值。
单行函数	时间日期函数	ROUND (date)	返回以参数 `fmt` 为单位距离的离指定日期 `date` 最近的日期时间值。
单行函数	时间日期函数	SESSIONTIMEZONE	返回当前会话时区。
单行函数	时间日期函数	SYS_EXTRACT_UTC	返回与指定时间相对应的的标准 UTC 时间。
单行函数	时间日期函数	SYSDATE	返回当前日期。

函数分类	函数子分类	函数名	功能描述
单行函数	时间日期函数	SYSTIMESTAMP	返回系统当前日期，返回值的秒的小数位包含 6 位精度，且包含当前时区信息。
单行函数	时间日期函数	TO_CHAR (datetime)	将 DATE 、TIMESTAMP 、TIMESTAMP WITH TIME ZONE 、TIMESTAMP WITH LOCAL TIME ZONE 、INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 等数据类型的值按照参数 fmt 指定的格式转换为 VARCHAR2 数据类型的值。
单行函数	时间日期函数	TO_DSINTERVAL	将一个 CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串转换为一个 INTERVAL DAY TO SECOND 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。
单行函数	时间日期函数	TO_TIMESTAMP	将字符串转换为 TIMESTAMP 数据类型。
单行函数	时间日期函数	TO_TIMESTAMP_TZ	将字符串转换为 TIMESTAMP WITH TIME ZONE 数据类型，包含时区信息。

函数分类	函数子分类	函数名	功能描述
单行函数	时间日期函数	TO_YMINTERVAL	将一个 CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串转换为一个 INTERVAL YEAR TO MONTH 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。
单行函数	时间日期函数	TRUNC (date)	返回以参数 fmt 为单位距离的离指定日期 date 最近的日期时间值，并且返回的日期值在 date 之前。
单行函数	时间日期函数	TZ_OFFSET	返回时区 n 的时区偏移量。时区偏移量是指与格林尼治标准时间 GMT 的差（小时和分钟）。

通用比较函数

可以通过本类别函数快速的在集合中寻找到最大值和最小值。

函数分类	函数子分类	函数名	功能描述
单行函数	通用比较函数	GREATEST	返回一个或多个表达式列表中的最大值。
单行函数	通用比较函数	LEAST	返回一个或多个表达式列表中的最小值。

转换函数

可以通过本类型的函数将原本的数据类型转换为另外一种数据类型。

函数分类	函数子分类	函数名	功能描述
函数分类	函数子分类	函数名	功能描述
单行函数	转换函数	ASCIISTR	OceanBase 暂不支持。
单行函数	转换函数	BIN_TO_NUM	OceanBase 暂不支持。
单行函数	转换函数	CHARTOROWID	OceanBase 暂不支持。
单行函数	转换函数	HEXTORAW	将 CHAR 、 VARCHAR2 、 NCHAR 或 NVARCHAR2 数据类型中包含十六进制数字的字符转换为 RAW 数据类型。
单行函数	转换函数	RAWTOHEX	将二进制数转换为一个相应的十六进制表示的字符串。
单行函数	转换函数	TO_BINARY_DOUBLE	返回一个双精度的 64 位浮点数.
单行函数	转换函数	TO_BINARY_FLOAT	返回一个单精度的 32 位浮点数。
单行函数	转换函数	TO_CHAR (character)	将 NCHAR 、 NVARCHAR2 或 CLOB 数据转换为数据库字符集。

函数分类	函数子分类	函数名	功能描述
单行函数	转换函数	TO_CHAR (datetime)	将 DATE 、TIMESTAMP 、TIMESTAMP WITH TIME ZONE 、TIMESTAMP WITH LOCAL TIME ZONE 、INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 等数据类型的值按照参数 fmt 指定的格式转换为 VARCHAR2 数据类型的值。
单行函数	转换函数	TO_CHAR (number)	将 NUMBER 、BINARY_FLOAT 或 BINARY_DOUBLE 类型的数值 n 按照指定数值格式 fmt 转换为 varchar2 数据类型的值。
单行函数	转换函数	TO_DATE	将 CHAR 、VARCHAR 、NCHAR 或 NVARCHAR2 数据类型的字符转换为日期数据类型的值。

函数分类	函数子分类	函数名	功能描述
单行函数	转换函数	TO_DSINTERVAL	将一个 CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串转换为一个 INTERVAL DAY TO SECOND 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。
单行函数	转换函数	TO_NUMBER	将 expr 转换为数值数据类型的值。
单行函数	转换函数	TO_TIMESTAMP	将字符串转换为 TIMESTAMP 数据类型。
单行函数	转换函数	TO_TIMESTAMP_TZ	将字符串转换为 TIMESTAMP WITH TIME ZONE 数据类型，包含时区信息。
单行函数	转换函数	TO_YMINTERVAL	将一个 CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串转换为一个 INTERVAL YEAR TO MONTH 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。

编码解码函数

可以通过本类型的函数在 OceanBase 数据库中实现数据的编码解密需求。

函数分类	函数子分类	函数名	功能描述
单行函数	编码解码函数	DECODE	会根据条件返回相应值。
单行函数	编码解码函数	ORA_HASH	获取对应表达式的 HASH 值。
单行函数	编码解码函数	VSIZE	返回 `X` 的字节大小数。

空值相关函数

函数分类	函数子分类	函数名	功能描述
单行函数	空值相关函数	COALESCE	返回参数列表中第一个非空表达式，必须指定最少两个参数。
单行函数	空值相关函数	LNNVL	判断条件中的一个或者两个操作数是否为 `NULL` 。
单行函数	空值相关函数	NULLIF	OceanBase 暂不支持。
单行函数	空值相关函数	NVL	从两个表达式返回一个非 NULL 值。如果 `expr1` 与 `expr2` 的结果皆为 NULL 值，则 NVL 函数返回 NULL。

函数分类	函数子分类	函数名	功能描述
单行函数	空值相关函数	NVL2	根据表达式是否为空，返回不同的值。如果 `expr1` 不为空，则返回 `expr2` 的值，如果 `expr1` 为空，则返回 `expr3` 的值。`expr2` 和 `expr3` 类型不同的话，`expr3` 会转换为 `expr1` 的类型。

环境相关函数

本分类的函数主要提供会话或者租户实例相关的环境信息。

函数分类	函数子分类	函数名	功能描述
单行函数	环境相关函数	SYS_CONTEXT	OceanBase 暂不支持。
单行函数	环境相关函数	UID	OceanBase 暂不支持。
单行函数	环境相关函数	USER	OceanBase 暂不支持。

聚合函数

函数分类	函数子分类	函数名	功能描述
统计函数	聚合函数	AVG	返回数值列的平均值。
统计函数	聚合函数	COUNT	用于查询参数 `expr` 的行数。
统计函数	聚合函数	SUM	返回参数中指定列的和。

函数分类	函数子分类	函数名	功能描述
统计函数	聚合函数	GROUPING	OceanBase 暂不支持。
统计函数	聚合函数	MAX	返回参数中指定的列中的最大值。
统计函数	聚合函数	MIN	返回参数中指定列的最小值。
统计函数	聚合函数	LISTAGG	用于列转行，`LISTAGG` 对 `ORDER BY` 子句中指定的每个组内的数据进行排序，然后合并度量列的值。
统计函数	聚合函数	ROLLUP	在数据统计和报表生成过程中，它可以为每个分组返回一个小计，同时为所有分组返回总计。
统计函数	聚合函数	STDDEV	用于计算总体标准差。
统计函数	聚合函数	STDDEV_POP	计算总体标准差。
统计函数	聚合函数	STDDEV_SAMP	计算样本标准差。
统计函数	聚合函数	VARIANCE	返回参数指定列的方差。
统计函数	聚合函数	APPROX_COUNT_DISTINCT	计算某一列去重后的行数，返回的值是一个近似值，该函数可以进一步用于计算被引用的列的选择性。

分析函数

函数分类	函数子分类	函数名	功能描述
统计函数	分析函数	AVG	返回数值列的平均值。
统计函数	分析函数	COUNT	用于查询参数 `expr` 的行数。
统计函数	分析函数	CUME_DIST	计算一个值在一组值中的累积分布。
统计函数	分析函数	DENSE_RANK	计算有序行组中行的秩，并将秩作为 `NUMBER` 返回。
统计函数	分析函数	MAX	返回参数中指定的列中的最大值。
统计函数	分析函数	MIN	返回参数中指定列的最小值。
统计函数	分析函数	SUM	返回参数中指定列的和。
统计函数	分析函数	FIRST_VALUE	返回有序值中的第一个值。
统计函数	分析函数	LAG	提供对多行表的访问，而不需要自连接。
统计函数	分析函数	LAST_VALUE	返回一组有序值中的最后一个值。
统计函数	分析函数	LEAD	它提供了对表多行的访问，而无需进行自我连接。给定从查询返回的一些列行和光标的位置，`LEAD` 提供超出该位置的物理偏移量的行的访问。

函数分类	函数子分类	函数名	功能描述
统计函数	分析函数	LISTAGG	用于列转行。
统计函数	分析函数	NTH_VALUE	返回 analytic_clause 定义的窗口中第 n 行的 measure_expr 值。
统计函数	分析函数	NTILE	将有序数据集划分为 expr 指示的若干桶，并为每一行分配适当的桶号。
统计函数	分析函数	PERCENT_RANK	类似于 CUME_DIST （累积分布）函数。 它的返回值范围为 0~1。任何集合中的第一行的 PERCENT_RANK 函数为 0，返回值为 NUMBER。
统计函数	分析函数	SUM	返回参数中指定列的和。
统计函数	分析函数	RANK	基于 OVER 子句中的 ORDER BY 表达式确定一组值的排名。
统计函数	分析函数	RATIO_TO_REPORT	计算一个值与一组值之和的比率。
统计函数	分析函数	ROW_NUMBER	为应用它的每一行分配一个唯一的数字。
统计函数	分析函数	STDDEV	用于计算总体标准差。
统计函数	分析函数	STDDEV_POP	计算总体标准差。
统计函数	分析函数	STDDEV_SAMP	计算样本标准差。

函数分类	函数子分类	函数名	功能描述
统计函数	分析函数	VARIANCE	返回参数指定列的方差。

更多信息

分析函数中的关键字 `OVER` ，请参阅 窗口函数说明。

6.6.2. 单行函数

6.6.2.1. 数字函数

6.6.2.1.1. ABS

`ABS` 函数是返回指定数值表达式的绝对值（正值）的数学函数。 `ABS` 将负值更改为正值，对零或正值没有影响。

语法

```
ABS (numeric_expression)
```

参数

参数	说明
numeric_expression	精确数值或近似数值数据类型（ `NUMBER` 、 `FLOAT` 、 `BINARY_FLOAT` 和 `BINARY_DOUBLE` ）的表达式。

返回类型

返回与 `numeric_expression` 相同的类型。

示例

本示例显示了对三个不同数字使用 `ABS` 函数所得的结果。

执行以下语句：

```
SELECT ABS(-1.0), ABS(0.0), ABS(1.0), ABS(1.666) FROM DUAL;
```

查询结果如下：

```
+-----------+----------+----------+------------+
| ABS(-1.0) | ABS(0.0) | ABS(1.0) | ABS(1.666) |
+-----------+----------+----------+------------+
|      1.00 |     0.00 |     1.00 |      1.666 |
+-----------+----------+----------+------------+
```

6.6.2.1.2. ACOS

`ACOS` 函数返回以弧度表示的角，其余弦为指定的 `NUMBER` 表达式，也称为反余弦。

语法

```
ACOS (num_expression)
```

参数

参数	值	说明
num_expression	-1.00~1.00	`NUMBER` 类型或可隐式转换为 `NUMBER` 类型的表达式 。 仅介于 -1.00 到 1.00 之间的值有效。 对于超出此范围的值，将返回 `NULL` 且报告域错误。

返回类型

`NUMBER` 数据类型。

示例

此示例返回指定数量的 `ACOS` 的值。

执行以下语句：

```
SELECT ACOS(0.3)"acos_test" FROM DUAL;
```

查询结果如下：

```
+-----------+
| acos_test |
+-----------+
|1.26610367 |
+-----------+
```

6.6.2.1.3. BITAND

运算符按位进行“与”操作。输入和输出类型均为 NUMBER 数据类型。

语法

```
BITAND (nExpression1, nExpression2)
```

参数

参数	说明
nExpression1, nExpression2	指定按位进行 AND 运算的两个数值。如果 nExpression1 和 nExpression2 为非整数型，那么它们在按位进行 AND 运算之前转换为整数。

返回类型

NUMBER 数据类型。

示例

a 的二进制为 0100；

b 的二进制为 0110；

比较该二进制，若相对应的位置都为 1，则该位的值为 1，否则值为 0。所以 BITAND(2 , 3) = 0100 = 2。

执行以下语句：

```
SELECT BITAND(2,3) FROM DUAL;
```

查询结果如下：

```
+-------------+
| BITAND(2,3) |
+-------------+
|           2 |
+-------------+
```

6.6.2.1.4. CEIL

CEIL 函数返回值大于等于数值 numeric_expression 的最小整数。

语法

```
CEIL (numeric_expression)
```

参数

参数	说明
numeric_expression	精确数值或近似数值数据类型（NUMBER、FLOAT、BINARY_FLOAT 和 BINARY_DOUBLE）的表达式。

返回类型

返回类型与参数 numeric_expression 的数据类型相同。

示例

此示例显示了对三个不同数字使用 CEIL 函数所得的结果。

执行以下语句：

```
SELECT CEIL(1.2), CEIL(2), CEIL(-12.1) FROM DUAL;
```

查询结果如下：

```
+-----------+---------+-------------+
| CEIL(1.2) | CEIL(2) | CEIL(-12.1) |
+-----------+---------+-------------+
|         2 |       2 |         -12 |
+-----------+---------+-------------+
```

6.6.2.1.5. EXP

EXP 函数返回 e 的 numeric_expression 次幂（e 为数学常量，e = 2.71828183... ）。

语法

```
EXP (numeric_expression)
```

参数

参数	说明
numeric_expression	精确数值或近似数值数据类型（NUMBER、FLOAT、BINARY_FLOAT 和 BINARY_DOUBLE）的表达式。

返回类型

返回类型与参数 numeric_expression 的数据类型相同。

示例

此示例显示了查询 e 的 4 次幂结果。

执行以下语句：

```
SELECT EXP(4) "e to the 4th power" FROM DUAL;
```

查询结果如下：

```
+-----------------------------------------+
| e to the 4th power                      |
+-----------------------------------------+
| 54.59815003314423907811026120286087840279 |
+-----------------------------------------+
```

6.6.2.1.6. FLOOR

FLOOR 函数返回小于等于数值 numeric_expression 的最大整数。

语法

```
FLOOR (numeric_expression)
```

参数

参数	说明
numeric_expression	精确数值或近似数值数据类型（NUMBER、FLOAT、BINARY_FLOAT 和 BINARY_DOUBLE）的表达式。

返回类型

返回类型与参数 numeric_expression 的数据类型相同。

示例

此示例显示了对三个不同数字使用 FLOOR 函数所得的结果。

执行以下语句：

```
SELECT FLOOR(1.2), FLOOR(2), FLOOR(-12.1) FROM DUAL;
```

查询结果如下：

```
+------------+----------+--------------+
| FLOOR(1.2) | FLOOR(2) | FLOOR(-12.1) |
+------------+----------+--------------+
|          1 |        2 |          -13 |
+------------+----------+--------------+
```

6.6.2.1.7. LN

LN 函数返回以 e 为底的 numeric_expression 的对数（e 为数学常量 e = 2.71828183...）。

语法

```
LN (numeric_expression)
```

参数

参数	说明
numeric_expression	精确数值或近似数值数据类型（NUMBER、FLOAT、BINARY_FLOAT 和 BINARY_DOUBLE）的表达式。

返回类型

当参数为 BINARY_FLOAT 数据类型时，返回类型为 BINARY_DOUBLE，其他情况下返回类型与参数 numeric_expression 的数据类型相同。

示例

此示例显示了计算以 e 为底的 4 次对数的结果。

执行以下语句：

```
SELECT LN(4) "Natural log of 4" FROM DUAL;
```

查询结果如下：

```
+-----------------------------------------+
| Natural log of 4                        |
+-----------------------------------------+
| 1.38629436111989061883446424291635313615 |
+-----------------------------------------+
```

6.6.2.1.8. LOG

`LOG` 函数返回以 `x` 为底的 `y` 的对数。

语法

```
LOG (x,y)
```

参数

参数	说明
x，y	数值型（`NUMBER`、`FLOAT`、`BINARY_FLOAT` 和 `BINARY_DOUBLE`）表达式。 `x` 和 `y` 都必须大于 0。

返回类型

当参数为 `BINARY_FLOAT` 和 `BINARY_DOUBLE` 数据类型时，返回类型为 `BINARY_DOUBLE`，其他情况下返回类型为 `NUMBER`。

示例

此示例显示了查询了以 2 为底的 8 的对数。

```
SELECT LOG(2,8) FROM DUAL;
```

查询结果如下：

```
+----------+
| LOG(2,8) |
+----------+
|        3 |
+----------+
```

6.6.2.1.9. MOD

MOD 函数返回 x 除以 y 的余数。

> 注意
>
> 与 REMAINDER 函数的区别为，在用 REMAINDER (x,y) 和 MOD (x,y) 函数在进行运算时，都用了一个公式 result=x-y*(x/y)，区别在于计算 x/y 时的处理方式不同。在 REMAINDER (x,y) 函数中，采用 ROUND(x/y) ，而在 MOD (x,y) 函数中采用 FLOOR(x/y) 。

语法

```
MOD (x,y)
```

参数

参数	说明
x，y	x 和 y 为数值型（ NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE ）表达式。

返回类型

返回类型与数值优先级高的参数的数据类型相同。

示例

此示例显示了计算 23/8 以及 24/8 的余数的结果。

执行以下语句：

```
SELECT MOD(23,8), MOD(24,8) FROM DUAL;
```

查询结果如下：

```
+-----------+-----------+
| MOD(23,8) | MOD(24,8) |
+-----------+-----------+
|         7 |         0 |
+-----------+-----------+
```

6.6.2.1.10. POWER

`POWER` 函数返回 `x` 的 `y` 次幂。

语法

```
POWER (x, y)
```

参数

参数	说明
x, y	`x` 和 `y` 为数值型（`NUMBER`、`FLOAT`、`BINARY_FLOAT` 和 `BINARY_DOUBLE`）表达式。

返回类型

当参数为 `BINARY_FLOAT` 和 `BINARY_DOUBLE` 数据类型时，返回类型为 `BINARY_DOUBLE`，其他情况下返回类型为 `NUMBER`。

示例

此示例显示了对三组不同数字使用 `POWER` 函数所得的结果。

执行以下语句：

```
SELECT POWER(2,2), POWER(1.5,0), POWER(20, -1) FROM DUAL;
```

查询结果如下：

```
+------------+--------------+--------------+
| POWER(2,2) | POWER(1.5,0) | POWER(20,-1) |
+------------+--------------+--------------+
|          4 |            1 |          .05 |
+------------+--------------+--------------+
```

6.6.2.1.11. REMAINDER

REMAINDER 函数返回 x 除以 y 的余数。

> 注意
>
> 与 MOD 函数的区别为：
>
> 在用 REMAINDER (x,y) 和 MOD (x,y) 函数在进行运算时，都用了一个公式 result=x-y*(x/y)，区别在于计算 x/y 时的处理方式不同。在 REMAINDER (x,y) 函数中，采用 ROUND(x/y) ，而在 MOD (x,y) 函数中采用 FLOOR(x/y) 。在 REMAINDER 函数中，当 ROUND(x/y) 的参数 x/y 的值的小数部分恰好为 0.5 时，如果 x/y 的值的整数部分为偶数，不向前一位进位，当 x/y 的值的整数部分为奇数，向前一位进位。列如， ROUND(1.5) =2、 ROUND(2.5) =2、 ROUND(3.5) =4、 ROUND(4.5) =4。

语法

```
REMAINDER (x, y)
```

参数

参数	说明
x，y	x 和 y 为数值型（ NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE ）表达式。

返回类型

返回类型与数值优先级高的参数的数据类型相同。

示例

此示例显示了使用 MOD 以及 REMAINDER 函数计算 1.5/1 的余数的结果，请注意两个函数的区别。

执行以下语句：

```
SELECT MOD(1.5,1), REMAINDER(1.5,1) FROM DUAL;
```

查询结果如下：

```
+------------+------------------+
| MOD(1.5,1) | REMAINDER(1.5,1) |
+------------+------------------+
|    0.5 |    -0.5     |
+------------+------------------+
```

6.6.2.1.12. ROUND

ROUND 函数返回参数 numeric 四舍五入后的值。

语法

```
ROUND (numeric[,decimal])
```

参数

参数	说明
numeric	数值型（ NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE ）表达式。
decimal	如果 decimal 大于等于 0 则将 numeric 四舍五入到 decimal 位小数，如果 decimal 小于 0 则四舍五入到小数点向左第 decimal 位。当 decimal 不为整数时，截取 decimal 的整数部分。不指定 decimal 时，将 numeric 四舍五入到整数位。

返回类型

不指定 decimal 时返回类型与参数 numeric 的类型相同，指定 decimal 时，返回类型为 NUMBER 数据类型。

示例

此示例显示了在 decimal 的不同值下四舍五入 5555.6666 的结果。

执行以下语句：

```
SELECT ROUND(5555.6666, 2.1), ROUND(5555.6666, -2.6), ROUND(5555.6666) FROM DUAL;
```

查询结果如下：

```
+----------------------+----------------------+------------------+
| ROUND(5555.6666,2.1) | ROUND(5555.6666,-2.6) | ROUND(5555.6666) |
+----------------------+----------------------+------------------+
|              5555.67 |                 5600 |             5556 |
+----------------------+----------------------+------------------+
```

6.6.2.1.13. SIGN

SIGN 函数返回数字 n 的符号，大于 0 返回 1，小于 0 返回 -1 ，等于 0 返回 0。

语法

```
SIGN (n)
```

参数

参数	说明
n	精确数值或近似数值数据类型（ NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE ）的表达式。

返回类型

返回数值 0，1 和 -1。

示例

此示例显示了对三个不同数字使用 SIGN 函数所得的结果。

执行以下语句：

```
SELECT SIGN(100), SIGN(-100), SIGN(0) FROM DUAL;
```

查询结果如下：

```
+-----------+------------+---------+
| SIGN(100) | SIGN(-100) | SIGN(0) |
+-----------+------------+---------+
|         1 |         -1 |       0 |
+-----------+------------+---------+
```

6.6.2.1.14. SQRT

SQRT 函数返回 n 的平方根。

语法

```
SQRT (n)
```

参数

参数	说明
n	数值型（ NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE ）表达式。 n 不能为负数。

返回类型

返回类型与参数 n 的数据类型相同。

示例

此示例显示了对两个不同数字使用 SQRT 函数求平方根的结果。

执行以下语句：

```
SELECT SQRT(64), SQRT(10) FROM DUAL;
```

查询结果如下：

```
+----------+-----------------------------------------+
| SQRT(64) | SQRT(10)                                |
+----------+-----------------------------------------+
|        8 | 3.16227766016837933199889354443271853372 |
+----------+-----------------------------------------+
```

6.6.2.1.15. TRUNC

TRUNC 函数返回 numberic 按精度 precision 截取后的值。

语法

```
TRUNC (numberic[,precision])
```

参数

参数	说明

参数	说明
numberic，precision	数值型（ NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE ）表达式。如果 precision 不为整数则截取 precision 的整数部分。 precision 大于等于 0 时截取 numeric 到 precision 位小数， precision 小于 0 时截取到 numeric 小数点向左第 precision 位，小数前其它数据用 0 表示。 precision 的默认值为 0。

返回类型

不指定 precision 时，返回类型与参数 numberic 的数据类型相同。指定 precision 时，返回类型为 NUMBER 。

示例

此示例显示了在 precision 的不同值下 5555.66666的计算结果。

执行以下语句：

```
SELECT TRUNC(5555.66666, 2.1), TRUNC(5555.66666, -2.6), TRUNC(5555.66666) FROM DUAL;
```

查询结果如下：

```
+----------------------+-----------------------+--------------------+
| TRUNC(5555.66666,2.1) | TRUNC(5555.66666,-2.6) | TRUNC(5555.033333) |
+----------------------+-----------------------+--------------------+
|              5555.66 |                  5500 |               5555 |
+----------------------+-----------------------+--------------------+
```

6.6.2.2. 返回字符串的字符串函数

6.6.2.2.1. CHR

CHR 函数将 n 转换为等价的一个或多个字符返回，且返回值与当前系统的字符集相关。

语法

```
CHR ( n )
```

参数

参数	取值范围
n	0~4294967295

返回类型

返回值与当前系统的字符集相关。而 OceanBase 支持的字符集是 `UTF-8` 、 `UTF-16` 、 `GBK` 和 `GB18030` 。

示例

十进制（25700）-> 十六进制（0x6464）-> `UTF-8` 编码（dd）

执行以下语句：

```
SELECT CHR(25700) AS str FROM DUAL;
```

查询结果如下：

```
+------+
| STR |
+------+
| dd  |
+------+
```

十进制（50318）-> 十六进制（0xC48E）-> `UTF-8` 编码（Ď）

执行以下语句：

```
SELECT CHR(50318) AS str FROM DUAL;
```

查询结果如下：

```
+------+
| STR |
+------+
| Ď  |
+------+
```

十进制 -> `UTF-8` 编码

```
SELECT CHR(67)||CHR(65)||CHR(84) "Dog" FROM DUAL;
```

查询结果如下：

```
+------+
| Dog |
+------+
| CAT |
+------+
```

6.6.2.2.2. CONCAT

`CONCAT` 函数可以连接两个字符串。

语法

```
CONCAT(c1,c2)
```

参数

参数	说明
c1	字符串，字符串类型可为：`CHAR` 、`VARCHAR2` 、`NCHAR` 、`NVARCHAR2` 或 `CLOB` 。
c1	字符串，字符串类型可为：`CHAR` 、`VARCHAR2` 、`NCHAR` 、`NVARCHAR2` 或 `CLOB` 。

返回类型

返回与 `c1` 相同的数据类型。

示例

本示例 `CONCAT` 函数用来连接字符 `'010-'` 和字符 `'88888888'` 。

执行以下语句

```
SELECT concat('010-','88888888')||'转23'XXXX的电话 FROM DUAL;
```

查询结果如下：

```
+------------------+
| XXXX的电话     |
+------------------+
| 010-88888888转23 |
+------------------+
```

6.6.2.2.3. INITCAP

`INITCAP` 函数返回字符串并将字符串中每个单词的首字母大写，其他字母小写。

语法

```
INITCAP(c1)
```

参数

参数	说明
c1	字符串。字符串类型可为：`CHAR`、`VARCHAR2`、`NCHAR` 或 `NVARCHAR2`。

返回类型

返回 `CHAR` 类型数据。

示例

执行以下语句：

```
SELECT initcap('smith abc aBC') upp FROM DUAL;
```

查询结果如下：

```
+------------------+
| UPP              |
+------------------+
| Smith Abc Abc |
+------------------+
```

6.6.2.2.4. LOWER

LOWER 函数将字符串全部转为小写。

语法

```
LOWER(c1)
```

参数

参数	说明
c1	表示字符串。字符串类型可为：CHAR 、VARCHAR2 、NCHAR 、NVARCHAR2 或 CLOB 。

返回类型

返回与 c1 相同的数据类型 。

示例

执行以下语句：

```
SELECT  lower('AaBbCcDd')AaBbCcDd FROM DUAL;
```

查询结果如下：

```
+----------+
| AABBCCDD |
+----------+
| aabbccdd |
+----------+
```

6.6.2.2.5. LPAD

LPAD 函数的功能是在字符串 c1 的左边用字符串 c2 填充，直到长度为 n 时为止。

语法

```
LPAD(c1,n[,c2])
```

参数

参数	说明
c1	表示字符串。字符串类型可为：CHAR、VARCHAR2、NCHAR、NVARCHAR2 或 CLOB。
n	追加后字符总长度，必须是 NUMBER 整数类型或可以隐式转换为 NUMBER 整数类型的类型。
c2	表示追加的字符串，默认为空格。字符串类型可为：CHAR、VARCHAR2、NCHAR、NVARCHAR2 或 CLOB。

返回类型

如果 c1 是字符型数据，则返回 VARCHAR2 类型；如果 c1 是国家字符数据类型，则返回 NVARCHAR2 类型；如果 c1 是 LOB 数据类型,，则返回 LOB 类型。

> 注意
>
> 如果 c1 长度大于 n，则返回 c1 左边 n 个字符。如果 c1 长度小于 n，c2 和 c1 连接后大于 n，则返回连接后的右边 n 个字符。

示例

执行以下语句：

```
SELECT lpad('gao',10,'*') FROM DUAL;
```

查询结果如下：

```
+--------------------+
| LPAD('GAO',10,'*') |
+--------------------+
| *******gao         |
+--------------------+
```

6.6.2.2.6. LTRIM

LTRIM 函数功能是删除左边出现的字符串。

语法

```
LTRIM(c1 [,c2])
```

`LTRIM` 函数从左端删除 `c1` 中包含的所有字符 `c2` 。如果未指定 `c2` ，则默认为单个空格。

参数

参数	说明
c1	表示字符串。字符串类型可为：`CHAR` 、`VARCHAR2` 、`NCHAR` 、`NVARCHAR2` 或 `CLOB` 。
c2	表示要删除的字符串，默认为空格。字符串类型可为：`CHAR` 、`VARCHAR2` 、`NCHAR` 、`NVARCHAR2` 或 `CLOB` 。

返回类型

如果 `c1` 是 `CHAR` 、`VARCHAR2` 数据类型，则函数返回 `VARCHAR2` 数据类型。如果 `c1` 是 `NCHAR` 、`NVARCHAR2` 数据类型，则函数返回 `NVARCHAR2` 数据类型。如果 `c1` 是 `LOB` 数据类型，则返回的字符串为 `LOB` 数据类型。

示例

执行以下语句：

```
SELECT LTRIM('  gao qian jing',' ') text FROM DUAL;
```

查询结果如下：

```
+------------------+
| TEXT       |
+------------------+
| gao qian jing |
+------------------+
```

6.6.2.2.7. REGEXP_REPLACE

`REGEXP_REPLACE` 函数用于正则表达式替换。

语法

```
REGEXP_REPLACE (source_char, pattern [,replace_string [, position [, occurrence [, match_param ] ] ] ])
```

参数

参数	说明
source_char	用作搜索值的字符表达式。它通常是一种字符列，数据类型可以是 `CHAR` 、 `VARCHAR2` 、 `NCHAR` 、 `NVARCHAR2` 或 `CLOB` 。
pattern	是正则表达式 ，它通常是一个文本文字，数据类型可以是 `CHAR` 、 `VARCHAR2` 、 `NCHAR` 或 `NVARCHAR2` 。
replace_string	表示替换的字符，可以是 `CHAR` 、 `VARCHAR2` 、 `NCHAR` 、 `NVARCHAR2` 或 `CLOB` 类型。
position	是一个正整数类型，指示 OceanBase 从第几个字符开始搜索 `source_char` 的字符，默认为 1，表示从第一个字符开始搜索 `source_char` 。
occurrence	是一个非负整数，指示替换操作的发生，如果指定 0，则 OceanBase 替换所有匹配项；如果指定正整数 n，则 OceanBase 将替换第 n 次出现，默认为全部都替换掉，如果指定参数 0 也是全部匹配。
match_param	是数据类型 `VARCHAR2` 或 `CHAR` 的字符表达式，它允许您更改函数的默认匹配行为。 `i` 表示大小写不敏感； `c` 表示大小写敏感； `n` 表示点号； `.` 表示不匹配换行符号； `m` 表示多行模式； `x` 表示忽略空格字符，默认情况下，空格字符会相互匹配。

返回类型

返回结果与 `source_char` 的数据类型相同。

示例

下面的示例检查字符串，查找两个或多个空格。OceanBase 用一个空间替换两个或多个空间的每一次出现。执行以下语句：

```
SELECT REGEXP_REPLACE('500  OceanBase  Parkway,  Redwood  Shores, CA', '( ){2,}', ' ') "REGEXP_REPLACE
"
FROM DUAL;
```

查询结果如下：

```
REGEXP_REPLACE
----------------------------------------
500 OceanBase Parkway, Redwood Shores, CA
```

6.6.2.2.8. REPLACE

REPLAC 函数将字符表达式值中，部分相同字符串，替换成新的字符串。

语法

```
REPLACE(c1,c2[,c3])
```

参数

参数	说明
c1	等待替换的字符串 CHAR
c2	搜索需要替换的字符串
c3	替换字符串，默认为空(即删除之意，不是空格)

c1 、 c2 和 c3 的数据类型可以是 CHAR 、 VARCHAR2 、 NCHAR 、 NVARCHAR2 和 CLOB 。

返回类型

返回的字符串与 c1 的字符集相同。如果 c3 缺省或者为 NULL ，那么所有 c1 中出现的 c2 都将被移除。如果 c2 为 NULL ，那么结果就是 c1 。如果 c1 是 LOB 数据类型，则函数返回 CLOB 数据类型。如果 c1 不是 LOB 数据类型，则函数返回 VARCHA2 数据类型。

示例

执行以下语句：

```
SELECT replace('he love you','he','i') test FROM DUAL;
```

查询结果如下：

```
+------------+
| TEST     |
+------------+
| i love you |
+------------+
```

6.6.2.2.9. RPAD

`RPAD` 函数在字符串 `c1` 的右边用字符串 `c2` 填充，直到长度为 `n` 时为止。

语法

```
RPAD(c1,n[,c2])
```

参数

参数	说明
c1	表示字符串。字符串类型可为：`CHAR` 、`VARCHAR2` 、`NCHAR` 、`NVARCHAR2` 或 `CLOB` 。
n	追加后字符总长度，必须是 `NUMBER` 整数类型或可以隐式转换为 `NUMBER` 整数类型的类型。
c2	表示追加的字符串，默认为空格。字符串类型可为：`CHAR` 、`VARCHAR2` 、`NCHAR` 、`NVARCHAR2` 或 `CLOB` 。

返回类型

如果 `c1` 是字符型数据，则返回 `VARCHAR2` 类型；如果 `c1` 是国家字符数据类型，则返回 `NVARCHAR2` 类型；如果 `c1` 是 `LOB` 数据类型,，则返回 `LOB` 类型。

说明

- 如果 c1 长度大于 n，则返回 c1 左边 n 个字符；
- 如果 c1 长度小于 n，c1 和 c2 连接后大于 n，则返回连接后的左边 n 个字符；
- 如果 c1 长度小于n，c1 和 c2 连接后小于 n，则返回 c1 与多个重复 c2 连接（总长度>= n ）后的左边 n 个字符。

示例

执行以下语句：

```
SELECT rpad('gao',10,'*a') FROM DUAL;
```

查询结果如下：

```
+--------------------+
| RPAD('GAO',10,'*A') |
+--------------------+
| gao*a*a*a*      |
+--------------------+
```

6.6.2.2.10. RTRIM

RTRIM 函数删除右边出现的字符串，此函数对于格式化查询的输出非常有用。

语法

```
RTRIM(c1 [,c2])
```

RTRIM 从 c1 中出现的所有字符的右端删除 c2 。如果未指定 c2 ，则默认为单个空格。

参数

参数	说明
c1	表示字符串
c2	表示要删除的字符串，默认为空格

c1 、 c2 可以是 CHAR 、 VARCHAR2 、 NCHAR 、 NVARCHAR2 和 CLOB 数据类型。

返回类型

如果 c1 是 CHAR 、VARCHAR2 数据类型，则函数返回 VARCHAR2 数据类型。

如果 c1 是 NCHAR 、NVARCHAR2 数据类型，则函数返回 NVARCHAR2 数据类型。

如果 c1 是 LOB 数据类型，则返回的字符串为 LOB 数据类型。

示例

执行以下语句：

```
SELECT RTRIM('gao qian jingXXXX','X') text FROM DUAL;
```

查询结果如下：

```
+------------------+
| TEXT       |
+------------------+
| gao qian jing |
+------------------+
```

6.6.2.2.11. SUBSTR

SUBSTR 函数截取子字符串。其中多字节符（汉字、全角符等）按 1 个字符计算。

语法

```
SUBSTR(c1,n1[,n2])
```

参数

参数	说明
c1	需要截取的字符串。字符串类型可为：CHAR 、VARCHAR2 、NCHAR 、NVARCHAR2 或 CLOB 。
n1	截取字符串的开始位置，当 n1 等于 0 或 1 时，都是从第一位开始截取。
n2	要截取的字符串的长度，若不指定 n2 ，则从第 n1 个字符直到结束的字串。

返回类型

返回与 `c1` 类型相同的数据。如果 `n2` 为 0，则将其视为 1。如果 `n2` 为正，则 OceanBase 从开始算起，从 `c1` 开始查找第一个字符。如果 `n2` 为负，则 OceanBase 从 `c1` 的末尾开始倒数 `c1` 。如果 `n3` 省略，则 OceanBase 将 `c1` 的所有字符返回。如果 `n3` 小于 1，则 OceanBase 返回 `NULL` 。

示例

执行以下语句：

```
SELECT SUBSTR('13088888888',3,8) test FROM DUAL;
```

查询结果如下：

```
+----------+
| TEST     |
+----------+
| 08888888 |
+----------+
```

6.6.2.2.12. TRANSLATE

`TRANSLATE` 函数将字符表达式值中，指定字符替换为新字符。多字节符（汉字、全角符等），按 1 个字符计算。

语法

```
TRANSLATE(c1,c2,c3)
```

参数

参数	说明
c1	希望被替换的字符或变量。
c2	查询原始的字符集。
c3	替换新的字符集，将 `c2` 对应顺序字符，替换为 `c3` 对应顺序字符 。

说明

- c1 、 c2 和 c3 的数据类型可以是 CHAR 、 VARCHAR2 、 NCHAR 、 NVARCHAR2 或 CLOB 。
- 如果 c3 长度大于 c2 ，则 c3 长出后面的字符无效；
- 如果 c3 长度小于 c2 ，则 c2 长出后面的字符均替换为空（删除）；
- 如果 c3 长度为 0，则返回空字符串；
- 如果 c2 里字符重复，按首次位置为替换依据。

返回类型

返回 CHAR 、 VARCHAR2 、 NCHAR 、 NVARCHAR2 或 CLOB 类型字符。

示例

执行以下语句：

```
SELECT TRANSLATE('he love you','he','i') FROM DUAL;
```

查询结果如下：

```
+------------------+
| TRANSLATE    |
+------------------+
| i love you    |
+------------------+
```

6.6.2.2.13. TRIM

TRIM 函数用来删除一个字符串的开头或结尾（或两者）的字符。

语法

```
TRIM([{{LEADING | TRAILING | BOTH}[trim_character] | trim_character} FROM ] trim_source)
```

参数

参数	说明
LEADING	开头字符。

参数	说明
TRAILING	结尾字符。
BOTH	开头和结尾字符。
trim_character	删除的字符。
trim_source	修剪源。

`trim_char` 和 `trim_source` 都可以是 `VARCHAR2` 或任何可以隐式转换为 `VARCHAR2` 的数据类型。如果指定 `LEADING`，则 OceanBase 将删除所有与前导字符相等的 `trim_character` 。如果指定 `TRAILING`，则 OceanBase 将删除所有与结尾字符相等的 `trim_character` 。如果您指定 `BOTH` 或三个都不指定，则 OceanBase 删除与前导和结尾字符相等的 `trim_character` 。如果未指定 `trim_character`，则默认值为空白。如果仅指定 `trim_source`，则 OceanBase 删除前导和尾随空格。如果函数返回的值数据类型为 `VARCHAR2`，则该值的最大长度为 `trim_source` 。

返回类型

如果 `trim_source` 为 `CHAR` 、 `VARCHAR2` 数据类型，则函数返回 `VARCHAR2` 数据类型。

如果 `trim_source` 为 `NCHAR` 、 `NVARCHAR2` 数据类型，则函数返回 `NVARCHAR2` 数据类型。

如果 `trim_source` 为 `CLOB` 数据类型，则函数返回 `CLOB` 数据类型。

如果 `trim_source` 或 `trim_character` 为 `NULL`，则 `TRIM` 函数返回 `NULL` 。

示例

执行以下语句：

```
SELECT TRIM('X' from 'XXXgao qian jingXXXX'),TRIM('X' from 'XXXgaoXXjingXXXX') text FROM DUAL;
```

查询结果如下：

```
+----------------------------------+-----------+
| TRIM('X'FROM'XXXGAOQIANJINGXXXX') | TEXT    |
+----------------------------------+-----------+
| gao qian jing                | gaoXXjing |
+----------------------------------+-----------+
```

6.6.2.2.14. UPPER

`UPPER` 函数将字符串全部转为大写。

语法

```
UPPER(c1)
```

参数

参数	说明
c1	字符串。字符串类型可为：CHAR 、VARCHAR2 、NCHAR 、NVARCHAR2 或 CLOB 。

返回类型

返回 CHAR 、VARCHAR2 、NCHAR 、NVARCHAR2 或 CLOB 类型的字符。

示例

执行以下语句：

```
SELECT UPPER('AaBbCcDd') upper FROM DUAL;
```

查询结果如下：

```
+----------+
| UPPER    |
+----------+
| AABBCCDD |
+----------+
```

6.6.2.3. 返回数字的字符串函数

6.6.2.3.1. ASCII

ASCII 函数返回字符表达式最左端字符的 ASCII 码值。

语法

```
ASCII(x)
```

参数

参数	说明
x	CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的表达式。

返回类型

NUMBER 数据类型。

示例

此示例使用 ASCII 函数返回 A、a、中 和空格的 ASCII 码值。

```
SELECT ASCII('A') A, ASCII('a') a, ASCII(' ') space,ASCII('中') hz FROM DUAL;
```

结果返回：

```
+----+----+-------+-----+
| A | a | space | hz |
+----+----+-------+-----+
| 65 | 97 |  32 | 228 |
+----+----+-------+-----+
```

6.6.2.3.2. INSTR

INSTR 函数在一个字符串中搜索指定的字符，返回发现指定的字符的位置。

> 注意
>
> 多字节符（汉字、全角符等），按 1 个字符计算。

语法

```
INSTR(c1,c2[,i[,j]])
```

参数

参数	说明

参数	说明
c1	被搜索的字符串。字符串类型可为：`CHAR`、`VARCHAR2`、`NCHAR`、`NVARCHAR2` 或 `CLOB`。
c2	希望搜索的字符串。字符串类型可为：`CHAR`、`VARCHAR2`、`NCHAR`、`NVARCHAR2` 或 `CLOB`。
i	搜索的开始位置，默认值为 1。当值小于 0 时，搜索从反方向开始，但是仍返回被搜索到的字符的正序位置。
j	第 `j` 次出现的位置，默认值为1。

返回类型

`NUMBER` 数据类型。

示例

此示例展示了字符 **某** 在字符串 **重庆某软件公司** 中第一次出现的位置。

```
SELECT INSTR ('重庆某软件公司','某',1,1) instring FROM DUAL;
```

返回结果：

```
+----------+
|instring|
+----------+
|    3|
+----------+
```

此示例中， **instring1** 应返回正向搜索 **ce** 时在字符串中第二次出现的位置，**instring2** 应返回反向搜索 **ce** 时在字符串中第二次出现的位置：

```
SELECT INSTR ('oceanbase pratice','ce',1,2) instring1，INSTR ('oceanbase pratice','ce',-1,2) instring2 FROM DUAL;
```

返回结果，正向搜索时 ce 第二次出现在第 16 位，反向搜索时 ce 第二次出现在第二位：

```
+----------+------------+
| instring1 | instring2 |
+----------+------------+
|    16 |      2 |
+----------+------------+
```

6.6.2.3.3. LENGTH

`LENGTH` 函数返回字符串的长度。

> 注意
>
> 多字节符（汉字、全角符等），按 1 个字符计算。

语法

```
LENGTH(c1)
```

参数

参数	说明
c1	`CHAR` 、`VARCHAR2` 、`NCHAR` 、`NVARCHAR2` 或 `CLOB` 数据类型的字符串。

返回类型

`NUMBER` 数据类型。

示例

此示例展示了字符串 **测试**、**北京市海淀区** 和 **北京TO_CHAR** 的长度。

```
SELECT LENGTH ('测试'), LENGTH('北京市海锭区'), LENGTH('北京TO_CHAR') FROM DUAL;
```

返回结果：

```
+-----------------+---------------------------+-----------------------+
| LENGTH('测试')| LENGTH('北京市海锭区') | LENGTH('北京TO_CHAR') |
+-----------------+---------------------------+-----------------------+
|       2|             6|           9|
+-----------------+---------------------------+-----------------------+
```

6.6.2.4. 时间日期函数

6.6.2.4.1. ADD_MONTHS

`ADD_MONTHS` 函数返回在日期 `date` 基础上 `n` 个月后的日期值，如果 `n` 的值为负数则返回日期 `date` 基础上 `n` 个月前的日期值（ `date` 减去 `n` 个月）。

> **注意**
>
> 由于每个月的天数不同，当 `date` 是一个月中的最后一天时，函数返回计算后该月的最后一天。例如，用 `ADD_MONTHS` 计算 2020 年 3 月 31 日一个月前的日期，返回 2020 年 2 月 29 日。

语法

```
ADD_MONTHS (date,n)
```

参数

参数	说明
date	`DATE` 数据类型。
n	`NUMBER` 数据类型。

返回类型

`DATE` 数据类型。

示例

示例 1： 以下示例查询了 3 个月后的日期。

执行以下语句：

```
SELECT SYSDATE, ADD_MONTHS(SYSDATE,3) FROM DUAL;
```

查询结果如下：

```
+---------------------+-----------------------+
| SYSDATE             | ADD_MONTHS(SYSDATE,3) |
+---------------------+-----------------------+
| 2020-03-26 12:21:40 | 2020-06-26 12:21:40   |
+---------------------+-----------------------+
```

示例 2： 以下示例查询了 3 个月前的日期。

执行以下语句：

```
SELECT SYSDATE, ADD_MONTHS(SYSDATE,-3) FROM DUAL;
```

查询结果如下：

```
+---------------------+-----------------------+
| SYSDATE             | ADD_MONTHS(SYSDATE,-3) |
+---------------------+-----------------------+
| 2020-03-26 12:21:04 | 2019-12-26 12:21:04   |
+---------------------+-----------------------+
```

6.6.2.4.2. CURRENT_DATE

`CURRENT_DATE` 函数返回当前会话时区中的当前日期。

语法

```
CURRENT_DATE
```

参数

无参数。

返回类型

`DATE` 数据类型。

示例

此示例显示了不同会话时区使用 `CURRENT_DATE` 函数所得的结果。

设置当前时区至 GMT-5 时区：

```
ALTER SESSION SET TIME_ZONE = '-05:00';
```

执行以下语句调用函数：

```
SELECT CURRENT_DATE FROM DUAL;
```

查询结果如下：

```
+---------------------+
| CURRENT_DATE        |
+---------------------+
| 2020-03-08 01:40:11 |
+---------------------+
```

切换当前时区至 GMT+8 时区

```
ALTER SESSION SET TIME_ZONE = '+08:00';
```

执行以下语句调用函数：

```
SELECT CURRENT_DATE FROM DUAL;
```

查询结果如下：

```
+---------------------+
| CURRENT_DATE        |
+---------------------+
| 2020-03-08 14:40:11 |
+---------------------+
```

6.6.2.4.3. CURRENT_TIMESTAMP

`CURRENT_TIMESTAMP` 函数返回 `TIMESTAMP WITH TIME ZONE` 数据类型的当前会话时区中的当前日期，返回值中包含当前的时区信息。

语法

```
CURRENT_TIMESTAMP (precision)
```

参数

参数	说明
precision	表示秒小数位的精度，默认值为 6，取值范围 0~9。

返回类型

包含当前的时区信息的 `TIMESTAMP WITH TIME ZONE` 数据类型。

示例

此示例显示了不同会话时区使用 `CURRENT_TIMESTAMP` 函数所得的结果。

设置当前时区至 GMT-5 时区：

```
ALTER SESSION SET TIME_ZONE = '-05:00';
```

执行以下语句调用函数：

```
SELECT CURRENT_TIMESTAMP FROM DUAL;
```

查询结果如下：

```
+-----------------------------------+
| CURRENT_TIMESTAMP                 |
+-----------------------------------+
| 2020-03-08 01:49:31.219066 -05:00 |
+-----------------------------------+
```

切换当前时区至 GMT+8 时区，且调整秒的小数位精度为 3：

```
ALTER SESSION SET TIME_ZONE = '+08:00';
```

执行以下语句调用函数：

```
SELECT CURRENT_TIMESTAMP(3) FROM DUAL;
```

查询结果如下：

```
+---------------------------------+
| CURRENT_TIMESTAMP(3)            |
+---------------------------------+
| 2020-03-08 14:50:32.499 +08:00  |
+---------------------------------+
```

6.6.2.4.4. DBTIMEZONE

`DBTIMEZONE` 函数返回当前数据库实例的时区，在 OceanBase 中数据库时区恒为+00:00，且不支持修改。

语法

```
DBTIMEZONE
```

参数

无参数。

返回类型

`VARCHAR2` 数据类型。

示例

执行以下语句：

```
SELECT DBTIMEZONE FROM DUAL;
```

查询结果如下：

```
+------------+
| DBTIMEZONE |
+------------+
| +00:00     |
+------------+
```

6.6.2.4.5. EXTRACT（datetime）

`EXTRACT(datetime)` 函数是从指定的时间字段或表达式中抽取年、月、日、时、分、秒等元素。

语法

```
EXTRACT (fields FROM datetime)
```

参数

参数	说明
fields	要抽取的元素的名称：`YEAR` 、`MONTH` 、`DAY` 、`HOUR` 、`MINUTE` 、`SECOND` 、`TIMEZONE_HOUR` 、`TIMEZONE_MINUTE` 、`TIMEZONE_REGION` 、`TIMEZONE_ABBR` 。

参数	说明
datetime	DATE 、 TIMESTAMP 、 TIMESTAMP WITH TIME ZONE 、 TIMESTAMP WITH LOCAL TIME ZONE 、 INTERVAL YEAR TO MONTH 、 INTERVAL DAY TO SECOND 等数据类型的值。

返回类型

抽取元素 TIMEZONE_REGION 、 TIMEZONE_ABBR 时，返回值的数据类型为 VARCHAR2 。抽取其他元素时，返回值的数据类型为 NUMBER 。

示例

执行以下语句：

```
SELECT  EXTRACT(HOUR FROM TIMESTAMP '2001-2-16 2:38:40 ') 小时,
    EXTRACT(MINUTE FROM TIMESTAMP '2001-2-16 2:38:40 ') 分钟,
      EXTRACT(SECOND FROM TIMESTAMP '2001-2-16 2:38:40 ') 秒,
      EXTRACT(DAY FROM TIMESTAMP '2001-2-16 2:38:40 ') 日,
      EXTRACT(MONTH FROM TIMESTAMP '2001-2-16 2:38:40 ') 月,
      EXTRACT(YEAR FROM TIMESTAMP '2001-2-16 2:38:40 ') 年
FROM DUAL;
```

查询结果如下：

```
+--------+--------+------+------+------+------+
|小时 |分钟 |秒 |日 |月 |年 |
+--------+--------+------+------+------+------+
|  2|  38| 40| 16|  2|2001|
+--------+--------+------+------+------+------+
```

6.6.2.4.6. FROM_TZ

FROM_TZ 函数将一个 TIMSTAMP 数据类型的值和时区信息拼成一个 TIMESTAMP WITH TIME ZONE 数据类型的时间值。

语法

```
FROM_TZ (timestamp_value,time_zone_value)
```

参数

参数	说明
timestamp_value	TIMSTAMP 数据类型的时间值。
time_zone_value	时区信息。

返回类型

TIMESTAMP WITH TIME ZONE 数据类型。

示例

执行以下语句：

```
SELECT FROM_TZ(TIMESTAMP '2020-03-28 08:00:00', '-03:00') FROM DUAL;
```

查询结果如下：

```
+---------------------------------------------+
| FROM_TZ(TIMESTAMP'2020-03-2808:00:00','-03:00') |
+---------------------------------------------+
| 2020-03-28 08:00:00.000000000 -03:00        |
+---------------------------------------------+
```

6.6.2.4.7. LAST_DAY

LAST_DAY 函数返回日期 date 所在月份的最后一天的日期。

语法

```
LAST_DAY (date)
```

参数

参数	说明
date	所有包含日期信息的数据类型（DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONE）的值。

返回类型

DATE 数据类型。

示例

本示例显示了当前日期与本月的最后一天日期以及本月剩余的天数。

执行以下语句：

```
SELECT SYSDATE, LAST_DAY(SYSDATE)Last, LAST_DAY(SYSDATE)-SYSDATE "Left" FROM DUAL;
```

查询结果如下：

```
+-------------------+-------------------+------+
| SYSDATE           | LAST              | Left |
+-------------------+-------------------+------+
| 2020-03-08 15:23:33 | 2020-03-31 15:23:33 |  23 |
+-------------------+-------------------+------+
```

6.6.2.4.8. LOCALTIMESTAMP

LOCALTIMESTAMP 函数返回当前会话时区中的当前日期，返回 TIMESTAMP 数据类型的值。与函数 CURRENT_TIMESTAMP 的区别是， CURRENT_TIMESTAMP 函数返回 TIMESTAMP WITH TIME ZONE 数据类型的值。

语法

```
LOCALTIMESTAMP (timestamp_precision)
```

参数

参数	说明
timestamp_precision	表示秒的小数位的精度，默认值为 6，取值范围 0~9。

返回类型

TIMESTAMP 数据类型。

示例

此示例显示了不同会话时区使用 LOCALTIMESTAMP 函数所得的结果。

设置当前时区至 GMT-5 时区：

```
ALTER SESSION SET TIME_ZONE = '-05:00';
```

执行以下语句调用函数：

```
SELECT LOCALTIMESTAMP FROM DUAL;
```

查询结果如下：

```
+----------------------------+
| LOCALTIMESTAMP             |
+----------------------------+
| 2020-03-08 02:30:20.062104 |
+----------------------------+
```

切换当前时区至 GMT+8 时区，且调整秒的小数位精度为 3：

```
ALTER SESSION SET TIME_ZONE = '+08:00';
```

执行以下语句调用函数：

```
SELECT LOCALTIMESTAMP(3) FROM DUAL;
```

查询结果如下：

```
+-------------------------+
| LOCALTIMESTAMP(3)       |
+-------------------------+
| 2020-03-08 15:30:54.500 |
+-------------------------+
```

6.6.2.4.9. MONTHS_BETWEEN

`MONTHS_BETWEEN` 函数是返回返回参数 `date1` 到 `date2` 之间的月数。

语法

```
MONTHS_BETWEEN (date1，date2)
```

参数

参数	说明
date1	`DATE` 数据类型的值。

参数	说明
date2	DATE 数据类型的值。

返回类型

NUMBER 数据类型。如果 date1 > date2，返回正数；如果 date1 < date2，则返回负数。

示例

以下示例展示了当前时间和指定时间值之间的月数：

```
SELECT SYSDATE,
MONTHS_BETWEEN(SYSDATE,TO_DATE('2006-01-01','YYYY-MM-DD')),
MONTHS_BETWEEN(SYSDATE,TO_DATE('2022-01-01','YYYY-MM-DD'))
FROM DUAL;
```

查询结果如下：

```
+---------------------+-----------------------------------------------------------+-----------------------------------------------------------
+
| SYSDATE             | MONTHS_BETWEEN(SYSDATE,TO_DATE('2006-01-01','YYYY-MM-DD')) | MONTHS_BETWEEN(
SYSDATE,TO_DATE('2022-01-01','YYYY-MM-DD')) |
+---------------------+-----------------------------------------------------------+-----------------------------------------------------------
+
| 2020-03-08 15:38:35 |              170.24683206391875746714456391875746714 5 |          -21.7531679360812425
3285543608124253285544 |
+---------------------+-----------------------------------------------------------+-----------------------------------------------------------
+
```

6.6.2.4.10. NEXT_DAY

NEXT_DAY 函数是返回日期 d1 的下一周中 c1 （星期值）所在的日期值。

语法

```
NEXT_DAY (d1[,c1])
```

参数

参数	说明
d1	`DATE` 数据类型的值。
c1	星期值：`MONDAY`、`TUESDAY`、`WEDNESDAY`、`THURSDAY`、`FRIDAY`、`SATURDAY` 和 `SUNDAY`。

返回类型

`DATE` 数据类型。

示例

以下示例展示了下周每个星期值对应的日期值：

```
SELECT SYSDATE 当时日期,
NEXT_DAY(SYSDATE,'MONDAY') 下周星期一,
NEXT_DAY(SYSDATE,'TUESDAY') 下周星期二,
NEXT_DAY(SYSDATE,'WEDNESDAY') 下周星期三,
NEXT_DAY(SYSDATE,'THURSDAY') 下周星期四,
NEXT_DAY(SYSDATE,'FRIDAY') 下周星期五,
NEXT_DAY(SYSDATE,'SATURDAY') 下周星期六,
NEXT_DAY(SYSDATE,'SUNDAY') 下周星期日
FROM DUAL;
```

查询结果如下：

```
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
| 当时日期            | 下周星期一          | 下周星期二          | 下周星期三          | 下周星期四          | 下周星期五          | 下周星期六          | 下周星期日          |
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
| 2020-03-08 15:47:57 | 2020-03-09 15:47:57 | 2020-03-10 15:47:57 | 2020-03-11 15:47:57 | 2020-03-12 15:47:57 | 2020-03-13 15:47:57 | 2020-03-14 15:47:57 | 2020-03-15 15:47:57 |
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
```

6.6.2.4.11. NUMTODSINTERVAL

NUMTODSINTERVAL 函数是把参数 n 转为以参数 interval_unit 为单位的 INTERVAL DAY TO SECOND 数据类型的值。

语法

```
NUMTODSINTERVAL (n,interval_unit)
```

参数

参数	说明
n	NUMBER 数据类型的值。
interval_unit	单位值：DAY 、HOUR 、MINUTE 、MINUTE 。

返回类型

INTERVAL DAY TO SECOND 数据类型。

示例

以下示例展示了当前日期 3 小时后的日期时间值：

```
SELECT SYSDATE,SYSDATE+NUMTODSINTERVAL(3,'HOUR') AS RES FROM DUAL;
```

查询结果如下：

```
+---------------------+---------------------+
| SYSDATE             | RES                 |
+---------------------+---------------------+
| 2020-03-08 16:01:40 | 2020-03-08 19:01:40 |
+---------------------+---------------------+
```

6.6.2.4.12. NUMTOYMINTERVAL

NUMTOYMINTERVAL 函数是把参数 n 转为以 interval_unit 为单位的 INTERVAL YEAR TO MONTH 数据类型的值。

语法

```
NUMTOYMINTERVAL (n,interval_unit)
```

参数

参数	说明
n	NUMBER 数据类型的值。
interval_unit	单位值：YEAR 、MONTH 。

返回类型

INTERVAL YEAR TO MONTH 数据类型。

示例

以下示例展示了当前日期 3 年后的日期时间值：

```
SELECT SYSDATE,SYSDATE+NUMTOYMINTERVAL(3,'YEAR') AS RES FROM DUAL;
```

查询结果如下：

```
+---------------------+---------------------+
| SYSDATE             | RES                 |
+---------------------+---------------------+
| 2020-03-08 16:03:58 | 2023-03-08 16:03:58 |
+---------------------+---------------------+
```

6.6.2.4.13. ROUND (date)

ROUND(date) 函数返回以参数 fmt 为单位距离的离指定日期 date 最近的日期时间值。

语法

```
ROUND (date，[fmt])
```

参数

参数	说明
date	所有包含日期的数据类型的值：DATE 、TIMESTAMP 、TIMESTAMP WITH TIME ZONE 、TIMESTAMP WITH LOCAL TIME ZONE 。

参数	说明
fmt	指定了函数返回值与 `date` 的距离单位，以下表格列举了该参数的可取值，大小写不敏感。

fmt参数表	说明
j	默认值，最近0点日期。
day、dy、d	返回离指定日期最近的星期日。
month、mon、mm、rm	返回离指定日期最近的月的第一天日期。
q	返回离指定日期最近的季的日期。
syear、year、yyyy、yyy、yy、y	多个 y 表示不同的精度，返回离指定日期最近的年的第一个日期。
cc、scc	返回离指定日期最近的世纪的初日期。

返回类型

`DATE` 数据类型。

示例

执行以下语句：

```
SELECT SYSDATE 当时日期,
ROUND(SYSDATE) 最近0点日期,
ROUND(SYSDATE,'DAY') 最近星期日,
ROUND(SYSDATE,'MONTH') 最近月初,
ROUND(SYSDATE,'Q') 最近季初日期,
ROUND(SYSDATE,'YEAR') 最近年初日期
FROM DUAL;
```

查询结果如下：

```
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
| 当时日期           | 最近0点日期        | 最近星期日         | 最近月初           | 最近季初日期       | 最近年初日期       |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
| 2020-03-08 20:24:53 | 2020-03-09 00:00:00 | 2020-03-08 00:00:00 | 2020-03-01 00:00:00 | 2020-04-01 00:00:00 | 2020-01-01 00:00:00 |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
```

6.6.2.4.14. SESSIONTIMEZONE

`SESSIONTIMEZONE` 函数是返回当前会话时区。

语法

```
SESSIONTIMEZONE
```

参数

无参数。

返回类型

`VARCHAR2` 数据类型。

示例

以下示例展示了数据库时区和当前会话时区：

```
SELECT DBTIMEZONE,SESSIONTIMEZONE FROM DUAL;
```

查询结果如下：

```
+------------+-----------------+
| DBTIMEZONE | SESSIONTIMEZONE |
+------------+-----------------+
| +00:00     | +08:00          |
+------------+-----------------+
```

可以通过 `ALTER SESSION` 语句修改当前会话时区，数据库时区不可修改：

```
ALTER SESSION SET TIME_ZONE = '+05:00';
```

执行以下语句查询修改后的当前会话时区：

```
SELECT DBTIMEZONE,SESSIONTIMEZONE FROM DUAL;
```

查询结果如下：

```
+------------+-----------------+
| DBTIMEZONE | SESSIONTIMEZONE |
+------------+-----------------+
| +00:00     | +05:00          |
+------------+-----------------+
```

6.6.2.4.15. SYS_EXTRACT_UTC

`SYS_EXTRACT_UTC` 函数是返回与指定时间相对应的标准 UTC 时间。

语法

```
SYS_EXTRACT_UTC (datetime_with_timezone)
```

> **注意**
>
> UTC（Universal Time Coordinated）是通用协调时间。UTC 与格林尼治平均时（GMT, Greenwich Mean Time）一样，都与英国伦敦的本地时相同。

参数

参数	说明
datetime_with_timezone	`TIMESTAMP WITH TIME ZONE` 和 `TIMESTAMP WITH LOCAL TIME ZONE` 数据类型的值。

返回类型

`TIMESTAMP` 数据类型。

示例

执行以下语句：

```
SELECT SYS_EXTRACT_UTC(TIMESTAMP '2020-03-28 11:30:00.00 -08:00')
FROM DUAL;
```

查询结果如下：

```
+-----------------------------------------------------+
| SYS_EXTRACT_UTC(TIMESTAMP'2020-03-2811:30:00.00-08:00') |
+-----------------------------------------------------+
| 2020-03-28 19:30:00.000000000                       |
+-----------------------------------------------------+
```

6.6.2.4.16. SYSDATE

`SYSDATE` 函数返回当前日期。

> **注意**
>
> 该函数不依赖于当前会话时区，而是依赖于数据库宿主机的 Linux 操作系统的时区。

语法

```
SYSDATE
```

参数

无参数。

返回类型

`DATE` 数据类型。

示例

以下示例将当前时间按指定格式输出：

```
SELECT TO_CHAR
(SYSDATE, 'MM-DD-YYYY HH24:MI:SS') "NOW"
FROM DUAL;
```

查询结果如下：

```
+---------------------+
| NOW                 |
+---------------------+
| 03-08-2020 20:44:04 |
+---------------------+
```

6.6.2.4.17. SYSTIMESTAMP

`SYSTIMESTAMP` 函数返回系统当前日期，返回值的秒的小数位包含 6 位精度，且包含当前时区信息。

> 注意
>
> 该函数不依赖于当前会话时区，而是依赖于数据库宿主机的 Linux 操作系统的时区。

语法

```
SYSTIMESTAMP
```

参数

无参数。

返回类型

TIMESTAMP WITH TIME ZONE 数据类型。

示例

执行以下语句：

```
SELECT SYSTIMESTAMP FROM DUAL;
```

查询结果如下：

```
+-----------------------------------+
| SYSTIMESTAMP                      |
+-----------------------------------+
| 2020-03-08 20:47:08.254086 +08:00 |
+-----------------------------------+
```

6.6.2.4.18. TO_CHAR (datetime)

TO_CHAR 函数将 DATE 、 TIMESTAMP 、 TIMESTAMP WITH TIME ZONE 、 TIMESTAMP WITH LOCAL TIME ZONE 、 INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 等数据类型的值按照参数 fmt 指定的格式转换为 VARCHAR2 数据类型的值。 如果不指定参数 fmt ，则参数 datetime 的值将按如下格式转换为 VARCHAR2 数据类型：

- DATE 、 TIMESTAMP 、 TIMESTAMP WITH TIME ZONE 和 TIMESTAMP WITH LOCAL TIME ZONE 的值被转换为数据库中日期时间值的默认格式。您可在数据类型章节中查看各日期时间类型的默认格式。
- INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 数据类型的值转换为数字格式的间隔值。

语法

```
TO_CHAR({ datetime | interval } [, fmt [, 'nlsparam' ] ])
```

参数

参数	说明
datetime	DATE 、TIMESTAMP 、TIMESTAMP WITH TIME ZONE 、TIMESTAMP WITH LOCAL TIME ZONE 、INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 等数据类型的值。
fmt	输出格式参数。
nlsparam	用来控制返回的月份和日份所使用的语言。

返回类型

VARCHAR2 数据类型

示例

示例 1： 以下语句通过 TO_CHAR 函数返回系统当前日期，并且将日期时间值转换为了 **DS DL** 格式：

```
SELECT TO_CHAR(SYSDATE,'DS DL') FROM DUAL;
```

查询结果如下：

```
+----------------------------------+
| TO_CHAR(SYSDATE,'DSDL')          |
+----------------------------------+
| 03/08/2020 Sunday, March 08, 2020 |
+----------------------------------+
```

示例 2：以下语句将间隔值转化为指定格式，并且设置了返回语言为 **AMERICAN**：

```
SELECT TO_CHAR(interval'1' year, 'SS-MI-HH', 'nls_language = AMERICAN') FROM DUAL;
```

查询结果如下：

```
+-----------------------------------------------------------+
| TO_CHAR(INTERVAL'1'YEAR,'SS-MI-HH','NLS_LANGUAGE=AMERICAN') |
+-----------------------------------------------------------+
| +01-00                                                    |
+-----------------------------------------------------------+
```

6.6.2.4.19. TO_DSINTERVAL

TO_DSINTERVAL 函数将一个 CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串转换为一个 INTERVAL DAY TO SECOND 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。

语法

```
TO_DSINTERVAL (days hours：minutes：seconds[.frac_secs])
```

参数

参数	说明
days hours：minutes：seconds[.frac_secs]	符合该参数格式的 CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串。

返回类型

INTERVAL DAY TO SECOND 数据类型。

示例

以下示例返回了当前时间 100 天后的日期时间值：

```
SELECT SYSDATE, SYSDATE+TO_DSINTERVAL('100 00:00:00') FROM DUAL;
```

查询结果如下：

```
+---------------------+-------------------------------------+
| SYSDATE             | SYSDATE+TO_DSINTERVAL('10000:00:00') |
+---------------------+-------------------------------------+
| 2020-03-26 12:40:39 | 2020-07-04 12:40:39                 |
+---------------------+-------------------------------------+
```

6.6.2.4.20. TO_TIMESTAMP

TO_TIMESTAMP 函数将字符串转换为 TIMESTAMP 数据类型。

语法

```
TO_TIMESTAMP (char，[fmt],['nlsparam'])
```

参数

	说明
char	CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串。
fmt	指定返回值的格式。
nlsparam	用来控制返回的月份和日份所使用的语言。

返回类型

TIMESTAMP 数据类型。

示例

执行以下语句：

```
SELECT TO_TIMESTAMP ('10-Sep-02 14:10:10.123000', 'DD-Mon-RR HH24:MI:SS.FF')
FROM DUAL;
```

查询结果如下：

```
+---------------------------------------------------------------+
| TO_TIMESTAMP('10-SEP-0214:10:10.123000','DD-MON-RRHH24:MI:SS.FF') |
+---------------------------------------------------------------+
| 2002-09-10 14:10:10.123000000                                 |
+---------------------------------------------------------------+
```

6.6.2.4.21. TO_TIMESTAMP_TZ

TO_TIMESTAMP_TZ 函数将字符串转换为 TIMESTAMP WITH TIME ZONE 数据类型，包含时区信息。

语法

```
TO_TIMESTAMP_TZ (char，[fmt],['nlsparam'])
```

参数

参数	说明
char	CHAR 、VARCHAR2 、NCHAR 或 NVARCHAR2 数据类型的字符串。

fmt	指定输出格式。
nlsparam	用来控制返回的月份和日份所使用的语言。

返回类型

`TIMESTAMP WITH TIME ZONE` 数据类型。

示例

执行以下语句：

```
SELECT TO_TIMESTAMP_TZ ('10-Sep-02 14:10:10.123000', 'DD-Mon-RR HH24:MI:SS.FF') FROM DUAL;
```

查询结果如下：

```
+---------------------------------------------------------------+
| TO_TIMESTAMP_TZ('10-SEP-0214:10:10.123000','DD-MON-RRHH24:MI:SS.FF') |
+---------------------------------------------------------------+
| 2002-09-10 14:10:10.123000000 +08:00                          |
+---------------------------------------------------------------+
```

6.6.2.4.22. TO_YMINTERVAL

`TO_YMINTERVAL` 函数将一个 `CHAR` 、 `VARCHAR2` 、 `NCHAR` 或 `NVARCHAR2` 数据类型的字符串转换为一个 `INTERVAL YEAR TO MONTH` 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。

语法

```
TO_YMINTERVAL (years-months)
```

参数

参数	说明
years-months	符合该参数格式的 `CHAR` 、 `VARCHAR2` 、 `NCHAR` 或 `NVARCHAR2` 数据类型的字符串。

返回类型

INTERVAL YEAR TO MONTH 数据类型。

示例

以下示例展示了返回当前时间 1 年 2 个月后的时间日期值：

```
SELECT SYSDATE,SYSDATE+TO_YMINTERVAL('01-02') FROM DUAL;
```

查询结果如下：

```
+---------------------+-------------------------------+
| SYSDATE             | SYSDATE+TO_YMINTERVAL('01-02') |
+---------------------+-------------------------------+
| 2020-03-08 22:32:01 | 2021-05-08 22:32:01            |
+---------------------+-------------------------------+
```

6.6.2.4.23. TRUNC (date)

TRUNC 函数返回以参数 fmt 为单位距离的离指定日期 date 最近的日期时间值，并且返回的日期值在 date 之前。

> 注意
>
> 与函数 ROUND 的区别为，TRUNC 返回的值必须是在 date 之前的离 date 最近的日期，ROUND 可以是 date 之前也可以是 date 之后的离它最近的日期值。

语法

```
TRUNC (date,[fmt])
```

参数

参数	说明
date	DATE 数据类型。
fmt	指定了函数返回值与 date 的距离单位，以下表格列举了该参数的可取值，大小写不敏感。

fmt参数表	说明
j	默认值，最近 0 点日期。
day、dy、d	返回离指定日期最近的星期日。
month、mon、mm、rm	返回离指定日期最近的月的第一天日期。
q	返回离指定日期最近的季的日期。
yyyy、yyy、yy、y	多个 y 表示不同的精度，返回离指定日期最近的年的第一个日期。
cc、scc	返回离指定日期最近的世纪的初日期。

返回类型

`DATE` 数据类型。

示例

以下示例展示了用 `TRUNC` 计算距离 SYSDATE 最近的符合要求的日期值：

```
SELECT SYSDATE 当时日期,
TRUNC(SYSDATE) 今天日期,
TRUNC(SYSDATE,'DAY') 本周星期日,
TRUNC(SYSDATE,'MONTH') 本月初,
TRUNC(SYSDATE,'Q') 本季初日期,
TRUNC(SYSDATE,'YEAR') 本年初日期 FROM DUAL;
```

查询结果如下：

```
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
| 当时日期            | 今天日期            | 本周星期日          | 本月初              | 本季初日期          | 本年初日期          |
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
| 2020-03-08 22:41:46 | 2020-03-08 00:00:00 | 2020-03-08 00:00:00 | 2020-03-01 00:00:00 | 2020-01-01 00:00:00 | 2020-01-01 00:00:00 |
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
```

以下示例展示了在相同日期下用 `ROUND` 计算符合要求的最近日期的结果：

```
SELECT SYSDATE 当时日期,
ROUND(SYSDATE) 最近0点日期,
ROUND(SYSDATE,'DAY') 最近星期日,
ROUND(SYSDATE,'MONTH') 最近月初,
ROUND(SYSDATE,'Q') 最近季初日期,
ROUND(SYSDATE,'YEAR') 最近年初日期 FROM DUAL;
```

查询结果如下：

```
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
| 当时日期            | 最近0点日期         | 最近星期日          | 最近月初            | 最近季初日期        | 最近年初日期        |
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
| 2020-03-08 22:41:02 | 2020-03-09 00:00:00 | 2020-03-08 00:00:00 | 2020-03-01 00:00:00 | 2020-04-01 00:00:00 | 2020-01-01 00:00:00 |
+---------------------+---------------------+---------------------+---------------------+---------------------+---------------------+
```

6.6.2.4.24. TZ_OFFSET

`TZ_OFFSET` 函数返回时区 `n` 的时区偏移量。时区偏移量是指与格林尼治标准时间 GMT 的差（小时和分钟）。

语法

```
TZ_OFFSET (n)
```

参数

参数	说明
n	时区区域名称。

返回类型

`VARCHAR2` 数据类型。

示例

以下示例返回了当前会话时区 `SESSIONTIMEZONE` 、数据库时区 `DBTIMEZONE` 和 US/Eastern 所在时区的时区偏移量：

```
SELECT TZ_OFFSET(SESSIONTIMEZONE),TZ_OFFSET(DBTIMEZONE),TZ_OFFSET('US/Eastern') FROM DUAL;
```

查询结果如下：

```
+---------------------------+----------------------+-------------------------+
| TZ_OFFSET(SESSIONTIMEZONE) | TZ_OFFSET(DBTIMEZONE) | TZ_OFFSET('US/EASTERN') |
+---------------------------+----------------------+-------------------------+
| +08:00                    | +00:00               | -04:00                  |
+---------------------------+----------------------+-------------------------+
```

6.6.2.5. 通用比较函数

6.6.2.5.1. GREATEST

GREATEST 函数返回一个或多个表达式列表中的最大值。 数据库使用第一个参数 expr 来确定返回类型。如果其余参数的数据类型和第一个参数 expr 的数据类型不同，则 OceanBase 将第一个参数 expr 之后的每个参数隐式转换为比较之前的第一个 expr 的数据类型。

语法

```
GREATEST(expr [, expr ]...)
```

参数

参数	说明
expr	一个表达式或表达式列表。数据类型可以是：NUMBER 、FLOAT 、BINARY_FLOAT 、BINARY_DOUBLE 、CHAR 、VARCHAR2 、NCHAR 、NVARCHAR2 或 CLOB 。

返回类型

如果第一个参数 expr 的数据类型为 NUMBER 、 FLOAT 、 BINARY_FLOAT 或 BINARY_DOUBLE ，则返回与第一个参数 expr 相同的数据类型。如果第一个参数 expr 的数据类型为 CHAR 、 VARCHAR2 或 CLOB ，则返回 VARCHAR2 类型。如果第一个参数 expr 的数据类型为 NCHAR 或 NVARCHAR2 ，则返回 NVARCHAR2 类型。

示例

以下语句比较了字符串的大小，并返回了其中最大的字符串：

```
SELECT GREATEST('HAPPY', 'HAPPEN', 'HAPPINESS') "Greatest"
 FROM DUAL;
```

返回结果：

```
+----------+
| Greatest |
+----------+
|   HAPPY |
+----------+
```

以下语句比较了整数 1 和 字符串 3.925、2.4 三者间的大小，由于第一个参数的数据类型为数值数据类型，所以其余参数将被隐式转换为数值数据类型后再进行比较：

```
SELECT GREATEST (1, '3.935', '2.4') "Greatest"
 FROM DUAL;
```

返回结果：

```
+----------+
| Greatest |
+----------+
|   3.935 |
+----------+
```

6.6.2.5.2. LEAST

LEAST 函数返回一个或多个表达式列表中的最小值。 数据库使用第一个参数 expr 来确定返回类型。 如果其余参数的数据类型和第一个参数 expr 的数据类型不同，则 OceanBase 将第一个参数 expr 之后的每个参数隐式转换为比较之前的第一个 expr 的数据类型。

语法

```
LEAST(expr [, expr ]...)
```

参数

参数	说明
expr	一个表达式或表达式列表。数据类型可以是：NUMBER 、FLOAT 、BINARY_FLOAT 、BINARY_DOUBLE 、CHAR 、VARCHAR2 、NCHAR 、NVARCHAR2 或 CLOB 。

返回类型

如果第一个参数 expr 的数据类型为 NUMBER 、 FLOAT 、 BINARY_FLOAT 或 BINARY_DOUBLE ，则返回与第一个参数 expr 相同的数据类型。如果第一个参数 expr 的数据类型为 CHAR 、 VARCHAR2 或 CLOB ，则返回 VARCHAR2 类型。如果第一个参数 expr 的数据类型为 NCHAR 或 NVARCHAR2 ，则返回 NVARCHAR2 类型。

示例

以下语句比较了字符串的大小，并返回了其中最小的字符串：

```
SELECT LEAST('HAPPY', 'HAPPEN', 'HAPPINESS') "Least"
 FROM DUAL;
```

返回结果：

```
+--------+
| Least  |
+--------+
| HAPPEN |
+--------+
```

以下语句比较了整数 1 和 字符串 3.925、2.4 三者间的大小，由于第一个参数的数据类型为数值数据类型，所以其余参数将被隐式转换为数值数据类型后再进行比较：

```
SELECT LEAST (1, '3.925', '2.4') "Least"
 FROM DUAL;
```

返回结果：

```
+-------+
| Least |
+-------+
|     1 |
+-------+
```

6.6.2.6. 转换函数

6.6.2.6.1. CAST

CAST 函数用于将源数据类型的表达式显式转换为另一种数据类型。

语法

```
CAST (expr AS type_name )
```

参数

参数	说明
expr	列名或者表达式。
AS	用于分隔两个参数，在 `AS` 之前的是要处理的数据，在 `AS` 之后是要转换的数据类型。
type_name	数据类型为 OceanBase 的 内建数据类型 。

返回类型

返回与 `type_name` 相同的类型。

下表显示了哪些数据类型可以转换为其他内置数据类型：

	from BINARY_FLOAT, BINARY_DOUBLE	from CHAR, VARCHAR2	from NUMBER	1 from DATETIME/INTERVAL	from RAW	from NCHAR, NVARCHAR2
to BINARY_FLOAT, BINARY_DOUBLE	yes	yes	yes	no	no	yes
to CHAR, VARCHAR2	yes	yes	yes	yes	yes	no
to NUMBER	yes	yes	yes	no	no	yes
to DATETIME, INTERVAL	no	yes	no	yes	no	no
to RAW	yes	yes	yes	no	yes	no

to NCHAR, NVARCHAR2	yes	no	yes	yes	yes	yes

[1] To DATETIME/INTERVAL 数据类型包括 DATE 、 TIMESTAMP 、 TIMESTAMP WITH TIMEZONE 、 INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 。

示例

执行以下语句：

```
SELECT CAST('123' AS INT),CAST(1 AS VARCHAR2(10)),CAST('22-OCT-1997' AS TIMESTAMP WITH LOCAL TIME ZONE)
AS RESULT FROM DUAL;
```

查询结果如下：

```
+------------------+----------------------+---------------------------+
| CAST('123'ASINT) | CAST(1ASVARCHAR2(10)) | RESULT                    |
+------------------+----------------------+---------------------------+
|              123 | 1                    | 1997-10-22 00:00:00.000000 |
+------------------+----------------------+---------------------------+
```

6.6.2.6.2. HEXTORAW

HEXTORAW 函数将 CHAR 、 VARCHAR2 、 NCHAR 或 NVARCHAR2 数据类型中包含十六进制数字的字符转换为 RAW 数据类型。

语法

```
HEXTORAW (char)
```

参数

参数	说明
char	十六进制的字符串。字符串类型可为： CHAR 、 VARCHAR2 、 NCHAR 或 NVARCHAR2 。

返回类型

返回 RAW 类型的数据。

示例

执行以下语句：

```
select HEXTORAW('A123')  from dual;
```

查询结果如下：

```
+------------------+
| HEXTORAW('A123') |
+------------------+
| A123             |
+------------------+
```

6.6.2.6.3. RAWTOHEX

`RAWTOHEX` 函数将二进制数转换为一个相应的十六进制表示的字符串。

语法

```
RAWTOHEX (raw)
```

参数

参数	说明
raw	二进制的字符串。

返回类型

十六进制表示的字符串。

示例

执行以下语句：

```
SELECT RAWTOHEX('AB')  FROM DUAL;
```

查询结果如下：

```
+----------------+
| RAWTOHEX('AB') |
+----------------+
| 4142           |
+----------------+
```

6.6.2.6.4. TO_BINARY_DOUBLE

TO_BINARY_DOUBLE 函数返回一个双精度的 64 位浮点数.

语法

```
TO_BINARY_DOUBLE(expr [, fmt [, 'nlsparam' ] ])
```

参数

参数	说明
expr	字符串或 NUMBER 、BINARY_FLOAT 或 BINARY_DOUBLE 数值数据类型。

当 expr 是字符串时，可选的 fmt 和 nlsparam 参数才有效。它们的作用与 TO_CHAR(number) 功能的作用相同。当 expr 为 BINARY_DOUBLE ，则该函数返回 expr 。

返回类型

双精度的 64 位浮点数。从字符串或 NUMBER 到 BINARY_DOUBLE 的转换可能不准确，因为 NUMBER 和字符类型使用十进制精度表示数值，而 BINARY_DOUBLE 使用二进制精度。从 BINARY_FLOAT 到 BINARY_DOUBLE 的转换是准确的。

示例

执行以下语句：

```
SELECT TO_BINARY_DOUBLE(1222.111) FROM DUAL;
```

查询结果如下：

```
+---------------------------+
| TO_BINARY_DOUBLE(1222.111) |
+---------------------------+
|         1.222111E+003 |
+---------------------------+
```

您可以看 TO_BINARY_FLOAT 和 TO_CHAR(number)。

6.6.2.6.5. TO_BINARY_FLOAT

TO_BINARY_FLOAT 函数返回一个单精度的 32 位浮点数。

语法

```
TO_BINARY_FLOAT(expr [, fmt [, 'nlsparam' ] ])
```

参数

参数	说明
expr	字符串或 NUMBER 、 BINARY_FLOAT 或 BINARY_DOUBLE 数值数据类型。

当 expr 是字符串时，可选的 fmt 和 nlsparam 参数才有效。它们的作用与 TO_CHAR(number) 功能的作用相同。当 expr 为 BINARY_FLOAT ，则函数返回 expr 。

返回类型

单精度的 32 位浮点数。从字符串或 NUMBER 到 BINARY_FLOAT 的转换可能不精确，因为 NUMBER 和字符类型使用十进制精度表示数值，而 BINARY_FLOAT 使用二进制精度。如果 BINARY_DOUBLE 值使用的精度比 BINARY_FLOAT 支持的精度高，则从 BINARY_DOUBLE 到 BINARY_FLOAT 的转换是不精确的。

示例

执行以下语句：

```
SELECT TO_BINARY_FLOAT(1222.111) from dual;
```

查询结果如下：

```
+---------------------------+
| TO_BINARY_FLOAT(1222.111) |
+---------------------------+
|       1.22211096E+003 |
+---------------------------+
```

您可以看 TO_BINARY_DOUBLE 和 TO_CHAR(number)。

6.6.2.6.6. TO_CHAR (character)

TO_CHAR (character) 函数将 NCHAR 、 NVARCHAR2 或 CLOB 数据转换为数据库字符集。

语法

```
TO_CHAR(character )
```

参数

	说明
character	可以是 NCHAR 、NVARCHAR2 或 CLOB 数据类型。

返回类型

返回 VARCHAR2 。当函数将字符 LOB 转换为数据库字符集时，如果要转换的 LOB 值大于目标类型，则数据库返回错误。

示例

新建表 CLOBTEST，并给 RAW 数据类型的列插入数据。

```
CREATE TABLE CLOBTEST(TEXT CLOB);
INSERT INTO CLOBTEST VALUES('DWUIDBWUIDBWIOBFWUIOBFIOWBFWUIOBFUWIFB') ;
```

执行以下语句：

```
SELECT TO_CHAR(TEXT) FROM CLOBTEST;
```

查询结果如下：

```
+----------------------------------------+
| TO_CHAR(TEXT)                |
+----------------------------------------+
| dwuidbwuidbwiobfwuiobfiowbfwuiobfuwifb |
+----------------------------------------+
```

6.6.2.6.7. TO_CHAR (datetime)

TO_CHAR 函数将 DATE 、TIMESTAMP 、TIMESTAMP WITH TIME ZONE 、TIMESTAMP WITH LOCAL TIME ZONE 、INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 等数据类型的值按照参数 fmt 指定的格式转换为 VARCHAR2 数据类型的值。如果不指定参数 fmt ，则参数 datetime 的值将按如下格式转换为 VARCHAR2 数据类型：

- DATE 、TIMESTAMP 、TIMESTAMP WITH TIME ZONE 和 TIMESTAMP WITH LOCAL TIME ZONE 的值被转换为数据库中日期时间值的默认格式。您可在数据类型章节中查看各日期时间类型的默认格式。
- INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 数据类型的值转换为数字格式的间隔值。

语法

```
TO_CHAR({ datetime | interval } [, fmt [, 'nlsparam' ] ])
```

参数

参数	说明
datetime	DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONE、INTERVAL DAY TO SECOND 和 INTERVAL YEAR TO MONTH 等数据类型的值。
fmt	输出格式参数。
nlsparam	用来控制返回的月份和日份所使用的语言。

返回类型

VARCHAR2 数据类型。

示例

示例 1：以下语句通过 TO_CHAR 函数返回系统当前日期，并且将日期时间值转换为了 DS DL 格式：

```
SELECT TO_CHAR(SYSDATE,'DS DL') FROM DUAL;
```

查询结果如下：

```
+----------------------------------+
| TO_CHAR(SYSDATE,'DSDL')          |
+----------------------------------+
| 03/08/2020 Sunday, March 08, 2020 |
+----------------------------------+
```

示例 2：以下语句将间隔值转化为指定格式，并且设置了返回语言为 AMERICAN：

```
SELECT TO_CHAR(interval'1' year, 'SS-MI-HH', 'nls_language = AMERICAN') FROM DUAL;
```

查询结果如下：

```
+-----------------------------------------------------------+
| TO_CHAR(INTERVAL'1'YEAR,'SS-MI-HH','NLS_LANGUAGE=AMERICAN') |
+-----------------------------------------------------------+
| +01-00                                                    |
+-----------------------------------------------------------+
```

6.6.2.6.8. TO_CHAR (number)

TO_CHAR 函数将 NUMBER 、 BINARY_FLOAT 或 BINARY_DOUBLE 类型的数值 n 按照指定数值格式 fmt 转换为 varchar2 数据类型的值。如果省略 fmt ，则 n 被转换为 VARCHAR2 值，其长度足以保持有效数字。若 n 为负值，则负号显示在输出值最左侧，比如 TO_CHAR(-1,’$9’) 返回 -$1 ，而不是 $-1 。

语法

```
TO_CHAR(n [, fmt [, 'nlsparam' ] ])
```

参数

参数	说明
n	精确数值或近似数值数据类型类别的表达式。数值数据类型可以为 NUMBER 、 BINARY_FLOAT 或 BINARY_DOUBLE 数值数据类型。
fmt	输出格式参数。
nlsparam	支持的语言从 sys.V_$NLS_VALID_VALUES 获取。

fmt 参数列表

fmt 参数	说明
9	返回指定位数的值。
0	它返回前导 0，它返回尾随 0。
,（逗号）	返回指定位置的逗号。您可以在数字格式化中指定多个逗号。 **限制条件**：格式化数值不能以逗号开头，且逗号不能出现在小数字符或句点的右边。
.（小数点）	返回一个小数，且小数点在指定位置。 **限制条件**：在数字格式化中，您只能指定一个小数点。

返回类型

VARCHAR2 数据类型。

示例

执行以下语句：

```
SELECT TO_CHAR(123.456,'999') FROM DUAL;
```

查询结果如下：

```
+-----------------------+
| TO_CHAR(123.456,'999') |
+-----------------------+
| 123                   |
+-----------------------
```

6.6.2.6.9. TO_DATE

TO_DATE 函数将 CHAR 、 VARCHAR 、 NCHAR 或 NVARCHAR2 数据类型的字符转换为日期数据类型的值。

语法

```
TO_DATE(char [, fmt [, 'nlsparam' ] ])
```

参数

参数	说明
char	CHAR 、 VARCHAR 、 NCHAR 或 NVARCHAR2 字符数据。
fmt	指定日期时间格式化。
nlsparam	由 nls_territory 初始化参数隐式指定或 nls_date_format 参数显式指定。

返回类型

返回 DATE 类型。

示例

执行以下语句：

```
SELECT TO_DATE('199912','YYYYMM'),TO_DATE('2000.05.20','YYYY.MM.DD'),
(DATE '2008-12-31') XXDATE,
TO_DATE('2008-12-31 12:31:30','YYYY-MM-DD HH24:MI:SS'),
(TIMESTAMP '2008-12-31 12:31:30') XXTIMESTAMP
FROM DUAL;
```

查询结果如下：

```
+--------------------------+----------------------------------+--------------------+-----------------------------------------------------
+------------------------------+
| TO_DATE('199912','YYYYMM') | TO_DATE('2000.05.20','YYYY.MM.DD') | XXDATE          | TO_DATE('2008-12-311
2:31:30','YYYY-MM-DDHH24:MI:SS') | XXTIMESTAMP              |
+--------------------------+----------------------------------+--------------------+-----------------------------------------------------
+------------------------------+
| 1999-12-01 00:00:00      | 2000-05-20 00:00:00           | 2008-12-31 00:00:00 | 2008-12-31 12:31:30
| 2008-12-31 12:31:30.000000000 |
+--------------------------+----------------------------------+--------------------+-----------------------------------------------------
+------------------------------+
```

6.6.2.6.10. TO_DSINTERVAL

`TO_DSINTERVAL` 函数将一个 `CHAR`、`VARCHAR2`、`NCHAR` 或 `NVARCHAR2` 数据类型的字符串转换为一个 `INTERVAL DAY TO SECOND` 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。

语法

```
TO_DSINTERVAL (days hours：minutes：seconds[.frac_secs])
```

参数

参数	说明
days hours：minutes：seconds[.frac_secs]	符合该参数格式的 `CHAR`、`VARCHAR2`、`NCHAR` 或 `NVARCHAR2` 数据类型的字符串。

返回类型

`INTERVAL DAY TO SECOND` 数据类型。

示例

以下示例返回了当前时间 100 天后的日期时间值：

```
SELECT SYSDATE, SYSDATE+TO_DSINTERVAL('100 00:00:00') FROM DUAL;
```

查询结果如下：

```
+---------------------+-------------------------------------+
| SYSDATE             | SYSDATE+TO_DSINTERVAL('10000:00:00') |
+---------------------+-------------------------------------+
| 2020-03-26 12:40:39 | 2020-07-04 12:40:39                 |
+---------------------+-------------------------------------+
```

6.6.2.6.11. TO_NUMBER

`TO_NUMBER` 函数将 `expr` 转换为数值数据类型的值。`expr` 可以是 `CHAR`、`VARCHAR2`、`NCHAR`、`NVARCHAR2`、`BINARY_FLOAT` 或 `BINARY_DOUBLE` 数据类型的数值。

语法

```
TO_NUMBER(expr [, fmt [, 'nlsparam' ] ])
```

参数

参数	说明
expr	`CHAR`、`VARCHAR2`、`NCHAR`、`NVARCHAR2`、`BINARY_FLOAT` 或 `BINARY_DOUBLE` 数据类型的数值。
fmt	格式模型。
nlsparam	支持的语言从 `sys.V_$NLS_VALID_VALUES` 获取。

返回类型

返回 `NUMBER` 类型的数据。

示例

执行以下语句：

```
SELECT TO_NUMBER('199912'),TO_NUMBER('450.05') FROM DUAL;
```

查询结果如下：

```
+--------------------+--------------------+
| TO_NUMBER('199912') | TO_NUMBER('450.05') |
+--------------------+--------------------+
|             199912 |             450.05 |
+--------------------+--------------------+
```

6.6.2.6.12. TO_TIMESTAMP

`TO_TIMESTAMP` 函数将字符串转换为 `TIMESTAMP` 数据类型。

语法

```
TO_TIMESTAMP (char，[fmt],['nlsparam'])
```

参数

参数	说明
char	`CHAR` 、`VARCHAR2` 、`NCHAR` 或 `NVARCHAR2` 数据类型的字符串。
fmt	指定返回值的格式。
nlsparam	用来控制返回的月份和日份所使用的语言。

返回类型

`TIMESTAMP` 数据类型。

示例

执行以下语句：

```
SELECT TO_TIMESTAMP ('10-Sep-02 14:10:10.123000', 'DD-Mon-RR HH24:MI:SS.FF')
FROM DUAL;
```

查询结果如下：

```
+-----------------------------------------------------------------+
| TO_TIMESTAMP('10-SEP-0214:10:10.123000','DD-MON-RRHH24:MI:SS.FF') |
+-----------------------------------------------------------------+
| 2002-09-10 14:10:10.123000000                                   |
+-----------------------------------------------------------------+
```

6.6.2.6.13. TO_TIMESTAMP_TZ

`TO_TIMESTAMP` 函数将字符串转换为 `TIMESTAMP WITH TIME ZONE` 数据类型，包含时区信息。

语法

```
TO_TIMESTAMP_TZ (char，[fmt],['nlsparam'])
```

参数

参数	说明
char	`CHAR` 、`VARCHAR2` 、`NCHAR` 或 `NVARCHAR2` 数据类型的字符串。
fmt	指定输出格式。
nlsparam	用来控制返回的月份和日份所使用的语言。

返回类型

`TIMESTAMP WITH TIME ZONE` 数据类型。

示例

执行以下语句：

```
SELECT TO_TIMESTAMP_TZ ('10-Sep-02 14:10:10.123000', 'DD-Mon-RR HH24:MI:SS.FF') FROM DUAL;
```

查询结果如下：

```
+-----------------------------------------------------------------+
| TO_TIMESTAMP_TZ('10-SEP-0214:10:10.123000','DD-MON-RRHH24:MI:SS.FF') |
+-----------------------------------------------------------------+
| 2002-09-10 14:10:10.123000000 +08:00                            |
+-----------------------------------------------------------------+
```

6.6.2.6.14. TO_YMINTERVAL

`TO_YMINTERVAL` 函数将一个 `CHAR` 、 `VARCHAR2` 、 `NCHAR` 或 `NVARCHAR2` 数据类型的字符串转换为一个 `INTERVAL YEAR TO MONTH` 数据类型的值，该函数可以用来对一个日期时间值进行加减计算。

语法

```
TO_YMINTERVAL (years-months)
```

参数

参数	说明
years-months	符合该参数格式的 `CHAR` 、 `VARCHAR2` 、 `NCHAR` 或 `NVARCHAR2` 数据类型的字符串。

返回类型

`INTERVAL YEAR TO MONTH` 数据类型。

示例

以下示例展示了返回当前时间 1 年 2 个月后的时间日期值：

```
SELECT SYSDATE,SYSDATE+TO_YMINTERVAL('01-02') FROM DUAL;
```

查询结果如下：

```
+---------------------+-------------------------------+
| SYSDATE             | SYSDATE+TO_YMINTERVAL('01-02') |
+---------------------+-------------------------------+
| 2020-03-08 22:32:01 | 2021-05-08 22:32:01           |
+---------------------+-------------------------------+
```

6.6.2.7. 编码解码函数

6.6.2.7.1. DECODE

`DECODE` 函数依次用参数 `search` 与 `condition` 做比较，直至 `condition` 与 `search` 的值相等，则返回对应 `search` 后跟随的参数 `result` 的值。如果没有 `search` 与 `condition` 相等，则返回参数 `default` 的值。

语法

```
DECODE (condition, search 1, result 1, search 2, result 2 ... search n, result n, default)
```

DECODE 函数的含义可以用 IF...ELSE IF...END 语句进行解释：

```
IF condition = search 1 THEN
RETURN(result 1)
ELSE IF condition = search 2 THEN
RETURN(result 2)
......
ELSE IF condition = search n THEN
RETURN(result n)
ELSE
RETURN(default)
END IF
```

参数

参数	说明
condition、search 1...search n、result 1...result n、default	数值型（NUMBER、FLOAT、BINARY_FLOAT 或 BINARY_DOUBLE）或字符型（CHAR、VARCHAR2、NCHAR 或 NVARCHAR2）的表达式。

注意

search 1 ~ search n 不能为条件表达式，这种情况只能用 CASE WHEN THEN END 语句解决：

```
WHEN CASE condition = search 1 THEN
RETURN(result 1)
ELSE CASE condition = search 2 THEN
RETURN(result 2)
......
ELSE CASE condition = search n THEN
RETURN(result n)
ELSE
RETURN(default)
END
```

示例

示例 1： 使用 `DECODE` 比较数值的大小。

以下语句使用 `DECODE` 函数来返回数值 10 与 20 当中较小的那个数。`SIGN()` 函数用来计算两个值差的符号，由于 10 小于 20，所以差为负数，`SIGN()` 返回 -1。此时 `DECODE` 函数将参数 -1 与 `SIGN()` 函数的返回值作比较。相等时，返回值 10，不相等时，返回值 20：

```
SELECT DECODE(SIGN(10-20),-1,10,20) FROM DUAL;
```

返回结果如下：

```
+------------------------------+
| DECODE(SIGN(10-20),-1,10,20) |
+------------------------------+
|                           10 |
+------------------------------+
```

示例 2： 使用 `DECODE` 函数查看数据中是否包含字符 S。

以下语句创建了表 EMP，表中包含列 ename 和 sal，并向其中插入值：

```
CREATE TABLE EMP(ename VARCHAR(30),sal NUMBER);
INSERT INTO EMP VALUES('CLARK', 2750);
INSERT INTO EMP VALUES('KING', 5300);
INSERT INTO EMP VALUES('MILLER', 1600);
INSERT INTO EMP VALUES('ADAMS', 1400);
INSERT INTO EMP VALUES('FORD', 3300);
INSERT INTO EMP VALUES('JONES', 3275);
INSERT INTO EMP VALUES('SCOTT', 3300);
INSERT INTO EMP VALUES('SMITH', 1100);
INSERT INTO EMP VALUES('ALLEN', 1900);
INSERT INTO EMP VALUES('BLAKE', 3150);
INSERT INTO EMP VALUES('JAMES', 1250);
INSERT INTO EMP VALUES('MARTIN', 1550);
INSERT INTO EMP VALUES('TURNER', 1800);
INSERT INTO EMP VALUES('WARD', 1550);
```

以下语句通过函数 `INSTR()` 返回字符 S 在列 ename 的值中出现的位置，若没有出现过则返回 0。此时 `DECODE` 函数将 `INSTR` 函数的返回值与 0 做比较，相等时说明字符 S 在值中没有出现过，则 `DECODE` 函数返回值 不含有 S，否则返回值 含有 S：

```
SELECT ENAME, SAL, DECODE(INSTR(ename, 'S'), 0, '不含有 S', '含有 S') AS INFO FROM EMP;
```

查询结果如下：

```
+--------+------+------------+
| ENAME | SAL | INFO     |
+--------+------+------------+
| CLARK | 2750 | 不含有 S   |
| KING  | 5300 | 不含有 S   |
| MILLER | 1600 | 不含有 S  |
| ADAMS | 1400 | 含有 S     |
| FORD  | 3300 | 不含有 S   |
| JONES | 3275 | 含有 S     |
| SCOTT | 3300 | 含有 S     |
| SMITH | 1100 | 含有 S     |
| ALLEN | 1900 | 不含有 S   |
| BLAKE | 3150 | 不含有 S   |
| JAMES | 1250 | 含有 S     |
| MARTIN | 1550 | 不含有 S  |
| TURNER | 1800 | 不含有 S  |
| WARD  | 1550 | 不含有 S   |
+--------+------+------------+
```

6.6.2.7.2. ORA_HASH

`ORA_HASH` 函数获取对应表达式的 HASH 值。

语法

```
ORA_HASH(expr [, max_bucket [, seed_value ] ])
```

参数

参数	说明
expr	通常为数据库表的列名，数据类型可以是数值类型、字符类型、日期时间类型 或 `RAW` 类型。
max_bucket	可选的 `max_bucket` 参数确定哈希函数返回的最大桶数。取值范围为 0~4294967295，默认值是 4294967295。

参数	说明
seed_value	可选的 `seed_value` 参数使 OceanBase 能够为同一组数据产生许多不同的结果。 您可以指定 0~4294967295 之间的任何值。默认值为 0。

返回类型

`NUMBER` 类型数据。

示例

创建表 SALE，并向里面插入数据。执行以下语句：

```
CREATE TABLE SALE(MONTH CHAR(6), SELL NUMBER(10,2));
INSERT INTO SALE VALUES(200001, 1000);
INSERT INTO SALE VALUES(200002, 1100);
INSERT INTO SALE VALUES(200003, 1200);
INSERT INTO SALE VALUES(200004, 1300);
INSERT INTO SALE VALUES(200005, 1400);
INSERT INTO SALE VALUES(200006, 1500);
INSERT INTO SALE VALUES(200007, 1600);
INSERT INTO SALE VALUES(200101, 1100);
INSERT INTO SALE VALUES(200202, 1200);
INSERT INTO SALE VALUES(200301, 1300);
```

使用 `ORA_HASH` 函数查询 SALE 表，并执行以下语句：

```
SELECT ORA_HASH(CONCAT(month,sell),12,0), month, sell FROM Sale;
```

查询结果如下：

```
+----------------------------------+--------+------+
| ORA_HASH(CONCAT(MONTH,SELL),12,0) | MONTH  | SELL |
+----------------------------------+--------+------+
|                                1 | 200001 | 1000 |
|                                6 | 200002 | 1100 |
|                                5 | 200003 | 1200 |
|                                4 | 200004 | 1300 |
|                                5 | 200005 | 1400 |
|                                2 | 200006 | 1500 |
|                                7 | 200007 | 1600 |
|                               10 | 200101 | 1100 |
|                                7 | 200202 | 1200 |
|                                4 | 200301 | 1300 |
+----------------------------------+--------+------+
```

6.6.2.7.3. VSIZE

`VSIZE` 函数返回 `X` 的字节大小数。

语法

```
VSIZE(X)
```

返回类型

返回 x 的字节数，如果 x 为 `NULL`，则函数返回 `NULL`。

示例

创建 employees，并向里面插入数据。执行以下语句：

```
CREATE TABLE employees(manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(300, 'Wei', '2019-09-11',23600);
INSERT INTO employees VALUES(200, 'Red', '2019-11-05', 23800);
INSERT INTO employees VALUES(100, 'Part', '2018-10-01',24000);
INSERT INTO employees VALUES(200, 'Ross', '2019-06-11',23500);
COMMIT;
```

使用 `VSIZE` 函数查询 last_name 列中 manager_id = 300 的字节数。执行以下语句：

```
SELECT last_name, VSIZE (last_name) "BYTES" FROM employees WHERE manager_id = 300;
```

查询结果如下：

```
+-----------+-------+
| LAST_NAME | BYTES |
+-----------+-------+
| Wei       |     3 |
+-----------+-------+
```

6.6.2.8. 空值相关函数

6.6.2.8.1. COALESCE

`COALESCE` 函数返回参数列表中第一个非空表达式，必须指定最少两个参数。

语法

```
COALESCE(expr1, expr2[,…, exprn])
```

参数

参数	说明
expr1, expr2[,..., exprn]	非空表达式，且最少 2 个。

返回类型

返回参数列表中第一个非空表达式，如果所有的参数都是 `NULL`，则返回 `NULL`。

示例

假设有一张表 product_information，product_id 为商品 ID，list_price 为该商品原价，min_price 为商品最低价，Sale 为商品实际售价。设置商品折扣为 9 折，计算各商品的实际售价。此时可使用 `COALESCE` 函数，若 list_price 为空，就按最低价 min_price 计算；若 min_price 也为空，则按 5 计算。

您可以执行以下语句，建立 product_information 数据表，并插入数据：

```
CREATE TABLE product_information(supplier_id INT, product_id INT,list_price numeric, min_price numeric);
INSERT INTO PRODUCT_INFORMATION VALUES ('102050', '1659', '45', NULL);
INSERT INTO PRODUCT_INFORMATION VALUES ('102050', '1770', NULL, '70');
INSERT INTO PRODUCT_INFORMATION VALUES ('102050', '2370', '305', '247');
INSERT INTO PRODUCT_INFORMATION VALUES ('102050', '2380', '750', '731');
INSERT INTO PRODUCT_INFORMATION VALUES ('102050', '3255', NULL, NULL);
```

执行以下查询语句：

```
SELECT product_id, list_price,min_price,COALESCE(0.9*list_price, min_price, 5) "Sale"
FROM product_information WHERE supplier_id = 102050 ORDER BY product_id;
```

查询结果如下：

```
+-------------+-------------+-----------+--------+
| PRODUCT_ID | LIST_PRICE | MIN_PRICE | Sale |
+-------------+-------------+-----------+--------+
|    1659 |    45    |      | 40.5 |
+-------------+-------------+-----------+--------+
|    1770 |        |    70    |  70 |
+-------------+-------------+-----------+--------+
|    2370 |    305    |  247    |274.5 |
+-------------+-------------+-----------+--------+
|    2380 |    750    |  731    |  675 |
+-------------+-------------+-----------+--------+
|    3255 |        |      |  5 |
+-------------+-------------+-----------+--------+
```

6.6.2.8.2. LNNVL

`LNNVL` 函数判断条件中的一个或者两个操作数是否为 `NULL` 。该函数可以在 `WHERE` 子句中使用，也可以作为 `CASE` 表达式中的 `WHEN` 条件。将条件作为参数，如果条件为 `FALSE` 或 `UNKNOWN` ，则返回 `TRUE` ；如果条件为 `TRUE` ，则返回 `FALSE` 。

语法

```
LNNVL(condition)
```

参数

参数	说明
condition	条件

假设 a = 2，b 值为 NULL，下表显示了 `LNNVL` 函数的返回值。

条件	条件判断结果	LNNVL 返回值
a = 1	FALSE	TRUE
a = 2	TRUE	FALSE
a IS NULL	FALSE	TRUE
b = 1	UNKNOWN	TRUE
b IS NULL	TRUE	FALSE
a = b	UNKNOWN	TRUE

返回类型

返回布尔型 `TRUE` 或 `FALSE` 。

示例

假设有一张表 EMPLOYEES，给员工姓名列 name 和佣金列 commission_pct 里面插入数据，执行以下语句：

```
CREATE TABLE EMPLOYEES (name VARCHAR(20), commission_pct numeric);
INSERT INTO EMPLOYEEs VALUES ('Baer', null);
INSERT INTO EMPLOYEEs VALUES ('Bada', null);
INSERT INTO EMPLOYEEs VALUES ('Boll', 0.1);
INSERT INTO EMPLOYEEs VALUES ('Bates', 0.15);
INSERT INTO EMPLOYEEs VALUES ('Eros', null);
INSERT INTO EMPLOYEEs VALUES ('Girl', 0.25);
```

您想知道佣金率低于 20% 的员工人数，包括没有收到佣金的员工。执行以下语句，您只能查询实际获得佣金低于 20% 的员工人数：

```
SELECT COUNT(*) FROM employees WHERE commission_pct < .2;
```

查询结果如下：

```
+------------+
| COUNT(*) |
+------------+
|    2 |
+------------+
```

要包括没有收到佣金的另外 3 名员工，您需要使用 INNVL 函数重写查询。执行以下语句：

```
SELECT COUNT(*) FROM employees WHERE LNNVL(commission_pct >= .2);
```

查询结果如下：

```
+------------+
| COUNT(*) |
+------------+
|   4   |
+------------+
```

6.6.2.8.3. NVL

NVL 函数从两个表达式返回一个非 NULL 值。如果 expr1 与 expr2 的结果皆为 NULL 值，则 NVL 函数返回 NULL 。

语法

```
NVL(expr1, expr2)
```

参数

参数	说明
expr1	表达式。数据类型可以是 OceanBase 内建数据类型 中的任何数据类型。
expr2	表达式。数据类型可以是 OceanBase 内建数据类型中的任何数据类型。

expr1 和 expr2 必须是相同类型，或者可以隐式转换为相同类型。如果它们不能隐式转换，则 OceanBase 返回错误。隐式转换实现如下：

- 如果 expr1 是 CHAR 、 NCHAR 、 NVARCHAR 、 VARCHAR2 或 VARCHAR 字符型数据，则 OceanBase 在比较 expr1 之前将 expr2 转换为 expr1 的数据类型，并返回 expr1 字符集的 VARCHAR2 。
- 如果 expr1 是 NUMBER 、 FLOAT 、 BINARY_FLOAT 或 BINARY_DOUBLE 数值型数据，则 OceanBase 确定哪个参数具有最高的数字优先级，隐式地将另一个参数转换为该数据类型，并返回该数据类型。

返回类型

如果 expr1 、 expr2 是 NULL ，则返回 NULL 。如果 expr1 是 CHAR 、 NCHAR 、 NVARCHAR 、 VARCHAR2 或 VARCHAR 字符型数据，则返回 expr1 字符集的 VARCHAR2 。如果 expr1 是 NUMBER 、 FLOAT 、 BINARY_FLOAT 或 BINARY_DOUBLE 数值型数据，则返回 expr1 中具有最高数字优先级的数据类型。

示例

假设有一张表 EMPLOYEES，给员工姓名列 name 和佣金列 commission_pct 里面插入数据，执行以下语句：

```
CREATE TABLE EMPLOYEES (name VARCHAR(20),commission_pct float(3));
INSERT INTO EMPLOYEEs VALUES ('Baer', null);
INSERT INTO EMPLOYEEs VALUES ('Bada', null);
INSERT INTO EMPLOYEEs VALUES ('Boll', 0.1);
INSERT INTO EMPLOYEEs VALUES ('Bates', 0.15);
INSERT INTO EMPLOYEEs VALUES ('Eric', null);
```

查询员工的姓名和佣金，如果员工没有收到佣金，则显示 Not Applicable。执行以下语句：

```
SELECT name, NVL(TO_CHAR(commission_pct), 'Not Applicable') commission
FROM employees WHERE name LIKE 'B%' ORDER BY name;
```

查询结果如下：

```
+--------------+----------------+
|    NAME |    COMMISSION |
+--------------+----------------+
|    Baer | Not Applicable |
+--------------+----------------+
|    Bada | Not Applicable |
+--------------+----------------+
|    Boll |         .1 |
+--------------+----------------+
|    Bates |        .15 |
+--------------+----------------+
```

6.6.2.8.4. NVL2

NVL2 函数根据表达式是否为空，返回不同的值。如果 expr1 不为空，则返回 expr2 的值，如果 expr1 为空，则返回 expr3 的值。 expr2 和 expr3 类型不同的话， expr3 会转换为 expr1 的类型。

语法

```
NVL2(expr1, expr2, expr3)
```

参数

参数	说明
expr1	表达式。数据类型可以是 OceanBase 内建数据类型 中的任何数据类型。
expr2	表达式。数据类型可以是 OceanBase 内建数据类型中的任何数据类型。
expr3	表达式。数据类型可以是 OceanBase 内建数据类型中的任何数据类型。

如果 expr2 和 expr3 的数据类型不同，则 OceanBase 将其中一个隐式转换为另一个。 如果它们不能隐式转换，则数据库返回错误。 如果 expr2 是字符或数字数据，则隐式转换规则如下：

- 如果 expr2 是 CHAR 、 NCHAR 、 NVARCHAR 、 VARCHAR2 或 VARCHAR 字符型数据，则 OceanBase 在返回值之前将 expr3 转换为 expr2 的数据类型，除非 expr3 是 NULL 。 在这种情况下，不需要数据类型转换，数据库返回 expr2 字符集的 VARCHAR2 。
- 如果 expr2 是 NUMBER 、 FLOAT 、 BINARY_FLOAT 或 BINARY_DOUBL 数值型数据，则 OceanBase 确定哪个参数具有最高的数值优先级，隐式地将另一个参数转换为该数据类型，并返回该数据类型。

返回类型

如果 expr1 、 expr2 是 NULL ，则返回 NULL 。如果 expr1 是 CHAR 、 NCHAR 、 NVARCHAR 、 VARCHAR2 或 VARCHAR 字符型数据，则返回 expr1 字符集的 VARCHAR2 。如果 expr1 是 NUMBER 、 FLOAT 、 BINARY_FLOAT 或 BINARY_DOUBLE 数值型数据，则返回 expr1 中具有最高数字优先级的数据类型。

示例

假设有一张表 EMPLOYEES，给员工姓名列 name、薪资列 salary 和佣金列 commission_pct 里面插入数据，执行以下语句：

```
CREATE TABLE EMPLOYEES (name VARCHAR(20),commission_pct numeric);
INSERT INTO EMPLOYEEs VALUES ('Baer', 10000, null);
INSERT INTO EMPLOYEEs VALUES ('Bada', 2800, null);
INSERT INTO EMPLOYEEs VALUES ('Boll', 5600, .25);
INSERT INTO EMPLOYEEs VALUES ('Bates', 7300, .39);
INSERT INTO EMPLOYEEs VALUES ('Broll', 4000, null);
```

使用 `NVL2` 函数查询员工的总收入。如果员工的 commission_pct 列不为空，则员工的收入为工资加佣金，否则，收入仅仅为工资。执行以下语句：

```
SELECT name, salary,NVL2(commission_pct, salary + (salary * commission_pct), salary) income
FROM employees WHERE name like 'B%' ORDER BY name;
```

查询结果如下：

```
+--------------+---------------+---------------+
|    NAME |    SALARY |    INCOME |
+--------------+---------------+---------------+
|    Bear |    10000 |    10000 |
+--------------+---------------+---------------+
|    Bada |    2800 |    2800 |
+--------------+---------------+---------------+
|    Boll |    5600 |    7280 |
+--------------+---------------+---------------+
|    Bates |    7300 |    10220 |
+--------------+---------------+---------------+
|    Broll |    4000 |    4000 |
+--------------+---------------+---------------+
```

6.6.3. 聚合函数

6.6.3.1. AVG

`AVG` 函数返回数值列的平均值。

语法

```
AVG([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	查询时去除数据中的重复值，且忽略数据中的 NULL 值。
UNIQUE	查询时去除数据中的重复值，且忽略数据中的 NULL 值。
ALL	查询时不去除数据中的重复值，且忽略数据中的 NULL 值。`ALL` 为默认值。
expr	是数值类型或者可以转换成数值类型的表达式。数值类型可以为：`NUMBER`、`FLOAT`、`BINARY_FLOAT` 或 `BINARY_DOUBLE`。
OVER	使用 `OVER` 子句定义窗口进行计算。

> **注意**
>
> 如果您指定了 `DISTINCT` 或 `UNIQUE` 关键字，则 `analytic_clause` 中不允许出现 `order_by_clause` 和 `windowing_clause`。

返回类型

返回类型与参数 `expr` 的数据类型相同。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT, last_name varchar(50), hiredate varchar(50), SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

执行以下语句计算各列的平均值：

```
SELECT manager_id, last_name, hiredate, salary, AVG(salary) OVER (PARTITION BY manager_id
ORDER BY hiredate ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS c_mavg
FROM employees ORDER BY manager_id, hiredate, salary;
```

返回结果如下：

```
+------------+-----------+------------+--------+-----------------------------------------+
| MANAGER_ID | LAST_NAME | HIREDATE   | SALARY | C_MAVG                                  |
+------------+-----------+------------+--------+-----------------------------------------+
|        100 | Errazuriz | 2017-07-21 |   1400 |                                  1550 |
|        100 | Raphaely  | 2017-07-22 |   1700 |                                  4700 |
|        100 | De Haan   | 2018-05-01 |  11000 |                                  8900 |
|        100 | Partners  | 2018-12-01 |  14000 |                                 13000 |
|        100 | Hartstein | 2019-05-01 |  14000 |                              13833.333 |
|        100 | Weiss     | 2019-07-11 |  13500 |                                 13500 |
|        100 | Russell   | 2019-10-05 |  13000 |                                 13250 |
|        200 | Part      | 2018-08-11 |  14000 |                                 13500 |
|        200 | Bell      | 2019-05-25 |  13000 |                                 13500 |
|        200 | Ross      | 2019-06-11 |  13500 |                                 13250 |
+------------+-----------+------------+--------+-----------------------------------------+
```

聚合函数示例

执行以下语句计算 salary 的平均值：

```
SELECT AVG(salary) FROM employees;
```

查询结果如下：

```
+-----------------------------------------+
| AVG(SALARY)                             |
+-----------------------------------------+
| 10072.727272727272727272727272727272727273 |
+-----------------------------------------+
```

6.6.3.2. COUNT

`COUNT` 函数用于查询 `expr` 的行数。

语法

```
COUNT({ * | [ DISTINCT | UNIQUE | ALL ] expr }) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
*	表示满足条件的所有行，且包含值为 NULL 的行。
DISTINCT	返回的行中去除重复行，且忽略值为 NULL 的行。
UNIQUE	返回的行中去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	是数值类型或者可以转换成数值类型的表达式。数值类型可以为：`NUMBER` 、`FLOAT` 、`BINARY_FLOAT` 或 `BINARY_DOUBLE` 。
OVER	使用 OVER 子句定义窗口进行计算。

注意

- 对于 COUNT 函数，从不返回 NULL，如果指定了 expr，即返回 expr 不为 NULL 的统计个数，如果指定 COUNT(*) 返回所有行的统计数目。使用参数 DISTINCT 、UNIQUE 或 ALL 时需要与 expr 用空格隔开。
- 如果您指定了 DISTINCT 或 UNIQUE 关键字，则 analytic_clause 中不允许出现 order_by_clause 和 windowing_clause 。

返回类型

返回类型与参数 expr 的数据类型相同。

示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees(manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(300, 'Wei',  '2019-09-11',23600);
INSERT INTO employees VALUES(200, 'Red', '2019-11-05', 23800);
INSERT INTO employees VALUES(100, 'Part',  '2018-10-01',24000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',23500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 23000);
INSERT INTO employees VALUES(200, 'Part',  '2018-06-11',24500);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
COMMIT;
```

分析函数示例

执行以下语句查询表中的行数：

```
SELECT last_name, salary,COUNT(*) OVER (ORDER BY salary RANGE BETWEEN 50 PRECEDING
AND 150 FOLLOWING) AS mov_count FROM employees ORDER BY salary, last_name;
```

查询结果如下：

```
+-----------+--------+-----------+
| LAST_NAME | SALARY | MOV_COUNT |
+-----------+--------+-----------+
| Errazuriz |  1400 |      1 |
| De Haan   | 11000 |      1 |
| Hartstein | 14000 |      1 |
| Bell      | 23000 |      1 |
| Ross      | 23500 |      2 |
| Wei       | 23600 |      1 |
| Red       | 23800 |      1 |
| Part      | 24000 |      1 |
| Part      | 24500 |      1 |
+-----------+--------+-----------+
```

聚合函数示例

创建表 a，并向里面插入数据。执行以下语句：

```
CREATE TABLE a (
b INT
);
INSERT INTO a VALUES (1);
INSERT INTO a VALUES (null);
INSERT INTO a VALUES (null);
INSERT INTO a VALUES (1);
INSERT INTO a VALUES (null);
INSERT INTO a VALUES (1);
INSERT INTO a VALUES (1);
```

返回表 a 值不为 NULL 的统计个数，执行以下语句：

```
SELECT COUNT(b) FROM a;
```

查询结果如下：

```
+----------+
| COUNT(B) |
+----------+
|     4 |
+----------+
```

指定 `COUNT(*)` 返回所有行的统计数目，执行以下语句：

```
SELECT COUNT(*) FROM a;
```

查询结果如下：

```
+----------+
| COUNT(*) |
+----------+
|     7|
+----------+
```

6.6.3.3. SUM

SUM 函数返回参数中指定列的和。此函数将可以隐式转换为数值数据类型的任何数值数据类型或任何非数值数据类型作为参数。 函数返回与参数的数值数据类型相同的数据类型。

语法

```
SUM([ DISTINCT | UNQIUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 OVER 关键字。

参数

参数	说明
DISTINCT	去除重复行，且忽略值为 NULL 的行。
UNQIUE	去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 OVER 子句定义窗口进行计算。

> 注意
>
> :
>
> 如果您指定了 DISTINCT 或 UNIQUE 关键字，则 analytic_clause 中不允许出现 order_by_clause 和 windowing_clause 。

返回类型

返回与 expr 相同数据类型的值。

示例

分析函数示例

创建表 employees，并向里面插入数据，执行以下语句：

```
CREATE TABLE employees(manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(300, 'Wei', '2019-09-11',23600);
INSERT INTO employees VALUES(200, 'Red', '2019-11-05', 23800);
INSERT INTO employees VALUES(100, 'Part', '2018-10-01',24000);
INSERT INTO employees VALUES(200, 'Ross', '2019-06-11',23500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 23000);
INSERT INTO employees VALUES(200, 'Part', '2018-06-11',24500);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
COMMIT;
```

计算工资总额，执行以下语句：

```
SELECT manager_id, last_name, salary, SUM(salary) OVER (PARTITION BY manager_id
ORDER BY salary RANGE UNBOUNDED PRECEDING) l_csum
FROM employees ORDER BY manager_id, last_name, salary, l_csum;
```

查询结果如下：

```
+------------+-----------+--------+--------+
| MANAGER_ID | LAST_NAME | SALARY | L_CSUM |
+------------+-----------+--------+--------+
|     100 | De Haan   | 11000 | 12400 |
|     100 | Errazuriz |  1400 |  1400 |
|     100 | Hartstein | 14000 | 26400 |
|     100 | Part      | 24000 | 50400 |
|     200 | Bell      | 23000 | 23000 |
|     200 | Part      | 24500 | 94800 |
|     200 | Red       | 23800 | 70300 |
|     200 | Ross      | 23500 | 46500 |
|     300 | Wei       | 23600 | 23600 |
+------------+-----------+--------+--------+
```

聚合函数示例

计算工资总额，执行以下语句：

```
SELECT SUM(salary) FROM employees;
```

查询结果如下：

```
+-------------+
| SUM(SALARY) |
+-------------+
|    168800 |
+-------------+
```

6.6.3.4. MAX

`MAX` 函数返回参数中指定的列中的最大值。

语法

```
MAX([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	返回的行中去除重复行，且忽略值为 NULL 的行。
UNIQUE	返回的行中去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 `OVER` 子句定义窗口进行计算。

返回类型

返回与 `expr` 相同的数据类型值。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT, last_name varchar(50), hiredate varchar(50), SALARY INT);
INSERT INTO employees VALUES(100, 'Wei',  '2019-09-11',17000);
INSERT INTO employees VALUES(100, 'Red', '2019-11-05', 17000);
INSERT INTO employees VALUES(101, 'Part',  '2018-10-01',12008);
INSERT INTO employees VALUES(102, 'Wei',  '2019-09-11',9000);
INSERT INTO employees VALUES(103, 'Red', '2019-11-05', 6000);
INSERT INTO employees VALUES(104, 'Part',  '2018-10-01',8000);
COMMIT;
```

执行以下语句查询 SALARY 列的最大值：

```
SELECT manager_id, last_name, salary FROM (SELECT manager_id, last_name, salary,
MAX(salary) OVER (PARTITION BY manager_id) AS rmax_sal
FROM employees) WHERE salary = rmax_sal ORDER BY manager_id, last_name, salary;
```

查询结果如下：

```
+------------+-----------+--------+
| MANAGER_ID | LAST_NAME | SALARY |
+------------+-----------+--------+
|      100 | Red     | 17000 |
|      100 | Wei     | 17000 |
|      101 | Part    | 12008 |
|      102 | Wei     |  9000 |
|      103 | Red     |  6000 |
|      104 | Part    |  8000 |
+------------+-----------+--------+
```

聚合函数示例

执行以下语句查询 SALARY 列的最大值：

```
SELECT MAX(salary) FROM employees;
```

查询结果如下：

```
+-------------+
| MAX(SALARY) |
+-------------+
|     17000 |
+-------------+
```

6.6.3.5. MIN

`MIN` 函数返回参数中指定列的最小值。

语法

```
MIN([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	返回的行中去除重复行，且忽略值为 NULL 的行。

UNIQUE	返回的行中去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 OVER 子句定义窗口进行计算。

返回类型

返回与 `expr` 相同数据类型的值。

示例

分析函数示例

以下语句创建了表 **employees**，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
```

执行以下语句查询 **SALARY** 列的最小值：

```
SELECT manager_id, last_name, hiredate, salary, MIN(salary) OVER(PARTITION BY manager_id
ORDER BY hiredate RANGE UNBOUNDED PRECEDING) AS p_cmin
FROM employees ORDER BY manager_id, last_name, hiredate, salary;
COMMIT;
```

查询结果如下：

```
+------------+-----------+------------+--------+--------+
| MANAGER_ID | LAST_NAME | HIREDATE   | SALARY | P_CMIN |
+------------+-----------+------------+--------+--------+
|     100 | De Haan   | 2018-05-01 |  11000 |   1400 |
|     100 | Errazuriz | 2017-07-21 |   1400 |   1400 |
|     100 | Hartstein | 2019-05-01 |  14000 |   1400 |
|     100 | Partners  | 2018-12-01 |  14000 |   1400 |
|     100 | Raphaely  | 2017-07-01 |   1700 |   1700 |
|     100 | Raphaely  | 2017-07-22 |   1700 |   1400 |
|     100 | Russell   | 2019-10-05 |  13000 |   1400 |
|     100 | Weiss     | 2019-07-11 |  13500 |   1400 |
|     200 | Bell      | 2019-05-25 |  13000 |  13000 |
|     200 | Part      | 2018-08-11 |  14000 |  14000 |
|     200 | Ross      | 2019-06-11 |  13500 |  13000 |
+------------+-----------+------------+--------+--------+
```

聚合函数示例

执行以下语句查询 SALARY 列的最小值：

```
SELECT MIN(salary) FROM employees;
```

查询结果如下：

```
+-------------+
| MIN(SALARY) |
+-------------+
|     1400 |
+-------------+
```

6.6.3.6. LISTAGG

`LISTAGG` 函数用于列转行，`LISTAGG` 对 `ORDER BY` 子句中指定的每个组内的数据进行排序，然后合并度量列的值。作为单个集合的聚合函数，`LISTAGG` 对所有行进行操作并返回单个输出行。作为组集聚合，`LISTAGG` 将对 `GROUP BY` 子句定义的每个组进行操作并返回输出行。作为分析函数，`LISTAGG` 基于 `query_partition_clause` 中的一个或多个表达式将查询结果集分为几组。

语法

```
LISTAGG（measure_expr [,'delimiter']）WITHIN GROUP (order_by_clause)
[OVER query_partition_clause]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
OVER	使用 `OVER` 子句定义窗口进行计算。
measure_expr	可以是任何表达式。度量列中的空值将被忽略。
delimiter	指定用于分隔度量值的字符串。此子句是可选的，默认为 NULL。

返回类型

如果度量列是 `RAW` 的，则返回数据类型为 `RAW`，否则返回值为 `VARCHAR2` 型。

示例

分析函数示例

建表 employees，并向里面插入数据，执行以下语句：

```
CREATE TABLE employees (department_id INT,manager_id INT,last_name varchar(50),hiredate varchar(50),
SALARY INT);
INSERT INTO employees VALUES(30, 100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(30, 100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(40, 100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(50, 100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(50, 100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(70, 100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(90, 100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(90,100, 'Partners',  '2018-12-01',14000);
```

查询 2019 年 10 月 10 日之前雇用的雇员，以及该雇员的部门，雇用日期以及该部门中的其他雇员。执行以下语句：

```
SELECT department_id "Dept", hiredate "Date", last_name "Name",LISTAGG(last_name, '; ') WITHIN GROUP
(ORDER BY hiredate, last_name) OVER (PARTITION BY department_id) as "Emp_list"
FROM employees WHERE hiredate < '2019-10-10' ORDER BY "Dept", "Date", "Name";
```

查询结果如下：

```
+------+------------+-----------+--------------------+
| Dept | Date       | Name      | Emp_list           |
+------+------------+-----------+--------------------+
|   30 | 2017-07-01 | Raphaely  | Raphaely; De Haan  |
|   30 | 2018-05-01 | De Haan   | Raphaely; De Haan  |
|   40 | 2017-07-21 | Errazuriz | Errazuriz          |
|   50 | 2017-07-22 | Raphaely  | Raphaely; Hartstein|
|   50 | 2019-05-01 | Hartstein | Raphaely; Hartstein|
|   70 | 2019-07-11 | Weiss     | Weiss              |
|   90 | 2018-12-01 | Partners  | Partners; Russell  |
|   90 | 2019-10-05 | Russell   | Partners; Russell  |
+------+------------+-----------+--------------------+
```

聚合函数示例

建表 employees，并向里面插入数据，执行以下语句：

```
CREATE TABLE employees (department_id INT,manager_id INT,last_name varchar(50),hiredate varchar(50),
SALARY INT);
INSERT INTO employees VALUES(30, 100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(30, 100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(30, 100, 'Errazuriz', '2017-07-01', 1400);
INSERT INTO employees VALUES(30, 100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(30, 100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(30, 100, 'Weiss',  '2019-07-01',13500);
INSERT INTO employees VALUES(30, 100, 'Russell', '2019-07-01', 13000);
INSERT INTO employees VALUES(30,100, 'Partners',  '2018-12-01',14000);
```

查询第 30 部门的所有员工，并按雇用日期和姓氏排序。执行以下语句：

```
SELECT LISTAGG(last_name, '; ') WITHIN GROUP (ORDER BY hiredate, last_name)  as "Emp_list",
MIN(hiredate) as "Earliest" FROM employees WHERE department_id = 30;
```

查询结果如下：

```
+-----------------------------------------------------------------------------+------------+
| Emp_list                                                                    | Earliest   |
+-----------------------------------------------------------------------------+------------+
| Errazuriz; Raphaely; Raphaely; De Haan; Partners; Hartstein; Russell; Weiss | 2017-07-01 |
+-----------------------------------------------------------------------------+------------+
```

6.6.3.7. ROLLUP

ROLLUP 函数是聚合函数，它是 GROUP BY 语句的简单扩展。在数据统计和报表生成过程中，它可以为每个分组返回一个小计，同时为所有分组返回总计，效率比 GROUP BY 和 UNION 组合方法高。

ROLLUP 函数的执行很简单，以下是它的执行顺序：

- 根据参数指定的列从右到左递减分组；
- 对每个分组小计，再对所有分组合计；
- 按照 ORDER BY col1 (，col2，col3，col4 ...) 排序。

如果 ROLLUP 参数为 N 个，则相当于 N+1 个 GROUP BY 分组的 UNION 的结合。

在涉及分组统计的任务中使用 ROLLUP 函数非常高效。例如，沿着时间或地理等层次维度进行小计，您只需要查询 ROLLUP(y, m, day) 或 ROLLUP(country, state, city) 。数据仓库管理员使用 ROLLUP 函数可以简化和加快汇总表的维护。

语法

```
SELECT … GROUP BY ROLLUP(col1 [,col2...])
```

参数

参数	说明
col1	进行分组操作的列名。列数指的是数据库中的行数。

示例

创建表 group_test，并向里面插入数据，执行以下语句：

```
CREATE TABLE group_test (group_id int, job varchar2(10), name varchar2(10), salary int);
INSERT INTO group_test VALUES (10, 'Coding', 'Bruce', 1000);
INSERT INTO group_test VALUES (10, 'Programmer', 'Clair', 1000);
INSERT INTO group_test VALUES (20, 'Coding', 'Jason', 2000);
INSERT INTO group_test VALUES (20, 'Programmer', 'Joey', 2000);
INSERT INTO group_test VALUES (30, 'Coding', 'Rebecca', 3000);
INSERT INTO group_test VALUES (30, 'Programmer', 'Rex', 3000);
INSERT INTO group_test VALUES (40, 'Coding', 'Samuel', 4000);
INSERT INTO group_test VALUES (40, 'Programmer', 'Susy', 4000);
COMMIT;
```

对 group_id 进行 GROUP BY 分组操作，执行以下语句：

```
SELECT group_id, SUM(salary) FROM group_test GROUP BY group_id;
```

查询结果如下：

```
+----------+-------------+
| GROUP_ID | SUM(SALARY) |
+----------+-------------+
|       10 |        2000 |
|       20 |        4000 |
|       30 |        6000 |
|       40 |        8000 |
+----------+-------------+
```

对 group_id 使用 `ROLLUP` 函数进行分组，同时求总计。执行以下语句：

```
SELECT group_id, SUM(salary) FROM group_test GROUP BY ROLLUP (group_id);
```

查询结果如下：

```
+----------+-------------+
| GROUP_ID | SUM(SALARY) |
+----------+-------------+
|       10 |        2000 |
|       20 |        4000 |
|       30 |        6000 |
|       40 |        8000 |
|     NULL |       20000 |
+----------+-------------+
```

对 group_id 列和 job 列使用 `ROLLUP` 函数进行分组，同时求总计。执行以下语句：

```
SELECT group_id, job, SUM(salary) FROM group_test GROUP BY ROLLUP (group_id, job);
```

查询结果如下：

```
+----------+------------+-------------+
| GROUP_ID | JOB        | SUM(SALARY) |
+----------+------------+-------------+
|       10 | Coding     |        1000 |
|       10 | Programmer |        1000 |
|       10 | NULL       |        2000 |
|       20 | Coding     |        2000 |
|       20 | Programmer |        2000 |
|       20 | NULL       |        4000 |
|       30 | Coding     |        3000 |
|       30 | Programmer |        3000 |
|       30 | NULL       |        6000 |
|       40 | Coding     |        4000 |
|       40 | Programmer |        4000 |
|       40 | NULL       |        8000 |
|     NULL | NULL       |       20000 |
+----------+------------+-------------+
```

上面的 SQL 语句用 `GROUP BY` 与 `UNION` 组合方法替换，执行以下语句：

```
SELECT group_id, job, SUM(salary) FROM group_test GROUP BY group_id, job
UNION ALL
SELECT group_id, NULL, SUM(salary) FROM group_test GROUP BY group_id
UNION ALL
SELECT NULL, NULL, SUM(salary) FROM group_test ORDER BY 1, 2;
```

查询结果如下：

```
+----------+------------+-------------+
| GROUP_ID | JOB        | SUM(SALARY) |
+----------+------------+-------------+
|       10 | Coding     |        1000 |
|       10 | Programmer |        1000 |
|       10 | NULL       |        2000 |
|       20 | Coding     |        2000 |
|       20 | Programmer |        2000 |
|       20 | NULL       |        4000 |
|       30 | Coding     |        3000 |
|       30 | Programmer |        3000 |
|       30 | NULL       |        6000 |
|       40 | Coding     |        4000 |
|       40 | Programmer |        4000 |
|       40 | NULL       |        8000 |
|     NULL | NULL       |       20000 |
+----------+------------+-------------+
```

输出结果和 `ROLLUP` 函数一样，但是 `ROLLUP` 函数更加简洁、高效。

6.6.3.8. STDDEV

`STDDEV` 函数用于计算总体标准差。`STDDEV` 函数将数值型数据作为参数，返回数值型数据。它与函数 `STDDEV_SAMP` 的不同之处在于，`STDDEV` 只有一行输入数据时返回 0，而 `STDDEV_SAMP` 返回 NULL。

OceanBase 中标准差的值是函数 `VARIANCE` 计算出的方差的算术平方根。

语法

```
STDDEV([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	去重关键字，表示计算唯一值的总体标准差。
UNIQUE	去重关键字，表示计算唯一值的总体标准差。

参数	说明
ALL	全部数值列。
expr	是数值类型或者可以转换成数值类型的类型。
OVER	使用 OVER 子句定义窗口进行计算。

> **注意**
>
> 如果您指定了 DISTINCT 或 UNIQUE 关键字，则 analytic_clause 中不允许出现 order_by_clause 和 windowing_clause 。

返回类型

返回 NUMBER 类型数据。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据

```
CREATE TABLE employees(manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

调用函数并执行以下语句：

```
SELECT last_name, salary, STDDEV(salary) OVER (ORDER BY hiredate) "StdDev"
FROM employees WHERE manager_id = 100 ORDER BY last_name, salary, "StdDev";
```

查询结果如下：

```
+-----------+--------+-----------------------------------------+
| LAST_NAME | SALARY | StdDev                                  |
+-----------+--------+-----------------------------------------+
| De Haan   | 11000 | 4702.127178203498995615489088200868644482 |
| Errazuriz |  1400 | 212.132034355964257320253308631454711785 |
| Hartstein |  14000 | 6340.346993658943269176828928801701088079 |
| Partners  |  14000 | 6064.899009876421676804205219406952308814 |
| Raphaely  |  1700 |                                       0 |
| Raphaely  |  1700 | 173.205080756887729352744634150587236694 |
| Russell   | 13000 | 6026.474330580265330900400184969999384459 |
| Weiss     | 13500 | 6244.311697171159907069428668980211861012 |
+-----------+--------+-----------------------------------------+
```

聚合函数示例

调用函数并执行以下语句：

```
SELECT STDDEV(salary) FROM employees WHERE manager_id = 100 ;
```

查询结果如下：

```
+-----------------------------------------+
| STDDEV(SALARY)                          |
+-----------------------------------------+
| 6026.474330580265330900400184969999384459 |
+-----------------------------------------+
```

6.6.3.9. STDDEV_POP

`STDDEV_POP` 函数计算总体标准差。`STDDEV_POP` 函数将数值型数据作为参数，返回数值型数据。

> **注意**
>
> 总体标准差是总体方差的算术平方根。

语法

```
STDDEV_POP([ALL] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
ALL	全部数值列。
OVER	使用 `OVER` 子句定义窗口进行计算。
expr	是数值类型（ `NUMBER` 、 `FLOAT` 、 `BINARY_FLOAT` 和 `BINARY_DOUBLE` ）的表达式。

返回类型

返回类型与参数 `expr` 的数据类型相同。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

调用函数并执行以下语句：

```
SELECT manager_id, last_name, salary, STDDEV_POP(salary) OVER (PARTITION BY manager_id) AS pop_std
FROM employees ORDER BY manager_id, last_name, salary, pop_std;
```

查询结果如下：

```
+------------+-----------+--------+-----------------------------------------+
| MANAGER_ID | LAST_NAME | SALARY | POP_STD                                 |
+------------+-----------+--------+-----------------------------------------+
|        100 | De Haan   |  11000 | 5637.250548804798333699350384281939588505 |
|        100 | Errazuriz |   1400 | 5637.250548804798333699350384281939588505 |
|        100 | Hartstein |  14000 | 5637.250548804798333699350384281939588505 |
|        100 | Partners  |  14000 | 5637.250548804798333699350384281939588505 |
|        100 | Raphaely  |   1700 | 5637.250548804798333699350384281939588505 |
|        100 | Raphaely  |   1700 | 5637.250548804798333699350384281939588505 |
|        100 | Russell   |  13000 | 5637.250548804798333699350384281939588505 |
|        100 | Weiss     |  13500 | 5637.250548804798333699350384281939588505 |
|        200 | Bell      |  13000 |  408.248290463863016366214012450981899069 |
|        200 | Part      |  14000 |  408.248290463863016366214012450981899069 |
|        200 | Ross      |  13500 |  408.248290463863016366214012450981899069 |
+------------+-----------+--------+-----------------------------------------+
```

聚合函数示例

调用函数并执行以下语句：

```
SELECT STDDEV_POP(salary) FROM employees ;
```

查询结果如下：

```
+-----------------------------------------+
| STDDEV_POP(SALARY)                      |
+-----------------------------------------+
| 5249.950806538512715446505486136315088416 |
+-----------------------------------------+
```

6.6.3.10. STDDEV_SAMP

`STDDEV_SAMP` 函数计算样本标准差。`STDDEV_SAMP` 函数将数值型数据作为参数，返回数值型数据。它与函数 `STDDEV` 的不同之处在于，`STDDEV` 只有一行输入数据时返回 0，而 `STDDEV_SAMP` 返回 NULL。

> **说明**
>
> 样本标准差是样本方差的算术平方根。

语法

```
STDDEV_SAMP([ALL] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 OVER 关键字。

参数

参数	说明
ALL	全部数值列。
expr	是数值类型（ NUMBER 、 FLOAT 、 BINARY_FLOAT 和 BINARY_DOUBLE ）或者可以转换成数值类型的表达式。
OVER	使用 OVER 子句定义窗口进行计算。

返回类型

返回类型与参数 expr 的数据类型相同。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

调用函数并执行以下语句：

```
SELECT manager_id, last_name, hiredate, salary,STDDEV_SAMP(salary) OVER (PARTITION BY manager_id
ORDER BY hiredate ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_sdev
FROM employees ORDER BY manager_id, last_name, hiredate, salary, cum_sdev;
```

返回结果如下：

```
+------------+-----------+------------+--------+-----------------------------------------+
| MANAGER_ID | LAST_NAME | HIREDATE   | SALARY | CUM_SDEV                                |
+------------+-----------+------------+--------+-----------------------------------------+
|        100 | De Haan   | 2018-05-01 |  11000 | 4702.127178203498995615489088200868644482 |
|        100 | Errazuriz | 2017-07-21 |   1400 | 212.132034355964257320253308631454711785 |
|        100 | Hartstein | 2019-05-01 |  14000 | 6340.346993658943269176828928801701088079 |
|        100 | Partners  | 2018-12-01 |  14000 | 6064.899009876421676804205219406952308814 |
|        100 | Raphaely  | 2017-07-01 |   1700 |                                    NULL |
|        100 | Raphaely  | 2017-07-22 |   1700 | 173.205080756887729352744634150587236694 |
|        100 | Russell   | 2019-10-05 |  13000 | 6026.474330580265330900400184969999384459 |
|        100 | Weiss     | 2019-07-11 |  13500 | 6244.311697171159907069428668980211861012 |
|        200 | Bell      | 2019-05-25 |  13000 | 707.106781186547524400844362104849039285 |
|        200 | Part      | 2018-08-11 |  14000 |                                    NULL |
|        200 | Ross      | 2019-06-11 |  13500 |                                     500 |
+------------+-----------+------------+--------+-----------------------------------------+
```

聚合函数示例

调用函数并执行以下语句：

```
SELECT STDDEV_SAMP(salary) FROM employees ;
```

查询结果如下：

```
+----------------------------------------+
| STDDEV_SAMP(SALARY)                    |
+----------------------------------------+
| 5506.194858355615640082358245403620332764 |
+----------------------------------------+
```

6.6.3.11. VARIANCE

`VARIANCE` 函数返回参数指定列的方差。

语法

```
VARIANCE([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	查询时去除列中的重复值，且忽略列中的 NULL 值。
UNIQUE	查询时去除列中的重复值，且忽略列中的 NULL 值。
ALL	查询时不去除列中的重复值，且忽略列中的 NULL 值。`ALL` 为默认值。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 `OVER` 子句定义窗口进行计算。

> **注意**
>
> 如果您指定了 `DISTINCT` 或 `UNIQUE` 关键字，则 `analytic_clause` 中不允许出现 `order_by_clause` 和 `windowing_clause`。

返回类型

返回 `NUMBER` 类型数据。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

执行以下语句计算列 salary 的方差：

```
SELECT last_name, salary, VARIANCE(salary)  OVER (ORDER BY hiredate) "Variance"
FROM employees WHERE manager_id = 100 ORDER BY last_name, salary, "Variance";
```

查询结果如下：

```
+-----------+--------+-----------------------------------------+
| LAST_NAME | SALARY | Variance                                |
+-----------+--------+-----------------------------------------+
| De Haan   | 11000 |                    22110000 |
| Errazuriz |  1400 |                     45000 |
| Hartstein |  14000 |                    40200000 |
| Partners  |  14000 |                    36783000 |
| Raphaely  |  1700 |                        0 |
| Raphaely  |  1700 |                     30000 |
| Russell   | 13000 | 36318392.857142857142857142857142857142 |
| Weiss     | 13500 | 38991428.571428571428571428571428571428 |
+-----------+--------+-----------------------------------------+
```

聚合函数示例

执行以下语句计算列 salary 的方差：

```
SELECT VARIANCE(salary)  FROM employees;
```

查询结果如下：

```
+----------------------------------------+
| VARIANCE(SALARY)                       |
+----------------------------------------+
| 30318181.818181818181818181818181818182 |
+----------------------------------------+
```

6.6.3.12. APPROX_COUNT_DISTINCT

APPROX_COUNT_DISTINCT 函数是聚合函数，它对某一列去重后的行数进行计算，结果只能返回一个值，且该值是近似值，该函数可以进一步用于计算被引用的列的选择性。

与函数 COUNT(DISTINCT x) 相比， APPROX_COUNT_DISTINCT 返回的是近似值，所以计算速度极快。在处理大量级的数据时 COUNT(DISTINCT x) 经常要花费很长的时间，使用 APPROX_COUNT_DISTINCT 牺牲了少量的精确度，却换来了计算效率的极大提升。

语法

```
APPROX_COUNT_DISTINCT(expr)
```

参数

参数	说明
expr	数值列。

返回类型

返回 NUMBER 类型数据。

示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

执行以下语句：

```
SELECT last_name, salary, APPROX_COUNT_DISTINCT(salary) OVER (ORDER BY hiredate) "Variance"
FROM employees WHERE manager_id = 100 ORDER BY last_name, salary, "Variance";
```

查询结果如下：

```
+-----------+--------+----------+
| LAST_NAME | SALARY | Variance |
+-----------+--------+----------+
| De Haan   | 11000 |      3 |
| Errazuriz |  1400 |      2 |
| Hartstein |  14000 |      4 |
| Partners  |  14000 |      4 |
| Raphaely  |  1700 |      1 |
| Raphaely  |  1700 |      2 |
| Russell   | 13000 |      6 |
| Weiss     | 13500 |      5 |
+-----------+--------+----------+
```

6.6.4. 分析函数

6.6.4.1. 窗口函数说明

分析函数（也叫窗口函数）与聚合函数，都是对行集组（一组行的集合）进行聚合计算，不同的是，聚合函数每组只能返回一个值（一行），而窗口函数每组可以返回多个值（多行）。行集组又称为窗口（Window），由 analytic_clause 定义。而窗口大小取决于实际的行数或逻辑间隔（例如时间）。组内每一行都是基于窗口的逻辑计算的结果。

触发一个分析函数需要特殊的关键字 OVER 来指定窗口。一个窗口包含三个组成部分：

- 分区规范，用于将输入行分裂到不同的分区中。这个过程和 GROUP BY 子句的分裂过程相似。
- 排序规范，用于决定输入数据行在窗口函数中执行的顺序。
- 窗口边界，指定计算数据的窗口边界。默认值为 RANGE UNBOUNDED PRECEDING 。这个边界包含当前分区中所有从开始到目前行所有数据。

分析函数是查询中执行的最后一组操作，除了最后的 ORDER BY 子句。在处理窗口函数之前，必须完成所有 JOIN 以及 WHERE 、 GROUP BY 和 HAVING 子句。因此，窗口函数只能出现在选择列表或 ORDER BY 子句中。

分析函数通常用于计算累积、移动、居中和报告汇总。

语法

analytic_function

```
analytic_function([ arguments ]) OVER (analytic_clause)
```

analytic_clause

```
[ query_partition_clause ] [ order_by_clause [ windowing_clause ] ]
```

query_partition_clause

```
PARTITION BY { expr[, expr ]... | ( expr[, expr ]... ) }
```

order_by_clause

```
ORDER [ SIBLINGS ] BY{ expr | position | c_alias } [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ] [, { expr | position | c_alias } [ ASC | DESC ][ NULLS FIRST | NULLS LAST ]]...
```

windowing_clause

```
{ ROWS | RANGE } { BETWEEN { UNBOUNDED PRECEDING | CURRENT ROW | value_expr { PRECEDING | FOLLOWING } } AND{ UNBOUNDED FOLLOWING | CURRENT ROW | value_expr { PRECEDING | FOLLOWING } } | { UNBOUNDED PRECEDING | CURRENT ROW| value_expr PRECEDING}}
```

在以下各节中将讨论此语法的语义。

analytic_function

分析函数（analytic_function）指定了分析函数的名称。

arguments

参数（arguments），分析函数采用 0~3 个参数。参数可以是任何数字数据类型或可以隐式转换为数字数据类型的任何非数字数据类型。OceanBase 根据数据类型优先级确定具有最高数值优先级的参数，然后将其余参数隐式转换为这个具有最高数值优先级参数的数据类型。返回类型也是具有最高数值优先级参数的数据类型，除非对单个函数另有说明。

analytic_clause

分析子句（analytic_clause），使用 `OVER analytic_clause` 指示函数在查询结果集上操作。此子句在 `FROM`、`WHERE`、`GROUP BY` 和 `HAVING` 子句之后计算。您可以在选择列表或按子句顺序中用此子句指定分析函数。若要筛选基于分析函数的查询结果，请将这些函数嵌套在父查询中，然后筛选嵌套子查询的结果。

> 注意
>
> - 您不能通过在分析子句中指定分析函数来嵌套分析函数。但是您可以在子查询中指定分析函数，并在其上计算另一个分析函数。
> - 您可以使用用户定义的解析函数以及内置的解析函数指定 `analytic_clause`。

query_partition_clause

分区子句（query_partition_clause），使用 `PARTITION BY` 子句将查询结果集划分为基于一个或多个 `value_expr` 的组。如果省略此子句，则函数将查询结果集的所有行视为单个组。

您可以在同一查询中指定多个分析函数，每个函数通过键具有相同或不同的分区。如果您用 `query_partition_clause` 指定了一个分析函数，并且被查询的对象具有并行属性，那么函数计算也是并行化的。

`value_expr` 的有效值是常量、列、非分析函数、函数表达式或涉及其中任何一个的表达式。

order_by_clause

使用排序子句 `order_by_clause` 指定如何在分区内对数据进行排序。 对于所有分析函数，您可以在多个键上的分区中对值进行排序，每个键由 `value_expr` 定义，由排序序列限定。

在每个函数中，您可以指定多个排序表达式。当使用对值进行排序的函数时，这样做特别有用。

当 `order_by_clause` 对多行产生相同的值时，函数的行为如下：

- `CUME_DIST`、`DENSE_RANK`、`NTILE`、`PERCENT_RANK` 和 `RANK` 为每一行返回相同的结果。
- 即使有一个基于 `order_by_clause` 的值，`ROW_NUMBER` 也会为每一行分配一个不同的值。该值基于行处理的顺序，如果 `ORDER BY` 不能实现总排序，则该顺序可能是不确定的。
- 对于其他分析函数，它的结果取决于窗口规则。如果您指定了一个带有 `RANGE` 关键字的逻辑窗口，则函数将为每个行返回相同的结果。如果您用 `ROWS` 关键字指定了物理窗口，则结果是不确定的。

ORDER BY 子句的限制

使用 `ORDER BY` 子句会受到以下限制：

- 在分析函数中，order_by_clause 必须使用表达式（expr）。SIBLINGS 关键字无效（仅与分层查询中相关）。位置（position）和列别名（c_alias）也无效。否则，该 order_by_clause 与整个查询或子查询的排序命令相同。
- 使用 RANGE 关键字的分析函数可以在其函数的 ORDER BY 子句中使用多个排序键。您需要指定以下窗口：
  - RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW，简称 RANGE UNBOUNDED PRECEDING。
  - RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
  - RANGE BETWEEN CURRENT ROW AND CURRENT ROW
  - RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

除了这四个之外，窗口边界只能在分析函数的 ORDER BY 子句中有一个排序键。此限制不适用于 ROW 关键字指定的窗口边界。

ASC 或 DESC 关键字

指定排序序列，ASC 指升序，DESC 指降序。默认为升序（ASC）。

NULLS FIRST 或 NULLS LAST 关键字

order_by_clause 中的 nulls first 和 nulls last，nulls first 表示排序时NULL值作为最小值处理，而 nulls last 表示排序时NULL值作为最大值处理。

windowing_clause

窗口函数子句（windowing_clause）有一些分析函数可以使用 windowing_clause。相关的关键字如下。

ROWS 或 RANGE 关键字

这些关键字为每一行定义一个用于计算函数结果的窗口，然后将该函数应用于窗口中的所有行。窗口从上到下通过查询结果集或分区移动。窗口也称为 FRAME，OceanBase 同时支持以下窗口语句：

- ROWS：以物理单位（行）指定窗口。
- RANGE：将窗口指定为逻辑偏移量。默认方式是 RANGE UNBOUNDED PRECEDING。您可以在分析函数中使用窗口函数，而要使用 windowing_clause，就必须添加 order_by_clause。

windowing_clause 若由 RANGE 子句定义窗口边界，那么在 order_by_clause 中您只能指定一个表达式。请参阅ORDER BY 子句的限制。具有逻辑偏移的分析函数返回的值总是确定性的。但是具有物理偏移量的分析函数返回的值可能会产生不确定的结果。排序表达式返回唯一的排序才能使具有物理偏移量的分析函数返回确定的值，因此，您必须在 order_by_clause 中指定多个列实现唯一的排序。

BETWEEN ... AND 关键字

使用 BETWEEN ... AND 子句指定窗口的起点和终点。第一个表达式（AND 之前）定义起点，第二个表达式（AND 之后）定义终点。如果省略 BETWEEN 并仅指定一个终点，则 OceanBase 将其视为起点，并且终点默认为当前行。

UNBOUNDED PRECEDING 关键字

UNBOUNDED PRECEDING 指示窗口从分区的第一行开始。这是起点，不是终点。

UNBOUNDED FOLLOWING 关键字

UNBOUNDED FOLLOWING 表示窗口在分区的最后一行结束。这是终点，不是起点。

CURRENT ROW 关键字

作为起点， CURRENT ROW 指定了窗口从当前行或当前值开始（取决于分别指定了 ROW 还是 RANGE ）。在这种情况下，端点不能为 value_expr PRECEDING 。作为终点， CURRENT ROW 指定窗口在当前行或值处结束（分别取决于您是否指定了 ROW 或 RANGE ）。在这种情况下，起点不能为 value_expr FOLLOWING 。

value_expr PRECEDING 或 value_expr FOLLOWING 关键字

- 如果 value_expr FOLLOWING 是起点，则终点必须是 value_expr FOLLOWING 。
- 如果 value_expr PRECEDING 是终点，则起点必须是 value_expr PRECEDING 。

如果要定义由时间间隔定义的数值格式的逻辑窗口，则可能需要使用转换函数。

如果您指定 ROWS ：

- value_expr 是物理偏移量。则它必须是一个常数或表达式，并且必须计算为正数。
- 如果 value_expr 是起点的一部分，则它必须把起点与终点之前的部分当作一行计算。

如果您指定 RANGE ：

- value_expr 是逻辑偏移量。它必须是一个常数或表达式，其结果为正数值或间隔字面量。
- 您只能在 order_by_clause 中指定一个表达式。
- 如果 value_expr 为数值，则 ORDER BY expr 必须为数值或 DATE 数据类型。
- 如果 value_expr 为间隔值，则 ORDER BY expr 必须为 DATE 数据类型。如果您完全省略 windowing_clause ，则默认值为 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW 。

6.6.4.2. AVG

AVG 函数返回数值列的平均值。

语法

```
AVG([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 OVER 关键字。

参数

参数	说明
DISTINCT	查询时去除数据中的重复值，且忽略数据中的 NULL 值。
UNIQUE	查询时去除数据中的重复值，且忽略数据中的 NULL 值。
ALL	查询时不去除数据中的重复值，且忽略数据中的 NULL 值。`ALL` 为默认值。
expr	是数值类型或者可以转换成数值类型的表达式。数值类型可以为：`NUMBER`、`FLOAT`、`BINARY_FLOAT` 或 `BINARY_DOUBLE`。
OVER	使用 `OVER` 子句定义窗口进行计算。

> **注意**
>
> 如果您指定了 `DISTINCT` 或 `UNIQUE` 关键字，则 `analytic_clause` 中不允许出现 `order_by_clause` 和 `windowing_clause`。

返回类型

返回类型与参数 `expr` 的数据类型相同。

示例

分析函数示例

以下语句创建了表 **employees**，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT, last_name varchar(50), hiredate varchar(50), SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

执行以下语句计算各列的平均值：

```
SELECT manager_id, last_name, hiredate, salary, AVG(salary) OVER (PARTITION BY manager_id
ORDER BY hiredate ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS c_mavg
FROM employees ORDER BY manager_id, hiredate, salary;
```

返回结果如下：

```
+------------+-----------+------------+--------+-----------------------------------------+
| MANAGER_ID | LAST_NAME | HIREDATE   | SALARY | C_MAVG                                  |
+------------+-----------+------------+--------+-----------------------------------------+
|        100 | Errazuriz | 2017-07-21 |   1400 |                                  1550 |
|        100 | Raphaely  | 2017-07-22 |   1700 |                                  4700 |
|        100 | De Haan   | 2018-05-01 |  11000 |                                  8900 |
|        100 | Partners  | 2018-12-01 |  14000 |                                 13000 |
|        100 | Hartstein | 2019-05-01 |  14000 |                              13833.333 |
|        100 | Weiss     | 2019-07-11 |  13500 |                                 13500 |
|        100 | Russell   | 2019-10-05 |  13000 |                                 13250 |
|        200 | Part      | 2018-08-11 |  14000 |                                 13500 |
|        200 | Bell      | 2019-05-25 |  13000 |                                 13500 |
|        200 | Ross      | 2019-06-11 |  13500 |                                 13250 |
+------------+-----------+------------+--------+-----------------------------------------+
```

聚合函数示例

执行以下语句计算 salary 的平均值：

```
SELECT AVG(salary) FROM employees;
```

查询结果如下：

```
+-----------------------------------------+
| AVG(SALARY)                             |
+-----------------------------------------+
| 10072.727272727272727272727272727272727273 |
+-----------------------------------------+
```

6.6.4.3. COUNT

`COUNT` 函数用于查询 `expr` 的行数。

语法

```
COUNT({ * | [ DISTINCT | UNIQUE | ALL ] expr }) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
*	表示满足条件的所有行，且包含值为 NULL 的行。
DISTINCT	返回的行中去除重复行，且忽略值为 NULL 的行。
UNIQUE	返回的行中去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	是数值类型或者可以转换成数值类型的表达式。数值类型可以为：`NUMBER` 、`FLOAT` 、`BINARY_FLOAT` 或 `BINARY_DOUBLE` 。
OVER	使用 OVER 子句定义窗口进行计算。

注意

- 对于 COUNT 函数，从不返回 NULL，如果指定了 expr，即返回 expr 不为 NULL 的统计个数，如果指定 COUNT(*) 返回所有行的统计数目。使用参数 DISTINCT 、UNIQUE 或 ALL 时需要与 expr 用空格隔开。
- 如果您指定了 DISTINCT 或 UNIQUE 关键字，则 analytic_clause 中不允许出现 order_by_clause 和 windowing_clause 。

返回类型

返回类型与参数 expr 的数据类型相同。

示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees(manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(300, 'Wei',  '2019-09-11',23600);
INSERT INTO employees VALUES(200, 'Red', '2019-11-05', 23800);
INSERT INTO employees VALUES(100, 'Part',  '2018-10-01',24000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',23500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 23000);
INSERT INTO employees VALUES(200, 'Part',  '2018-06-11',24500);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
COMMIT;
```

分析函数示例

执行以下语句查询表中的行数：

```
SELECT last_name, salary,COUNT(*) OVER (ORDER BY salary RANGE BETWEEN 50 PRECEDING
AND 150 FOLLOWING) AS mov_count FROM employees ORDER BY salary, last_name;
```

查询结果如下：

```
+-----------+--------+-----------+
| LAST_NAME | SALARY | MOV_COUNT |
+-----------+--------+-----------+
| Errazuriz |  1400 |      1 |
| De Haan   | 11000 |      1 |
| Hartstein | 14000 |      1 |
| Bell      | 23000 |      1 |
| Ross      | 23500 |      2 |
| Wei       | 23600 |      1 |
| Red       | 23800 |      1 |
| Part      | 24000 |      1 |
| Part      | 24500 |      1 |
+-----------+--------+-----------+
```

聚合函数示例

创建表 a，并向里面插入数据。执行以下语句：

```
CREATE TABLE a (
b INT
);
INSERT INTO a VALUES (1);
INSERT INTO a VALUES (null);
INSERT INTO a VALUES (null);
INSERT INTO a VALUES (1);
INSERT INTO a VALUES (null);
INSERT INTO a VALUES (1);
INSERT INTO a VALUES (1);
```

返回表 a 值不为 NULL 的统计个数，执行以下语句：

```
SELECT COUNT(b) FROM a;
```

查询结果如下：

```
+----------+
| COUNT(B) |
+----------+
|     4 |
+----------+
```

指定 `COUNT(*)` 返回所有行的统计数目，执行以下语句：

```
SELECT COUNT(*) FROM a;
```

查询结果如下：

```
+----------+
| COUNT(*) |
+----------+
|        7 |
+----------+
```

6.6.4.4. SUM

SUM 函数返回参数中指定列的和。此函数将可以隐式转换为数值数据类型的任何数值数据类型或任何非数值数据类型作为参数。 函数返回与参数的数值数据类型相同的数据类型。

语法

```
SUM([ DISTINCT | UNQIUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 OVER 关键字。

参数

参数	说明
DISTINCT	去除重复行，且忽略值为 NULL 的行。
UNQIUE	去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 OVER 子句定义窗口进行计算。

> 注意
>
> :
>
> 如果您指定了 DISTINCT 或 UNIQUE 关键字，则 analytic_clause 中不允许出现 order_by_clause 和 windowing_clause 。

返回类型

返回与 expr 相同数据类型的值。

示例

分析函数示例

创建表 employees，并向里面插入数据，执行以下语句：

```
CREATE TABLE employees(manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(300, 'Wei', '2019-09-11',23600);
INSERT INTO employees VALUES(200, 'Red', '2019-11-05', 23800);
INSERT INTO employees VALUES(100, 'Part', '2018-10-01',24000);
INSERT INTO employees VALUES(200, 'Ross', '2019-06-11',23500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 23000);
INSERT INTO employees VALUES(200, 'Part', '2018-06-11',24500);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
COMMIT;
```

计算工资总额，执行以下语句：

```
SELECT manager_id, last_name, salary, SUM(salary) OVER (PARTITION BY manager_id
ORDER BY salary RANGE UNBOUNDED PRECEDING) l_csum
FROM employees ORDER BY manager_id, last_name, salary, l_csum;
```

查询结果如下：

```
+------------+-----------+--------+--------+
| MANAGER_ID | LAST_NAME | SALARY | L_CSUM |
+------------+-----------+--------+--------+
|     100 | De Haan   | 11000 | 12400 |
|     100 | Errazuriz |  1400 |  1400 |
|     100 | Hartstein | 14000 | 26400 |
|     100 | Part      | 24000 | 50400 |
|     200 | Bell      | 23000 | 23000 |
|     200 | Part      | 24500 | 94800 |
|     200 | Red       | 23800 | 70300 |
|     200 | Ross      | 23500 | 46500 |
|     300 | Wei       | 23600 | 23600 |
+------------+-----------+--------+--------+
```

聚合函数示例

计算工资总额，执行以下语句：

```
SELECT SUM(salary) FROM employees;
```

查询结果如下：

```
+-------------+
| SUM(SALARY) |
+-------------+
|    168800 |
+-------------+
```

6.6.4.5. MAX

`MAX` 函数返回参数中指定的列中的最大值。

语法

```
MAX([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	返回的行中去除重复行，且忽略值为 NULL 的行。
UNIQUE	返回的行中去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 `OVER` 子句定义窗口进行计算。

返回类型

返回与 `expr` 相同的数据类型值。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT, last_name varchar(50), hiredate varchar(50), SALARY INT);
INSERT INTO employees VALUES(100, 'Wei',  '2019-09-11',17000);
INSERT INTO employees VALUES(100, 'Red', '2019-11-05', 17000);
INSERT INTO employees VALUES(101, 'Part',  '2018-10-01',12008);
INSERT INTO employees VALUES(102, 'Wei',  '2019-09-11',9000);
INSERT INTO employees VALUES(103, 'Red', '2019-11-05', 6000);
INSERT INTO employees VALUES(104, 'Part',  '2018-10-01',8000);
COMMIT;
```

执行以下语句查询 SALARY 列的最大值：

```
SELECT manager_id, last_name, salary FROM (SELECT manager_id, last_name, salary,
MAX(salary) OVER (PARTITION BY manager_id) AS rmax_sal
FROM employees) WHERE salary = rmax_sal ORDER BY manager_id, last_name, salary;
```

查询结果如下：

```
+------------+-----------+--------+
| MANAGER_ID | LAST_NAME | SALARY |
+------------+-----------+--------+
|      100 | Red     | 17000 |
|      100 | Wei     | 17000 |
|      101 | Part    | 12008 |
|      102 | Wei     |  9000 |
|      103 | Red     |  6000 |
|      104 | Part    |  8000 |
+------------+-----------+--------+
```

聚合函数示例

执行以下语句查询 SALARY 列的最大值：

```
SELECT MAX(salary) FROM employees;
```

查询结果如下：

```
+-------------+
| MAX(SALARY) |
+-------------+
|     17000 |
+-------------+
```

6.6.4.6. MIN

`MIN` 函数返回参数中指定列的最小值。

语法

```
MIN([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	返回的行中去除重复行，且忽略值为 NULL 的行。

UNIQUE	返回的行中去除重复行，且忽略值为 NULL 的行。
ALL	返回所有值，包含重复行，且忽略值为 NULL 的行。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 OVER 子句定义窗口进行计算。

返回类型

返回与 `expr` 相同数据类型的值。

示例

分析函数示例

以下语句创建了表 **employees**，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
```

执行以下语句查询 **SALARY** 列的最小值：

```
SELECT manager_id, last_name, hiredate, salary, MIN(salary) OVER(PARTITION BY manager_id
ORDER BY hiredate RANGE UNBOUNDED PRECEDING) AS p_cmin
FROM employees ORDER BY manager_id, last_name, hiredate, salary;
COMMIT;
```

查询结果如下：

```
+------------+-----------+------------+--------+--------+
| MANAGER_ID | LAST_NAME | HIREDATE   | SALARY | P_CMIN |
+------------+-----------+------------+--------+--------+
|     100 | De Haan   | 2018-05-01 |  11000 |   1400 |
|     100 | Errazuriz | 2017-07-21 |   1400 |   1400 |
|     100 | Hartstein | 2019-05-01 |  14000 |   1400 |
|     100 | Partners  | 2018-12-01 |  14000 |   1400 |
|     100 | Raphaely  | 2017-07-01 |   1700 |   1700 |
|     100 | Raphaely  | 2017-07-22 |   1700 |   1400 |
|     100 | Russell   | 2019-10-05 |  13000 |   1400 |
|     100 | Weiss     | 2019-07-11 |  13500 |   1400 |
|     200 | Bell      | 2019-05-25 |  13000 |  13000 |
|     200 | Part      | 2018-08-11 |  14000 |  14000 |
|     200 | Ross      | 2019-06-11 |  13500 |  13000 |
+------------+-----------+------------+--------+--------+
```

聚合函数示例

执行以下语句查询 SALARY 列的最小值：

```
SELECT MIN(salary) FROM employees;
```

查询结果如下：

```
+-------------+
| MIN(SALARY) |
+-------------+
|     1400 |
+-------------+
```

6.6.4.7. LISTAGG

`LISTAGG` 函数用于列转行，`LISTAGG` 对 `ORDER BY` 子句中指定的每个组内的数据进行排序，然后合并度量列的值。作为单个集合的聚合函数，`LISTAGG` 对所有行进行操作并返回单个输出行。作为组集聚合，`LISTAGG` 将对 `GROUP BY` 子句定义的每个组进行操作并返回输出行。作为分析函数，`LISTAGG` 基于 `query_partition_clause` 中的一个或多个表达式将查询结果集分为几组。

语法

```
LISTAGG（measure_expr [,'delimiter']）WITHIN GROUP (order_by_clause)
[OVER query_partition_clause]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
OVER	使用 `OVER` 子句定义窗口进行计算。
measure_expr	可以是任何表达式。度量列中的空值将被忽略。
delimiter	指定用于分隔度量值的字符串。此子句是可选的，默认为 NULL。

返回类型

如果度量列是 `RAW` 的，则返回数据类型为 `RAW`，否则返回值为 `VARCHAR2` 型。

示例

分析函数示例

建表 employees，并向里面插入数据，执行以下语句：

```
CREATE TABLE employees (department_id INT,manager_id INT,last_name varchar(50),hiredate varchar(50),
SALARY INT);
INSERT INTO employees VALUES(30, 100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(30, 100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(40, 100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(50, 100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(50, 100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(70, 100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(90, 100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(90,100, 'Partners',  '2018-12-01',14000);
```

查询 2019 年 10 月 10 日之前雇用的雇员，以及该雇员的部门，雇用日期以及该部门中的其他雇员。执行以下语句：

```
SELECT department_id "Dept", hiredate "Date", last_name "Name",LISTAGG(last_name, '; ') WITHIN GROUP
(ORDER BY hiredate, last_name) OVER (PARTITION BY department_id) as "Emp_list"
FROM employees WHERE hiredate < '2019-10-10' ORDER BY "Dept", "Date", "Name";
```

查询结果如下：

```
+------+------------+-----------+--------------------+
| Dept | Date       | Name      | Emp_list           |
+------+------------+-----------+--------------------+
|   30 | 2017-07-01 | Raphaely  | Raphaely; De Haan  |
|   30 | 2018-05-01 | De Haan   | Raphaely; De Haan  |
|   40 | 2017-07-21 | Errazuriz | Errazuriz          |
|   50 | 2017-07-22 | Raphaely  | Raphaely; Hartstein|
|   50 | 2019-05-01 | Hartstein | Raphaely; Hartstein|
|   70 | 2019-07-11 | Weiss     | Weiss              |
|   90 | 2018-12-01 | Partners  | Partners; Russell  |
|   90 | 2019-10-05 | Russell   | Partners; Russell  |
+------+------------+-----------+--------------------+
```

聚合函数示例

建表 employees，并向里面插入数据，执行以下语句：

```
CREATE TABLE employees (department_id INT,manager_id INT,last_name varchar(50),hiredate varchar(50),
SALARY INT);
INSERT INTO employees VALUES(30, 100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(30, 100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(30, 100, 'Errazuriz', '2017-07-01', 1400);
INSERT INTO employees VALUES(30, 100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(30, 100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(30, 100, 'Weiss',  '2019-07-01',13500);
INSERT INTO employees VALUES(30, 100, 'Russell', '2019-07-01', 13000);
INSERT INTO employees VALUES(30,100, 'Partners',  '2018-12-01',14000);
```

查询第 30 部门的所有员工，并按雇用日期和姓氏排序。执行以下语句：

```
SELECT LISTAGG(last_name, '; ') WITHIN GROUP (ORDER BY hiredate, last_name)  as "Emp_list",
MIN(hiredate) as "Earliest" FROM employees WHERE department_id = 30;
```

查询结果如下：

```
+-----------------------------------------------------------------------------+------------+
| Emp_list                                                                    | Earliest   |
+-----------------------------------------------------------------------------+------------+
| Errazuriz; Raphaely; Raphaely; De Haan; Partners; Hartstein; Russell; Weiss | 2017-07-01 |
+-----------------------------------------------------------------------------+------------+
```

6.6.4.8. STDDEV

`STDDEV` 函数用于计算总体标准差。`STDDEV` 函数将数值型数据作为参数，返回数值型数据。它与函数 `STDDEV_SAMP` 的不同之处在于，`STDDEV` 只有一行输入数据时返回 0，而 `STDDEV_SAMP` 返回 NULL。

OceanBase 中标准差的值是函数 `VARIANCE` 计算出的方差的算术平方根。

语法

```
STDDEV([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	去重关键字，表示计算唯一值的总体标准差。
UNIQUE	去重关键字，表示计算唯一值的总体标准差。
ALL	全部数值列。
expr	是数值类型或者可以转换成数值类型的类型。
OVER	使用 `OVER` 子句定义窗口进行计算。

> **注意**
>
> 如果您指定了 `DISTINCT` 或 `UNIQUE` 关键字，则 `analytic_clause` 中不允许出现 `order_by_clause` 和 `windowing_clause` 。

返回类型

返回 `NUMBER` 类型数据。

示例

分析函数示例

以下语句创建了表 **employees**，并向里面插入数据

```
CREATE TABLE employees(manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

调用函数并执行以下语句：

```
SELECT last_name, salary, STDDEV(salary) OVER (ORDER BY hiredate) "StdDev"
FROM employees WHERE manager_id = 100 ORDER BY last_name, salary, "StdDev";
```

查询结果如下：

```
+-----------+--------+-----------------------------------------+
| LAST_NAME | SALARY | StdDev                                  |
+-----------+--------+-----------------------------------------+
| De Haan   | 11000 | 4702.12717820349899561548908820086864448 2 |
| Errazuriz |  1400 |  212.13203435596425732025330863145471178 5 |
| Hartstein |  14000 | 6340.34699365894326917682892880170108807 9 |
| Partners  |  14000 | 6064.89900987642167680420521940695230881 4 |
| Raphaely  |  1700 |                                       0 |
| Raphaely  |  1700 |  173.20508075688772935274463415058723669 4 |
| Russell   | 13000 | 6026.47433058026533090040018496999938445 9 |
| Weiss     | 13500 | 6244.31169717115990706942866898021186101 2 |
+-----------+--------+-----------------------------------------+
```

聚合函数示例

调用函数并执行以下语句：

```
SELECT STDDEV(salary) FROM employees WHERE manager_id = 100 ;
```

查询结果如下：

```
+-----------------------------------------+
| STDDEV(SALARY)                          |
+-----------------------------------------+
| 6026.474330580265330900400184969999384459 |
+-----------------------------------------+
```

6.6.4.9. STDDEV_POP

`STDDEV_POP` 函数计算总体标准差。`STDDEV_POP` 函数将数值型数据作为参数，返回数值型数据。

> **注意**
>
> 总体标准差是总体方差的算术平方根。

语法

```
STDDEV_POP([ALL] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
ALL	全部数值列。
OVER	使用 `OVER` 子句定义窗口进行计算。
expr	是数值类型（`NUMBER`、`FLOAT`、`BINARY_FLOAT` 和 `BINARY_DOUBLE`）的表达式。

返回类型

返回类型与参数 `expr` 的数据类型相同。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

调用函数并执行以下语句：

```
SELECT manager_id, last_name, salary, STDDEV_POP(salary) OVER (PARTITION BY manager_id) AS pop_std
FROM employees ORDER BY manager_id, last_name, salary, pop_std;
```

查询结果如下：

```
+------------+-----------+--------+-----------------------------------------+
| MANAGER_ID | LAST_NAME | SALARY | POP_STD                                 |
+------------+-----------+--------+-----------------------------------------+
|        100 | De Haan   |  11000 | 5637.250548804798333699350384281939588505 |
|        100 | Errazuriz |   1400 | 5637.250548804798333699350384281939588505 |
|        100 | Hartstein |  14000 | 5637.250548804798333699350384281939588505 |
|        100 | Partners  |  14000 | 5637.250548804798333699350384281939588505 |
|        100 | Raphaely  |   1700 | 5637.250548804798333699350384281939588505 |
|        100 | Raphaely  |   1700 | 5637.250548804798333699350384281939588505 |
|        100 | Russell   |  13000 | 5637.250548804798333699350384281939588505 |
|        100 | Weiss     |  13500 | 5637.250548804798333699350384281939588505 |
|        200 | Bell      |  13000 |  408.248290463863016366214012450981899069 |
|        200 | Part      |  14000 |  408.248290463863016366214012450981899069 |
|        200 | Ross      |  13500 |  408.248290463863016366214012450981899069 |
+------------+-----------+--------+-----------------------------------------+
```

聚合函数示例

调用函数并执行以下语句：

```
SELECT STDDEV_POP(salary) FROM employees ;
```

查询结果如下：

```
+-----------------------------------------+
| STDDEV_POP(SALARY)                      |
+-----------------------------------------+
| 5249.95080653851271544650548613631508841 6 |
+-----------------------------------------+
```

6.6.4.10. STDDEV_SAMP

STDDEV_SAMP 函数计算样本标准差。STDDEV_SAMP 函数将数值型数据作为参数，返回数值型数据。它与函数 STDDEV 的不同之处在于，STDDEV 只有一行输入数据时返回 0，而 STDDEV_SAMP 返回 NULL。

> 说明
> 样本标准差是样本方差的算术平方根。

语法

```
STDDEV_SAMP([ALL] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值 。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 OVER 关键字。

参数

参数	说明
ALL	全部数值列。
expr	是数值类型（NUMBER、FLOAT、BINARY_FLOAT 和 BINARY_DOUBLE）或者可以转换成数值类型的表达式。
OVER	使用 OVER 子句定义窗口进行计算。

返回类型

返回类型与参数 expr 的数据类型相同。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

调用函数并执行以下语句：

```
SELECT manager_id, last_name, hiredate, salary,STDDEV_SAMP(salary) OVER (PARTITION BY manager_id
ORDER BY hiredate ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_sdev
FROM employees ORDER BY manager_id, last_name, hiredate, salary, cum_sdev;
```

返回结果如下：

```
+------------+-----------+------------+--------+-----------------------------------------+
| MANAGER_ID | LAST_NAME | HIREDATE   | SALARY | CUM_SDEV                                |
+------------+-----------+------------+--------+-----------------------------------------+
|        100 | De Haan   | 2018-05-01 |  11000 | 4702.127178203498995615489088200868644482 |
|        100 | Errazuriz | 2017-07-21 |   1400 |  212.13203435596425732025330863145471178 5 |
|        100 | Hartstein | 2019-05-01 |  14000 | 6340.346993658943269176828928801701088079 |
|        100 | Partners  | 2018-12-01 |  14000 | 6064.899009876421676804205219406952308814 |
|        100 | Raphaely  | 2017-07-01 |   1700 |                                    NULL |
|        100 | Raphaely  | 2017-07-22 |   1700 | 173.20508075688772935274463415058723669 4 |
|        100 | Russell   | 2019-10-05 |  13000 | 6026.474330580265330900400184969999384459 |
|        100 | Weiss     | 2019-07-11 |  13500 | 6244.311697171159907069428668980211861012 |
|        200 | Bell      | 2019-05-25 |  13000 |  707.106781186547524400844362104849039285 |
|        200 | Part      | 2018-08-11 |  14000 |                                    NULL |
|        200 | Ross      | 2019-06-11 |  13500 |                                     500 |
+------------+-----------+------------+--------+-----------------------------------------+
```

聚合函数示例

调用函数并执行以下语句：

```
SELECT STDDEV_SAMP(salary) FROM employees ;
```

查询结果如下：

```
+-----------------------------------------+
| STDDEV_SAMP(SALARY)                     |
+-----------------------------------------+
| 5506.19485835561564008235824540362033276 4 |
+-----------------------------------------+
```

6.6.4.11. VARIANCE

`VARIANCE` 函数返回参数指定列的方差。

语法

```
VARIANCE([ DISTINCT | UNIQUE | ALL ] expr) [ OVER (analytic_clause) ]
```

作为分析函数使用时，您需要使用窗口函数的完整语法，它对一组行的集合进行计算并返回多个值。作为聚合函数使用时，该函数对一组行的集合进行聚合计算，结果只能返回一个值，此时不需要加 `OVER` 关键字。

参数

参数	说明
DISTINCT	查询时去除列中的重复值，且忽略列中的 NULL 值。
UNIQUE	查询时去除列中的重复值，且忽略列中的 NULL 值。
ALL	查询时不去除列中的重复值，且忽略列中的 NULL 值。`ALL` 为默认值。
expr	可为数值、字符、日期型或其它类型的数据列或表达式。
OVER	使用 `OVER` 子句定义窗口进行计算。

> 注意
>
> 如果您指定了 DISTINCT 或 UNIQUE 关键字，则 analytic_clause 中不允许出现 order_by_clause 和 windowing_clause 。

返回类型

返回 NUMBER 类型数据。

示例

分析函数示例

以下语句创建了表 employees，并向里面插入数据：

```
CREATE TABLE employees (manager_id INT,last_name varchar(50),hiredate varchar(50),SALARY INT);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-01', 1700);
INSERT INTO employees VALUES(100, 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Errazuriz', '2017-07-21', 1400);
INSERT INTO employees VALUES(100, 'Hartstein', '2019-05-01',14000);
INSERT INTO employees VALUES(100, 'Raphaely', '2017-07-22', 1700);
INSERT INTO employees VALUES(100, 'Weiss',  '2019-07-11',13500);
INSERT INTO employees VALUES(100, 'Russell', '2019-10-05', 13000);
INSERT INTO employees VALUES(100, 'Partners',  '2018-12-01',14000);
INSERT INTO employees VALUES(200, 'Ross',  '2019-06-11',13500);
INSERT INTO employees VALUES(200, 'Bell', '2019-05-25', 13000);
INSERT INTO employees VALUES(200, 'Part',  '2018-08-11',14000);
COMMIT;
```

执行以下语句计算列 salary 的方差：

```
SELECT last_name, salary, VARIANCE(salary)  OVER (ORDER BY hiredate) "Variance"
FROM employees WHERE manager_id = 100 ORDER BY last_name, salary, "Variance";
```

查询结果如下：

```
+-----------+--------+-----------------------------------------+
| LAST_NAME | SALARY | Variance                                |
+-----------+--------+-----------------------------------------+
| De Haan   | 11000 |                                22110000 |
| Errazuriz |  1400 |                                   45000 |
| Hartstein |  14000 |                                40200000 |
| Partners  |  14000 |                                36783000 |
| Raphaely  |  1700 |                                       0 |
| Raphaely  |  1700 |                                   30000 |
| Russell   | 13000 | 36318392.857142857142857142857142857142 |
| Weiss     | 13500 | 38991428.571428571428571428571428571857 |
+-----------+--------+-----------------------------------------+
```

聚合函数示例

执行以下语句计算列 salary 的方差：

```
SELECT VARIANCE(salary)  FROM employees;
```

查询结果如下：

```
+---------------------------------------+
| VARIANCE(SALARY)                      |
+---------------------------------------+
| 30318181.818181818181818181818181818182 |
+---------------------------------------+
```

6.6.4.12. RANK

RANK 函数基于 OVER 子句中的 ORDER BY 表达式确定一组值的排名。当有相同排序值时，将会有相同的排名，并且值相同的行数会被记录到下个排名中。

语法

```
RANK()  OVER ( [ PARTITION BY expr_list ] [ ORDER BY order_list ])
```

参数

参数	说明
OVER	使用 OVER 子句定义窗口进行计算。

参数	说明
PARTITION BY [col1, col2..]	指定开窗口的列。
ORDER BY col1[asc|desc]	指定排名依据的值。
expr_list	是数值类型或者可以转换成数值类型的类型。
order_list	定义排名值参考的数据列。

示例

建表 course，并向 name 和 grade 列插入数据，执行以下语句：

```
CREATE TABLE course
(
 name VARCHAR(8),
 grade NUMBER
);
INSERT INTO course VALUES('Linda',50);
INSERT INTO course VALUES('Tan',85);
INSERT INTO course VALUES('Tom',90);
INSERT INTO course VALUES('John',95);
INSERT INTO course VALUES('Mery',55);
INSERT INTO course VALUES('Peter',60);
INSERT INTO course VALUES('Jack',65);
INSERT INTO course VALUES('Rose',70);
INSERT INTO course VALUES('Tonny',75);
INSERT INTO course VALUES('Apple',80);
COMMIT;
```

执行以下语句：

```
SELECT name,grade ,RANK() over(ORDER BY grade DESC) FROM course;
```

查询结果如下：

```
+-------+-------+-----------------------------+
| NAME  | GRADE | RANK()OVER(ORDERBYGRADEDESC) |
+-------+-------+-----------------------------+
| John  |  95   |              1              |
| Tom   |  90   |              2              |
| Tan   |  85   |              3              |
| Apple |  80   |              4              |
| Tonny |  75   |              5              |
| Rose  |  70   |              6              |
| Jack  |  65   |              7              |
| Peter |  60   |              8              |
| Mery  |  55   |              9              |
| Linda |  50   |             10              |
+-------+-------+-----------------------------+
```

6.6.4.13. LEAD

`LEAD` 是一种分析函数，它提供了对表多行的访问，而无需进行自我连接。给定从查询返回的一些列行和光标的位置，`LEAD` 提供超出该位置的物理偏移量的行的访问。

语法

```
LEAD { (value_expr [,offset [,default]]) [RESPECT|IGNORE] NULLS
|(value_expr [RESPECT|IGNORE] NULLS [,offset [,default]]) }
OVER([query_partition_clause] order_by_clause)
```

参数

参数	说明
OVER	使用 `OVER` 子句定义窗口进行计算。
offset	是 value_expr 的可选参数偏移量。
default	如果未指定默认值，则其默认值为 `null` 。如果在 `LEAD` 没有显式的设置 default 值的情况下，返回值为 `NULL` 。

参数	说明
{RESPECT | IGNORE} NULLS	表示是否忽略 NULL 值。默认值为 RESPECT NULLS，考虑 NULL 值。
value_expr	是要做比对的字段。您不能使用 LEAD 函数或其他分析函数来嵌套 value_expr 。

> 注意
>
> LEAD 函数后必须跟 order_by_clause， query_partition_clause 是可选的。

返回类型

返回的数据类型不限。

示例

创建 emp_msg 表，并向列中插入数据。执行以下语句：

```
CREATE TABLE emp_msg(deptno INT, ename VARCHAR(30),sal INT);
INSERT INTO emp_msg VALUES(20,'ADAMS',1400);
INSERT INTO emp_msg VALUES(30,'ALLEN',1900);
INSERT INTO emp_msg VALUES(30,'BLAKE',3135);
INSERT INTO emp_msg VALUES(10,'CLARK',2750);
INSERT INTO emp_msg VALUES(20,'FORD',3300);
INSERT INTO emp_msg VALUES(30,'JAMES',1250);
INSERT INTO emp_msg VALUES(20,'JONES',3275);
INSERT INTO emp_msg VALUES(10,'KING',5300);
INSERT INTO emp_msg VALUES(30,'MARTIN',1550);
INSERT INTO emp_msg VALUES(10,'MILLER',1600);
INSERT INTO emp_msg VALUES(20,'SCOTT',3300);
INSERT INTO emp_msg VALUES(20,'SWITH',1100);
INSERT INTO emp_msg VALUES(30,'TURNER',1800);
INSERT INTO emp_msg VALUES(30,'WARD',1550);
```

查询 emp_msg 表，最后 5 个值用 Jane 代替，从倒数第 6 个值开始追加按 ename 列升序排列的值。

```
SELECT  deptno, ename, sal, LEAD(ename,5,'Jane') OVER (ORDER BY ename) AS new_ename
FROM emp_msg;
```

查询结果如下：

```
+--------+--------+------+-----------+
| DEPTNO | ENAME  | SAL  | NEW_ENAME |
+--------+--------+------+-----------+
|   20 | ADAMS  | 1400 | JAMES     |
|   30 | ALLEN  | 1900 | JONES     |
|   30 | BLAKE  | 3135 | KING      |
|   10 | CLARK  | 2750 | MARTIN    |
|   20 | FORD   | 3300 | MILLER    |
|   30 | JAMES  | 1250 | SCOTT     |
|   20 | JONES  | 3275 | SWITH     |
|   10 | KING   | 5300 | TURNER    |
|   30 | MARTIN | 1550 | WARD      |
|   10 | MILLER | 1600 | Jane      |
|   20 | SCOTT  | 3300 | Jane      |
|   20 | SWITH  | 1100 | Jane      |
|   30 | TURNER | 1800 | Jane      |
|   30 | WARD   | 1550 | Jane      |
+--------+--------+------+-----------+
```

6.6.4.14. LAG

`LAG` 是一个分析函数。它同时提供对多行表的访问，而不需要自连接。给定从查询返回的一系列行和游标的位置，`LAG` 可以访问位于该位置之前给定物理偏移量的行。您可以给偏移参数指定一个大于零的整数。如果不指定偏移量，则其默认值为 1。如果偏移量超出窗口的范围，则返回可选值。如果不指定默认值，则其默认值为 `NULL`。

语法

```
LAG { (value_expr [,offset [,default]]) [RESPECT|IGNORE] NULLS
|(value_expr [RESPECT | IGNORE] NULLS [,offset [,default] ]) }
OVER([query_partition_clause] order_by_clause)
```

参数

参数	说明
value_expr	是要做比对的字段。您不能使用 `LAG` 函数或其他分析函数来嵌套 `value_expr`。

参数	说明
offset	是 value_expr 的可选参数偏移量。
default	如果未指定默认值，则其默认值为 `NULL` 。如果在 `LEAD` 没有显式的设置 default 值的情况下，返回值为 `NULL` 。
{RESPECT | IGNORE} NULLS	表示是否忽略 `NULL` 值。默认值为 `RESPECT NULLS` ，考虑 `NULL` 值。
OVER	使用 OVER 子句定义窗口进行计算。

> **注意**
>
> `LAG` 函数后必须跟 order_by_clause ， query_partition_clause 是可选的。

返回类型

返回值的数据类型不限。

示例

创建 emp_msg 表，并向里面插入数据。执行以下语句：

```
CREATE TABLE emp_msg(deptno INT, ename varchar(30),sal INT);
INSERT INTO emp_msg VALUES(20,'ADAMS',1400);
INSERT INTO emp_msg VALUES(30,'ALLEN',1900);
INSERT INTO emp_msg VALUES(30,'BLAKE',3135);
INSERT INTO emp_msg VALUES(10,'CLARK',2750);
INSERT INTO emp_msg VALUES(20,'FORD',3300);
INSERT INTO emp_msg VALUES(30,'JAMES',1250);
INSERT INTO emp_msg VALUES(20,'JONES',3275);
INSERT INTO emp_msg VALUES(10,'KING',5300);
INSERT INTO emp_msg VALUES(30,'MARTIN',1550);
INSERT INTO emp_msg VALUES(10,'MILLER',1600);
INSERT INTO emp_msg VALUES(20,'SCOTT',3300);
INSERT INTO emp_msg VALUES(20,'SWITH',1100);
INSERT INTO emp_msg VALUES(30,'TURNER',1800);
INSERT INTO emp_msg VALUES(30,'WARD',1550);
```

查询 emp_msg 表，最后 5 个值用 Jane 代替，从倒数第 6 个值开始追加按 ename 字段升序排列的值。执行以下语句：

```
SELECT deptno, ename, sal, LAG(ename,5,'Jane') OVER (ORDER BY ename) AS new_ename
FROM emp_msg;
```

查询结果如下：

```
+--------+--------+------+-----------+
| DEPTNO | ENAME  | SAL  | NEW_ENAME |
+--------+--------+------+-----------+
|   20 | ADAMS  | 1400 | Jane      |
|   30 | ALLEN  | 1900 | Jane      |
|   30 | BLAKE  | 3135 | Jane      |
|   10 | CLARK  | 2750 | Jane      |
|   20 | FORD   | 3300 | Jane      |
|   30 | JAMES  | 1250 | ADAMS     |
|   20 | JONES  | 3275 | ALLEN     |
|   10 | KING   | 5300 | BLAKE     |
|   30 | MARTIN | 1550 | CLARK     |
|   10 | MILLER | 1600 | FORD      |
|   20 | SCOTT  | 3300 | JAMES     |
|   20 | SWITH  | 1100 | JONES     |
|   30 | TURNER | 1800 | KING      |
|   30 | WARD   | 1550 | MARTIN    |
+--------+--------+------+-----------+
```

6.6.4.15. FIRST_VALUE

`FIRST_VALUE` 是一个分析函数。它返回有序值中的第一个值。如果集合中的第一个值为 `NULL`，则函数返回 `NULL`，除非您指定 `IGNORE NULLS`，该设置对于数据致密化很有用。

语法

```
FIRST_VALUE { (expr) [ {RESPECT | IGNORE} NULLS ] | (expr [ {RESPECT | IGNORE} NULLS ]) } OVER (analytic_clause)
```

参数

参数	说明
expr	参数类型不限。
OVER	使用 `OVER` 子句定义窗口进行计算。
{RESPECT | IGNORE} NULLS	表示是否忽略 `NULL` 值。默认值为 `RESPECT NULLS`，考虑 `NULL` 值。
FROM { FIRST | LAST }	确定计算方向是从窗口的第一行还是最后一行开始，默认值为 `FROM FIRST`。如果您指定 `IGNORE NULLS`，则 `FIRST_VALUE` 返回集合中的第一个非空值，如果所有值都为 `NULL`，则返回 `NULL`。

返回类型

数据类型不限。

示例

建表 emp_msg，并向里面插入数据。执行以下语句：

```
CREATE TABLE emp_msg(deptno INT, ename VARCHAR(30), sal INT, MGR VARCHAR(30));
INSERT INTO emp_msg VALUES(10,'CLARK', 2750, 7839);
INSERT INTO emp_msg VALUES(10,'KING', 5300, NULL);
INSERT INTO emp_msg VALUES(10,'MILLER', 1600, 7782);
INSERT INTO emp_msg VALUES(20,'ADAMS', 1400, 7788);
INSERT INTO emp_msg VALUES(20,'FORD', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'JONES', 3275, 7839);
INSERT INTO emp_msg VALUES(20,'SCOTT', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'SMITH', 1100, 7902);
INSERT INTO emp_msg VALUES(30,'ALLEN', 1900, 7698);
INSERT INTO emp_msg VALUES(30,'BLAKE', 3150, 7839);
INSERT INTO emp_msg VALUES(30,'JAMES', 1250, 7698);
INSERT INTO emp_msg VALUES(30,'MARTIN', 1550, 7698);
INSERT INTO emp_msg VALUES(30,'TURNER', 1800, 7698);
INSERT INTO emp_msg VALUES(30,'WARD', 1550, 7698);
```

查询 emp_msg表中 sal 列最高的第一个非空 MGR 值作为 first_MGR 列。

```
SELECT deptno , ename , sal , MGR ,
FIRST_VALUE ( MGR ) IGNORE NULLS over ( ORDER BY sal DESC ROWS UNBOUNDED PRECEDING ) AS first_MGR

FROM emp_msg ORDER BY deptno , ename;
```

查询结果如下：

```
+--------+--------+------+------+-----------+
| DEPTNO | ENAME  | SAL  | MGR  | FIRST_MGR |
+--------+--------+------+------+-----------+
|   10 | CLARK  | 2750 | 7839 | 7566      |
|   10 | KING   | 5300 | NULL | NULL      |
|   10 | MILLER | 1600 | 7782 | 7566      |
|   20 | ADAMS  | 1400 | 7788 | 7566      |
|   20 | FORD   | 3300 | 7566 | 7566      |
|   20 | JONES  | 3275 | 7839 | 7566      |
|   20 | SCOTT  | 3300 | 7566 | 7566      |
|   20 | SMITH  | 1100 | 7902 | 7566      |
|   30 | ALLEN  | 1900 | 7698 | 7566      |
|   30 | BLAKE  | 3150 | 7839 | 7566      |
|   30 | JAMES  | 1250 | 7698 | 7566      |
|   30 | MARTIN | 1550 | 7698 | 7566      |
|   30 | TURNER | 1800 | 7698 | 7566      |
|   30 | WARD   | 1550 | 7698 | 7566      |
+--------+--------+------+------+-----------+
```

6.6.4.16. LAST_VALUE

`LAST_VALUE` 是一个分析函数。它返回一组有序值中的最后一个值。如果集合中的最后一个值为 `NULL`，则该函数将返回为 `NULL`，除非您指定 `IGNORE NULLS`，此设置对数据致密化很有用。

语法

```
LAST_VALUE {(expr) [RESPECT|IGNORE NULLS] | (expr [RESPECT|IGNORE NULLS])}
OVER (analytic_clause)
```

参数

参数	说明
OVER	使用 OVER 子句定义窗口进行计算。
expr	不能将 LAST_VALUE 或其他分析函数用于 expr 来嵌套分析函数。
FROM { FIRST | LAST }	确定计算方向是从窗口的第一行还是最后一行开始，默认值为 FROM FIRST 。
{RESPECT | IGNORE} NULLS	表示是否忽略 NULL 值。默认值为 RESPECT NULLS ，考虑 NULL 值。如果您指定 IGNORE NULLS ，则 LAST_VALUE 返回集合中的最后一个非空值，如果所有值都为 NULL ，则返回 NULL 。

返回类型

返回值的数据类型不限。

示例

建表 emp_msg，并向里面插入数据，执行以下语句：

```
CREATE TABLE emp_msg(deptno INT, ename varchar(30),sal INT, MGR varchar(30));
INSERT INTO emp_msg VALUES(10,'CLARK', 2750, 7839);
INSERT INTO emp_msg VALUES(10,'KING', 5300, NULL);
INSERT INTO emp_msg VALUES(10,'MILLER', 1600, 7782);
INSERT INTO emp_msg VALUES(20,'ADAMS', 1400, 7788);
INSERT INTO emp_msg VALUES(20,'FORD', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'JONES', 3275, 7839);
INSERT INTO emp_msg VALUES(20,'SCOTT', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'SMITH', 1100, 7902);
INSERT INTO emp_msg VALUES(30,'ALLEN', 1900, 7698);
INSERT INTO emp_msg VALUES(30,'BLAKE', 3150, 7839);
INSERT INTO emp_msg VALUES(30,'JAMES', 1250, 7698);
INSERT INTO emp_msg VALUES(30,'MARTIN', 1550, 7698);
INSERT INTO emp_msg VALUES(30,'TURNER', 1800, 7698);
INSERT INTO emp_msg VALUES(30,'WARD', 1550, 7698);
```

查询 emp_msg 表中 sal 列最低的最后一个非空 MGR 值作为 last_MGR 列，执行以下语句：

```
SELECT deptno , ename , sal , MGR ,
LAST_VALUE ( MGR ) IGNORE NULLS OVER (ORDER BY sal DESC ROWS BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING ) AS last_MGR
FROM emp_msg ORDER BY deptno , ename ;
```

查询结果如下：

```
+--------+--------+------+------+-----------+
| DEPTNO | ENAME  | SAL  | MGR  | FIRST_MGR |
+--------+--------+------+------+-----------+
|   10 | CLARK  | 2750 | 7839 | 7839      |
|   10 | KING   | 5300 | NULL | NULL      |
|   10 | MILLER | 1600 | 7782 | 7782      |
|   20 | ADAMS  | 1400 | 7788 | 7788      |
|   20 | FORD   | 3300 | 7566 | 7566      |
|   20 | JONES  | 3275 | 7839 | 7839      |
|   20 | SCOTT  | 3300 | 7566 | 7566      |
|   20 | SMITH  | 1100 | 7902 | 7902      |
|   30 | ALLEN  | 1900 | 7698 | 7698      |
|   30 | BLAKE  | 3150 | 7839 | 7839      |
|   30 | JAMES  | 1250 | 7698 | 7698      |
|   30 | MARTIN | 1550 | 7698 | 7698      |
|   30 | TURNER | 1800 | 7698 | 7698      |
|   30 | WARD   | 1550 | 7698 | 7698      |
+--------+--------+------+------+-----------+
```

6.6.4.17. NTH_VALUE

`NTH_VALUE` 返回 `analytic_clause` 定义的窗口中第 `n` 行的 `measure_expr` 值。返回的值具有 `measure_expr` 的数据类型。

语法

```
NTH_VALUE (measure_expr, n) [ FROM { FIRST | LAST } ][ { RESPECT | IGNORE } NULLS ] OVER (analytic_clause)
```

参数

参数	说明

参数	说明
OVER	使用 `OVER` 子句定义窗口进行计算。
measure_expr	字段名
n	n 为正数，确定要为其返回测量值的第 n 行，如果 n 是 NULL ，函数将返回错误，如果 n 大于窗口内所有的行数，函数返回 NULL。
FROM { FIRST | LAST }	确定计算方向是从窗口的第一行还是最后一行开始，默认值为 `FROM FIRST` 。
{RESPECT | IGNORE} NULLS	表示是否忽略 `NULL` 值。默认值为 `RESPECT NULLS` ，考虑 `NULL` 值。

返回类型

返回值的数据类型不限。

示例

创建表 emp_msg，并向里面插入数据。执行以下语句：

```
CREATE TABLE emp_msg(deptno INT, ename VARCHAR(30),sal INT, MGR VARCHAR(30), hiredate VARCHAR(50)
);
INSERT INTO emp_msg VALUES(10,'CLARK', 2750, 7839, '2018-05-01');
INSERT INTO emp_msg VALUES(10,'KING', 5300, NULL, '2018-05-10');
INSERT INTO emp_msg VALUES(10,'MILLER', 1600, 7782, '2018-06-01');
INSERT INTO emp_msg VALUES(20,'ADAMS', 1400, 7788, '2018-05-21');
INSERT INTO emp_msg VALUES(20,'FORD', 3300, 7566, '2018-06-01');
INSERT INTO emp_msg VALUES(20,'JONES', 3275, 7839, '2018-06-20');
INSERT INTO emp_msg VALUES(20,'SCOTT', 3300, 7566, '2018-07-01');
INSERT INTO emp_msg VALUES(20,'SMITH', 1100, 7902, '2018-07-10');
INSERT INTO emp_msg VALUES(30,'ALLEN', 1900, 7698, '2018-08-05');
INSERT INTO emp_msg VALUES(30,'BLAKE', 3150, 7839, '2018-06-10');
INSERT INTO emp_msg VALUES(30,'JAMES', 1250, 7698, '2018-09-05');
INSERT INTO emp_msg VALUES(30,'MARTIN', 1550, 7698, '2018-10-01');
INSERT INTO emp_msg VALUES(30,'TURNER', 1800, 7698, '2019-05-01');
INSERT INTO emp_msg VALUES(30,'WARD', 1550, 7698, '2019-05-10');
```

按部门 deptno 分组，查询每个部门的人员的薪资和该部门内排名第三的薪资金额的对比。执行以下语句：

```
SELECT deptno, ename, sal, nth_value(sal, 3) OVER (PARTITION BY deptno ORDER BY sal DESC
rows BETWEEN unbounded preceding AND unbounded following)  AS third_most_sal
FROM emp_msg ORDER BY deptno,sal DESC;
```

查询结果如下：

```
+--------+--------+------+----------------+
| DEPTNO | ENAME  | SAL  | THIRD_MOST_SAL |
+--------+--------+------+----------------+
|   10 | KING   | 5300 |       1600 |
|   10 | CLARK  | 2750 |       1600 |
|   10 | MILLER | 1600 |       1600 |
|   20 | FORD   | 3300 |       3275 |
|   20 | SCOTT  | 3300 |       3275 |
|   20 | JONES  | 3275 |       3275 |
|   20 | ADAMS  | 1400 |       3275 |
|   20 | SMITH  | 1100 |       3275 |
|   30 | BLAKE  | 3150 |       1800 |
|   30 | ALLEN  | 1900 |       1800 |
|   30 | TURNER | 1800 |       1800 |
|   30 | MARTIN | 1550 |       1800 |
|   30 | WARD   | 1550 |       1800 |
|   30 | JAMES  | 1250 |       1800 |
+--------+--------+------+----------------+
```

6.6.4.18. CUME_DIST

`CUME_DIST` 计算一个值在一组值中的累积分布。返回值的范围为 `0 < CUME_DIST <= 1` 。领带值总是评估到相同的累积分布值。此函数将可以隐式转换为数字数据类型的任何数字数据类型或任何非数字数据类型作为参数。OceanBase 数据库确定具有最高数值优先级的参数，隐式地将其余参数转换为该数据类型，进行计算并返回 `NUMBER` 。

作为一个分析函数， `CUME_DIST` 计算指定值在一组值中的相对位置。 对于行 r，假设升序，r 的 cume_dist 是值低于或等于 r 值的行数除以被计算的行数（整个查询结果集或分区）。

语法

```
CUME_DIST() OVER ([ query_partition_clause ] order_by_clause)
```

参数

参数	说明
expr	`NUMBER` 类型或可隐式转换为 `NUMBER` 类型的表达式 。
{RESPECT | IGNORE} NULLS	表示是否忽略 `NULL` 值。默认值为 `RESPECT NULLS` ，考虑 `NULL` 值。

返回类型

返回值为 `NUMBER` 数据类型。

示例

创建表 emp_msg，并向里面插入数据。执行以下语句：

```
CREATE TABLE emp_msg(deptno INT, ename varchar(30), sal INT, MGR varchar(30));
INSERT INTO emp_msg VALUES(10,'CLARK', 2750, 7839);
INSERT INTO emp_msg VALUES(10,'KING', 5300, NULL);
INSERT INTO emp_msg VALUES(10,'MILLER', 1600, 7782);
INSERT INTO emp_msg VALUES(20,'ADAMS', 1400, 7788);
INSERT INTO emp_msg VALUES(20,'FORD', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'JONES', 3275, 7839);
INSERT INTO emp_msg VALUES(20,'SCOTT', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'SMITH', 1100, 7902);
INSERT INTO emp_msg VALUES(30,'ALLEN', 1900, 7698);
INSERT INTO emp_msg VALUES(30,'BLAKE', 3150, 7839);
INSERT INTO emp_msg VALUES(30,'JAMES', 1250, 7698);
INSERT INTO emp_msg VALUES(30,'MARTIN', 1550, 7698);
INSERT INTO emp_msg VALUES(30,'TURNER', 1800, 7698);
INSERT INTO emp_msg VALUES(30,'WARD', 1550, 7698);
```

返回指定列 sal 在同 deptno 中的 `CUME_DIST` 的值，执行以下语句：

```
SELECT deptno , ename , sal, cume_dist ( ) over ( partition BY deptno ORDER BY sal DESC ) "RANK"
FROM emp_msg WHERE sal>2000;
```

查询结果如下：

```
+--------+-------+------+-----------------------------------------+
| DEPTNO | ENAME | SAL  | RANK                                    |
+--------+-------+------+-----------------------------------------+
|   10 | KING  | 5300 |                                  .5 |
|   10 | CLARK | 2750 |                                   1 |
|   20 | FORD  | 3300 | .6666666666666666666666666666666666666667 |
|   20 | SCOTT | 3300 | .6666666666666666666666666666666666666667 |
|   20 | JONES | 3275 |                                   1 |
|   30 | BLAKE | 3150 |                                   1 |
+--------+-------+------+-----------------------------------------+
```

6.6.4.19. DENSE_RANK

DENSE_RANK 计算有序行组中行的秩，并将秩作为 NUMBER 返回。行列是从 1 开始的连续整数，最大的秩值是查询返回的唯一值的数目。在关系的情况下，秩值不被跳过。具有相同值的排序标准的行接收相同的秩。此函数对于顶部 n 和底部 n 报告是有用的。

作为一个分析函数， DENSE_RANK 根据 order_by_clause 中 value_exprs 的值，计算从查询返回的每一行相对于其他行的秩。

语法

```
DENSE_RANK( ) OVER([ query_partition_clause ] order_by_clause)
```

参数

参数	说明
OVER	使用 OVER 子句定义窗口进行计算。

返回类型

返回值为 NUMBER 数据类型。

示例

创建表 emp_msg，并向里面插入数据。执行以下语句：

```
CREATE TABLE emp_msg(deptno INT, ename varchar(30), sal INT, MGR varchar(30));
INSERT INTO emp_msg VALUES(10,'CLARK', 2750, 7839);
INSERT INTO emp_msg VALUES(10,'KING', 5300, NULL);
INSERT INTO emp_msg VALUES(10,'MILLER', 1600, 7782);
INSERT INTO emp_msg VALUES(20,'ADAMS', 1400, 7788);
INSERT INTO emp_msg VALUES(20,'FORD', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'JONES', 3275, 7839);
INSERT INTO emp_msg VALUES(20,'SCOTT', 3300, 7566);
INSERT INTO emp_msg VALUES(20,'SMITH', 1100, 7902);
INSERT INTO emp_msg VALUES(30,'ALLEN', 1900, 7698);
INSERT INTO emp_msg VALUES(30,'BLAKE', 3150, 7839);
INSERT INTO emp_msg VALUES(30,'JAMES', 1250, 7698);
INSERT INTO emp_msg VALUES(30,'MARTIN', 1550, 7698);
INSERT INTO emp_msg VALUES(30,'TURNER', 1800, 7698);
INSERT INTO emp_msg VALUES(30,'WARD', 1550, 7698);
```

分析函数功能示例，执行以下语句：

```
SELECT deptno, ename, sal, DENSE_RANK ( ) OVER ( partition BY deptno ORDER BY sal DESC ) "RANK"
FROM emp_msg WHERE sal>2000;
```

查询结果如下：

```
+--------+-------+------+------+
| DEPTNO | ENAME | SAL  | RANK |
+--------+-------+------+------+
|   10 | KING  | 5300 |    1 |
|   10 | CLARK | 2750 |    2 |
|   20 | SCOTT | 3300 |    1 |
|   20 | FORD  | 3300 |    1 |
|   20 | JONES | 3275 |    2 |
|   30 | BLAKE | 3150 |    1 |
+--------+-------+------+------+
```

6.6.4.20. NTILE

`NTILE` 函数将有序数据集划分为 `expr` 指示的若干桶，并为每一行分配适当的桶号。桶编号为 1 到 `expr` 。 对于每个分区， `expr` 值必须解析为正常数。如果 `expr` 是一个非整数常量，则 OceanBase 将该值截断为整数。 返回值为 `NUMBER` 。

桶中的行数最多可以相差 1，其余值（行数的其余部分除以桶）为每个桶分配一个，从桶 1 开始。如果 `expr` 大于行数，则将填充与行数相等的多个桶，其余的桶将为空。

您不能通过使用 `NTILE` 或任何其他分析函数来嵌套分析函数。 但是您可以将其他内置函数表达式用在 `expr` 中。

语法

```
NTILE(expr) OVER ([query_partition_clause] order_by_clause)
```

参数

参数	说明
expr	只能为正常数。
OVER	使用 `OVER` 子句定义窗口进行计算。

返回类型

返回 `NUMERIC` 类型数据。

示例

根据成绩将同学分成四个等级，为学生评奖。创建表 course，并向里面插入数据，执行以下语句：

```
CREATE TABLE course
(
 name VARCHAR(8),
 grade NUMBER
);
INSERT INTO course VALUES('Linda',50);
INSERT INTO course VALUES('Tan',85);
INSERT INTO course VALUES('Tom',90);
INSERT INTO course VALUES('John',95);
INSERT INTO course VALUES('Mery',55);
INSERT INTO course VALUES('Peter',60);
INSERT INTO course VALUES('Jack',65);
INSERT INTO course VALUES('Rose',70);
INSERT INTO course VALUES('Tonny',75);
INSERT INTO course VALUES('Apple',80);
COMMIT;
```

执行以下语句：

```
SELECT name, grade, ntile(4) OVER (ORDER BY grade DESC) til FROM course;
```

查询结果如下：

```
+-------+-------+------+
| NAME | GRADE | TIL |
+-------+-------+------+
| John |  95 |  1 |
| Tom  |  90 |  1 |
| Tan  |  85 |  1 |
| Apple |  80 |  2 |
| Tonny |  75 |  2 |
| Rose |  70 |  2 |
| Jack |  65 |  3 |
| Peter |  60 |  3 |
| Mery |  55 |  4 |
| Linda |  50 |  4 |
+-------+-------+------+
```

6.6.4.21. PERCENT_RANK

`PERCENT_RANK` 函数类似于 `CUME_DIST`（累积分布）函数。 它的返回值范围为 0~1。任何集合中的第一行的 `PERCENT_RANK` 函数为 0，返回值为 `NUMBER`。

语法

```
PERCENT_RANK( ) OVER ([query_partition_clause] order_by_clause)
```

参数

参数	说明
OVER	使用 `OVER` 子句定义窗口进行计算。

返回类型

返回数值数据类型。

示例

根据成绩将同学分成四个等级，为学生评奖。建表 course_rank，并向里面插入数据，执行以下语句：

```
CREATE TABLE course_rank
(
 name VARCHAR(8),
  id NUMBER
);
INSERT INTO course_rank VALUES('Linda',1);
INSERT INTO course_rank VALUES('Tan',2);
INSERT INTO course_rank VALUES('Tom',3);
INSERT INTO course_rank VALUES('John',4);
INSERT INTO course_rank VALUES('Mery',5);
COMMIT;
```

执行以下语句：

```
SELECT name, id ,percent_rank() OVER (ORDER BY id) AS pr1 FROM course_rank;
```

查询结果如下：

```
+-------+------+------+
|NAME |ID  |PR1 |
+-------+------+------+
|Linda|  1|  0|
|Tan  |  2| .25|
|Tom  |  3| .5|
|John |  4| .75|
|Mery |  5|  1|
+-------+------+------+
```

6.6.4.22. RATIO_TO_REPORT

`RATIO_TO_REPORT` 函数计算一个值与一组值之和的比率。

语法

```
RATIO_TO_REPORT(expr) OVER ([query_partition_clause])
```

参数

参数	说明
expr	只能为正常数。
OVER	使用 `OVER` 子句定义窗口进行计算。

返回类型

返回数值型数据。

示例

根据部分展示员工产出占部门总展出的比例。建表 **product**，并向里面插入数据，执行以下语句：

```
CREATE TABLE product (name VARCHAR(8), deptno NUMBER, output NUMBER);
INSERT INTO product VALUES('Linda',100,5050);
INSERT INTO product VALUES('Tan',1001,8500);
INSERT INTO product VALUES('Tom',1001,3900);
INSERT INTO product VALUES('John',100,29500);
INSERT INTO product VALUES('Mery',1001,1500);
INSERT INTO product VALUES('Peter',100,1060);
COMMIT;
```

执行以下语句：

```
SELECT name, OUTPUT, deptno, RATIO_TO_REPORT(output) OVER (partition BY deptno) FROM product;
```

查询结果如下：

```
+-------+--------+--------+-----------------------------------------------+
| NAME  | OUTPUT | DEPTNO | RATIO_TO_REPORT(OUTPUT)OVER(PARTITIONBYDEPTNO) |
+-------+--------+--------+-----------------------------------------------+
| Linda |  5050  |  100   |    .141814097163718056725638865487222690255 5 |
| John  | 29500  |  100   |    .828418983431620331367593372648132547037 3 |
| Peter |  1060  |  100   |    .029766919404661611906767761864644762707 1 |
| Tan   |  8500  |  1001  |    .611510791366906474820143884892086330935 3 |
| Tom   |  3900  |  1001  |    .280575539568345323741007194244604316546 8 |
| Mery  |  1500  |  1001  |    .107913669064748201438848920863309352518  |
+-------+--------+--------+-----------------------------------------------+
```

6.6.4.23. ROW_NUMBER

ROW_NUMBER 函数为应用它的每一行分配一个唯一的数字（无论是分区中的每一行还是查询返回的每一行），按照 order_by_clause 中指定的行的有序序列，从 1 开始，通过在检索指定范围的 ROW_NUMBER 的查询中使用子查询嵌套一个子查询，您可以从内部查询的结果中找到一个精确的行子集。 此函数的使用允许您实现 top-n 、 bottom-n 和 inner-n 报告。对于一致的结果，查询必须确保确定的排序顺序。

语法

```
ROW_NUMBER( ) OVER ([ query_partition_clause ] order_by_clause)
```

参数

参数	说明
OVER	使用 OVER 子句定义窗口进行计算。

返回类型

返回数值型数据。

示例

以部门为单位根据员工的产出进行排名。建表 product，并向里面插入数据。执行以下语句：

```
CREATE TABLE product(name VARCHAR(8), deptno NUMBER, output NUMBER);
INSERT INTO product VALUES('Linda',100,5050);
INSERT INTO product VALUES('Tan',1001,8500);
INSERT INTO product VALUES('Tom',1001,3900);
INSERT INTO product VALUES('John',100,29500);
INSERT INTO product VALUES('Mery',1001,1500);
INSERT INTO product VALUES('Peter',100,1060);
COMMIT;
```

执行以下语句：

```
SELECT name,OUTPUT,deptno,ROW_NUMBER() OVER (partition BY deptno ORDER BY OUTPUT DESC) FROM p
roduct;
```

查询结果如下：

```
+-------+--------+--------+-----------------------------------------------------+
| NAME  | OUTPUT | DEPTNO | ROW_NUMBER()OVER(PARTITIONBYDEPTNOORDERBYOUTPUTDESC) |
+-------+--------+--------+-----------------------------------------------------+
| John  | 29500  |  100   |                                                   1 |
| Linda |  5050  |  100   |                                                   2 |
| Peter |  1060  |  100   |                                                   3 |
| Tan   |  8500  |  1001  |                                                   1 |
| Tom   |  3900  |  1001  |                                                   2 |
| Mery  |  1500  |  1001  |                                                   3 |
+-------+--------+--------+-----------------------------------------------------+
```

6.7. 表达式

6.7.1. SQL 表达式概述

表达式用来计算数据的值，它由一个或多个数值、运算符和 SQL 函数等组件组合而成，通常表达式中要为它的组件假定数据类型。

> 注意
>
> NLS_COMP 与 NLS_SORT 两个参数值共同影响字符的排序与比较。如果在数据库中 NLS_COMP 的值为 LINGUISTIC，那么本章节中所有被提到的实体都会遵循参数 NLS_SORT 所指定的规则。如果 NLS_COMP 的值没有被指定为 LINGUISTIC，那么函数将不会受 NLS_SORT 的影响。NLS_SORT 的值可以被直接指定，如果没有被指定，它将继承 NLS_LANGUAGE 的值。

下面这个简单表达式的计算结果为 4，数据类型为 NUMBER （与组件的数据类型一致）。

```
2*2
```

如下是一个使用了函数和运算符的复杂表达式。这个表达式表示将当前日期增加 7 天，然后移除时间部分，最后将结果转换成 CHAR 数据类型。

```
TO_CHAR(TRUNC(SYSDATE+7))
```

您还可以在如下情形中使用表达式：

- SELECT 语句的选择列中。
- WHERE 子句中和 HAVING 子句中。
- CONNECT BY 、START WITH 和 ORDER BY 子句中。
- INSERT 语句的 VALUES 子句中。
- UPDATE 语句的 SET 子句中。

例如，在下面的 `UPDATE` 语句的 `SET` 子句中您可以使用一个表达式替换字符串 Smith：

```
SET last_name = 'Smith';
```

这个 `SET` 子句使用表达式 `INITCAP(last_name)` 替代字符串 Smith：

```
SET last_name = INITCAP(last_name);
```

OceanBase 数据库中，并不是所有表达式都可以直接被任意 SQL 语句使用，请参考 SQL语句 章节中关于表达式的限制。

6.7.2. 简单表达式

一个简单表达式（Simple Expressions）可以是列、伪列、常量、序列数或空值。

除了用户schema，`schema` 也可以是 **PUBLIC**的，用来指定一个指向表、视图的同义词的属性。注意只有在数据操纵语言（DML）语句中支持使用 **PUBLIC** 为公共同义词，而在数据定义语言（DDL）语句中则不支持。`NCHAR` 与 `NVARCHAR2` 并不是有效的伪列数据类型。

下列是一些有效的简单表达式：

- `employees.last_name`
- `'this is a text string'`
- `10`
- `N 'this is an NCHAR string'`

6.7.3. 复合表达式

复合表达式（Compound Expressions）是由其它类型表达式组合而成的表达式。

您可以使用任意内置函数作为一个表达式。然而，在复合表达式中，一些函数的组合会因为不合适而被弃用。例如，在一个聚合函数中，`LENGTH` 函数就不适用。

`PRIOR` 运算符用于层次查询的 `CONNECT BY` 子句。

下列是一些有效的复合表达式：

- `('CLARK' || 'SMITH')`
- `LENGTH('MOOSE') * 57`
- `SORT(144) + 72`
- `my_fun(TO_CHAR(sysdate, 'DD-MMM-YY'))`

6.7.4. 条件表达式

条件表达式（Case Expression）允许用户在不调用存储过程的情况下，在 SQL 语句中使用 `IF ... THEN ... ELSE` 逻辑。

语法

```
CASE { simple_case_expression
   | searched_case_expression
   }
   [ ELSE else_expr ]
   END
```

其中简单条件表达式中 `simple_case_expression` 的语法为：

```
expr
 { WHEN comparison_expr THEN return_expr }...
```

搜索条件表达式 `searched_case_expression` 的语法为：

```
{ WHEN condition THEN return_expr }...
```

使用规则

条件的验证

在一个简单条件表达式中，OceanBase 数据库会以 `expr` 为基准，在 `WHEN ... THEN` 中搜寻第一个与之等值的 `comparison_expr`，并返回相应的 `return_expr`。如果没有 `WHEN ... THEN` 满足此情形，并且 `ELSE` 子句存在，那么 OceanBase 数据库会返回 `else_expr`。否则，OceanBase 会返回 NULL。

在搜索条件表达式中，OceanBase 会自左向右搜索直到 `condition` 条件成立，然后返回 `return_expr`。如果所有条件都不成立，且 `ELSE` 子句存在，则数据库返回 `else_expr`。否则，数据库会返回 NULL。

条件的计算

OceanBase 数据库使用短路计算法则。对于简单条件表达式，数据库仅会在与 `expr` 比较之前计算 `comparison_expr` 的值，而不是在与 `expr` 比较之前就将计算所有 `comparison_expr` 的值。因此，如果前一个 `comparison_expr` 与 `expr` 相等，OceanBase 将不会计算下一个 `comparison_expr` 的值。对于搜索条件表达式，数据库将会串行计算每个条件（Condition）是否为真，如果前一个条件（Condition）为真，OceanBase 将不会计算下一个条件。

数据类型

对于简单条件表达式，`expr` 和所有的 `comparison_expr` 的值的数据类型必须相同（如 `CHAR`、`VARCHAR2` 和 `NCHAR`；`NVARCHAR2`、`NUMBER` 和 `BINARY_FLOAT`，或 `BINARY_DOUBLE`），或者都是数值类型。如果所有的返回表达式都是数值类型，则 OceanBase 会选择最高优先级的数据类型，显式地将其它参数转化为此数据类型，并返回这个数据类型。

对于简单条件表达式和搜索条件表达式，所有的 `return_exprs` 的数据类型必须相同（如 `CHAR`、`VARCHAR2` 和 `NCHAR`；`NVARCHAR2`、`NUMBER` 和 `BINARY_FLOAT`；`BINARY_DOUBLE`），或者都是数值类型。如果所有的返回表达式都是数值类型，则 OceanBase 会选择最高优先级的数据类型，显式地将其它参数转化为此数据类型，并返回这个数据类型。

示例

```
SELECT cust_last_name,
  CASE credit_limit
  WHEN 100 THEN 'Low'
  WHEN 5000 THEN 'High'
  ELSE 'Medium' END AS credit
FROM customer
ORDER BY cust_last_name, credit;
```

以下是搜索条件表达式的示例：

```
SELECT AVG(CASE WHEN e.salary > 2000 THEN e.salary
      ELSE 2000 END) "Average Salary" FROM employee e;
```

6.7.5. 列表达式

列表达式（Column Expressions）是一种形式受限制的 `expr` ，在本章其他表达式的语法中被命名为 `column_expression` 。列表达式可以是 简单表达式、复合表达式、函数表达式 或者 表达式列表，但它只能包含以下形式的表达式：

- 目标表（被创建，变更或索引的表）的列。
- 常量（字符串或数字）。
- 确定性函数（SQL 内建函数或用户自定义函数）。

除以上形式表达式外，其它形式的表达式都不是有效的列表达式。此外，列表达式不支持使用 `PRIOR` 关键字的复合表达式与聚合函数。

使用列表达式可以实现以下目的：

- 创建基于函数的索引。
- 显式或隐式定义一个虚拟列。定义一个虚拟列时， `column_expression` 只适用于在此前语句中已经定义的目标表的列。

列表达式的组件必须是确定的，也就是说，输入同样的值必须返回同样的输出值。

6.7.6. 日期时间表达式

日期时间表达式（Datetime Expression）会生成日期时间数据类型的值。

语法如下：

```
{TIMESTAMP | DATE} string
```

TIMESTAMP或DATE加上字符串字面量可以生成TIMESTAMP或DATE类型的值，注意字符串的格式要与系统变量NLS_TIMESTAMP_FORMAT和NLS_DATE_FORMAT一致。

系统变量的值可以通过下面的SQL查询：

```
select * from v$nls_parameters where parameter like '%FORMAT';
```

例如在系统变量NLS_TIMESTAMP_FORMAT的值为YYYY-MM-DD HH24:MI:SS.FF，可以用下面的表达式生成一个timestamp类型的值。

```
select timestamp '2020-01-01 10:00:00' from dual;
+------------------------------+
| TIMESTAMP'2020-01-0110:00:00' |
+------------------------------+
| 2020-01-01 10:00:00.000000000 |
+------------------------------+
```

6.7.7. 函数表达式

函数表达式（Function Expression）使您可以使用任何内置 SQL 函数或是自定义函数表达式。

如下所示的是一些有效的内置函数表达式：

- `LENGTH('BLAKE')`
- `ROUND(1234.567*43)`
- `SYSDATE`

> **注意**
>
> 自定义函数表达式不能传递对象类型或 XMLType 的参数到远程函数或过程。

自定义函数表达式会调用：

- 一个自定义包、类型或一个独立自定义函数中的函数。
- 一个自定义函数或运算符。

下列是一些有效的自定义函数表达式：

- `circle_area(radius)`
- `payroll.tax_rate(empno)`
- `hr.employees.comm_pct@remote(dependents, empno)`
- `DBMS_LOB.getlength(column_name)`
- `my_function(a_column)`

使用自定义函数作为表达式时，支持位置表示法、名字表示法和混合表示法。例如，下面的表示法都是正确的：

```
CALL my_function(arg1 => 3, arg2 => 4) ...
CALL my_function(3, 4) ...
CALL my_function(3, arg2 => 4) ...
```

6.7.8. 间隔表达式

时间间隔表达式（Interval Expression）生成 `INTERVAL YEAR TO MONTH` 或 `INTERVAL DAY TO SECOND` 数据类型的值。

语法如下：

```
INTERVAL string
  { DAY [ (leading_field_precision) ] TO
    SECOND [ (fractional_second_precision) ]
  | YEAR [ (leading_field_precision) ] TO
    MONTH
  }
```

`leading_field_precision` 和 `fractional_second_precision` 可以是 0 到 9 的任意整数，这两个参数指定了对应元素值的精确度。如果在指定 `DAY` 和 `YEAR` 元素时省略了 `leading_field_precision` 参数，其使用默认值为 2，表示该元素的值不能超过 2 位整数。如果对 `SECOND` 元素省略了

`fractional_second_ precision` 参数，其默认值为 6，表示该值精确到小数点第6位。如果查询的返回值比默认精度位数更多，OceanBase 会返回一个错误。

例如，下面的语句展示了一个DAY TO SECOND类型的INTERVAL值。

```
select INTERVAL '999999999 23:59:59.999' day(9) to second from dual;
```

6.7.9. 标量子查询表达式

标量子查询表达式（Scalar Subquery Expression）是返回一行包含多列值的子查询。如果子查询返回 0 行，则标量子查询表达式的值是 NULL。如果子查询返回多行，则标量子查询表达式返回一个错误。大多数表达式都可以使用标量子查询表达式作为参数。

标量子查询在以下情形中是无效的表达式：

- 作为列的默认值。
- 用于哈希函数中。
- 在 DML 语句的 `RETURNING` 子句中。
- 在函数索引的定义中。
- 在 `CHECK` 约束中。
- 在 `GROUP BY` 子句中。
- 在与查询无关的语句中，如 `CREATE PROFILE` 。

6.7.10. 表达式列表

表达式列表（Expression List）是一组其它表达式的组合。

表达式列表可以出现在比较和成员条件，以及查询和子查询的 GROUP BY 子句中。在比较和成员条件中的表达式列表有时被称为行值构造器（Row value constructor）或者行构造器（Row constructor）。

比较和成员条件出现在 WHERE 子句中。它们可以包含一个或多个逗号分隔的表达式，或是一组或多组表达式，其中每组表达式包含一个或多个逗号分隔的表达式。在接下来的例子中（多组表达式）：

- 每一组被括号包含。
- 每一组必须包含相同数量的表达式。
- 每一组中表达式的数量需要与比较条件中运算符之前的表达式的数量匹配，或者与成员条件中 IN 关键字之前的表达式的数量匹配。

逗号分隔的表达式列表最多只能包含 1000 个表达式。逗号分隔的表达式组列表可以包含任意数量表达式组，但是每一表达式组最多只能包含 1000 个表达式。

下例是一些有效的表达式列表：

```
(10, 20, 40)
('SCOTT', 'BLAKE', 'TAYLOR')
( ('Guy', 'Himuro', 'GHIMURO'),('Karen', 'Colmenares', 'KCOLMENA') )
```

在第三个例子中，每一组的表达式数量必须与 SQL 语句条件的第一部分的表达式数量相等。例如：

```
SELECT * FROM employees
  WHERE (first_name, last_name, email) IN
    (('Guy', 'Himuro', 'GHIMURO'),('Karen', 'Colmenares', 'KCOLMENA'));
```

在简单的 GROUP BY 子句中，使用大写或小写形式的表达式列表都可以：

```
SELECT department_id, MIN(salary) min, MAX(salary) max FROM employees
  GROUP BY department_id, salary
  ORDER BY department_id, min, max;

SELECT department_id, MIN(salary) min, MAX(salary) max FROM employees
  GROUP BY (department_id, salary)
  ORDER BY department_id, min, max;
```

在 GROUP BY 子句的 ROLLUP 、 CUBE 和 GROUPING SETS 子句中，您可以在同一表达式列表中将单个表达式与表达式组结合。下例展示了一些在 SQL 语句中有效分组表达式列表：

```
SELECT prod_category, prod_subcategory, country_id, cust_city, count(*)
  FROM products, sales, customers
  WHERE sales.prod_id = products.prod_id
    AND sales.cust_id=customers.cust_id
    AND sales.time_id = '01-oct-00'
    AND customers.cust_year_of_birth BETWEEN 1960 and 1970
  GROUP BY GROUPING SETS (
    (prod_category, prod_subcategory, country_id, cust_city), (prod_category, prod_subcategory, country
_id),    (prod_category, prod_subcategory),
    country_id
    )
  ORDER BY prod_category, prod_subcategory, country_id, cust_city;
```

6.8. 条件

6.8.1. SQL 条件概述

条件用来判断数据的值，并返回 `TRUE` 、 `FALSE` 或 `UNKNOWN` 。它由一个或多个表达式和逻辑（布尔）运算符等组件组合而成。只要条件出现在 SQL 语句中，您就必须使用正确的条件语法。

> 注意
>
> 参数 `NLS_COMP` 和 `NLS_SORT` 共同影响字符的排序与比较。若将数据库的 `NLS_COMP` 指定为 `LINGUISTIC` ，那么本开发指南中所有被提到的实体都会遵循参数 `NLS_SORT` 所指定的规则。若 `NLS_COMP` 的值没有被指定为 `LINGUISTIC` ，那么函数将不受 `NLS_SORT` 影响。 `NLS_SORT` 的值可以被直接指定，如果没有被指定，它将继承 `NLS_LANGUAGE1` 的值。

您可以在这些语句的 `WHERE` 子句中使用条件：

- `DELETE`
- `SELECT`
- `UPDATE`

您也可以在 `SELECT` 语句的这些子句中使用条件：

- `WHERE`
- `START WITH`
- `CONNECT BY`
- `HAVING`

条件可以称为逻辑数据类型，尽管 OceanBase 数据库不正式支持这样的数据类型。

例如，简单条件 1 = 1 判断结果为 TRUE。

下面更复杂的条件将 salary 的值加上 commission_pct 得值（函数 NVL 将 salary 中的 NULL 值替换为 0），并判断总和是否大于 25000。

```
NVL(salary, 0) + NVL(salary + (salary * commission_pct, 0) > 25000)
```

逻辑条件 AND 可以将多个条件组合成一个条件。

```
(1 = 1) AND (5 < 7)
```

SQL 语句中的有效条件：

```
name = 'SMITH'
employees.department_id = departments.department_id
hire_date > '01-JAN-08'
job_id IN ('SA_MAN', 'SA_REP')
salary BETWEEN 5000 AND 10000
commission_pct IS NULL AND salary = 2100
```

条件优先

条件优先级指的是 OceanBase 数据库在同一表达式中判断不同条件的顺序。当计算包含多个条件的表达式时，先判断较高优先级的条件，最后判断较低优先级的条件，而优先级相等的条件则按照从左到右的顺序判断。例如 AND 和 OR 连接的多个条件，不能按照从左到右的顺序判断，而是按照先计算 AND 然后计算 OR 的顺序。

SQL 条件优先级表

条件类型	功能
=、!=、<、>、<=、>=	比较。
IS [NOT] NULL、LIKE、[NOT] BETWEEN、[NOT] IN、EXISTS、IS OF 类型	比较。
NOT	取幂，逻辑取反。
AND	比较。
OR	分离。

优先级按照从高到低排列出，同一行上列出的条件具有相同的优先级。

6.8.2. 比较条件

比较条件（Comparison Conditions）用于比较一个表达式与另一个表达式，比较的结果为 `TRUE`、`FALSE` 或 `UNKNOWN`。

比较条件不能比较大对象（LOB）数据类型。但是您可以通过 PL 程序对 `CLOB` 数据进行比较。

比较数值表达式时，OceanBase 使用数值优先级来确定条件中的数值比较顺序，比如比较符两边出现不同类型的 数值时，如果有 `BINARY_DOUBLE` 则按照 `BINARY_DOUBLE` 进行比较， 其次是 `BINARY_FLOAT`， 最后是 `NUMBER`。是比较 `NUMBER`、`BINARY_FLOAT` 还是 `BINARY_DOUBLE` 值。

比较字符表达式时，OceanBase 使用字符数据类型比较规则中的规定。它规定了表达式的字符集在比较之前是如何统一的，选择使用二进制和语言比较，还是选择使用空白填充和非空白填充的比较语义。

当使用比较条件对字符值进行语言比较时，首先将它们转换为排序键，然后做比较，比较的过程类似 `RAW` 数据类型。排序键是由函数 `NLSSORT` 返回的值。如果两个表达式生成的排序键的前缀相同，即使它们在值的其余部分不同，它们也可以在语言上相等。

如果两个非标量类型的对象具有相同的命名类型，并且它们的元素之间存在一一对应的关系，则它们是可比较的。用户定义的对象类型—嵌套表，在相等或 `IN` 条件下使用时，必须定义 `MAP` 方法。嵌套表的元素可以比较。

简单比较条件

一个简单比较条件可以是单个表达式与表达式列表比较或单个表达式与子查询结果比较。

简单比较条件语法如下：

```
expr {=|!=|^=|<>|<|>=|} (expression_list|subquery)
```

在简单比较条件中，如果是单个表达式与表达式列表比较，并且表达式列表中的表达式必须在数量和数据类型上与运算符左边的表达式匹配。如果是单个表达式与子查询的值比较，则子查询返回的值必须与运算符左侧的表达式在数量和数据类型上匹配。

组比较条件

一个组比较条件可以是单个表达式与表达式列表或子查询结果的任何或所有成员比较，也可以是多个表达式与表达式列表或子查询结果的任何或所有成员比较。

在组比较条件中，如果是单个表达式或多个表达式列表与表达式列表的任何或所有成员比较，则每个表达式列表中的表达式必须在数量和数据类型上与运算符左边的表达式匹配。如果是单个表达式或多个表达式与子查询结果的任何或所有成员比较，则子查询返回的值必须与运算符左侧的表达式在数量和数据类型上匹配。

组比较条件语法有如下两种：

```
expr {=|!=|^=|<>|<|>=|ANY|SOME|ALL}({expression_list|subquery})
```

```
(expr[,expr]...){=|!=|^=|ANY|SOME|ALL}({expression_list[,expression_list]...|subquery})
```

6.8.3. 逻辑条件

逻辑条件（Logical Conditions）将两个条件组合在一起，产生单个结果或反转单个条件的结果。

逻辑条件 NOT

逻辑条件 `NOT` 表示‘非’，可以反转单个条件的结果。如果条件为 `FALSE` ，则返回 `TRUE` 。如果条件为 `TRUE` ，则返回 `FALSE` 。如果它是 `UNKNOWN` ，则返回 `UNKNOWN` 。

逻辑条件 NOT 示例

```
SELECT * FROM employees WHERE NOT (job_id IS NULL) ORDER BY employee_id;
SELECT * FROM employees WHERE NOT (salary BETWEEN 1000 AND 2000) ORDER BY employee_id;
```

逻辑条件 AND

逻辑条件 `AND` 表示‘与’，用于连接两个条件。如果两个条件均为 `TRUE` ，则返回 `TRUE` 。如果任意一个为 `FALSE` ，则返回 `FALSE` 。否则返回 `UNKNOWN` 。

逻辑条件 AND 示例

```
SELECT * FROM employees WHERE job_id = 'PU_CLERK' AND department_id = 30 ORDER BY employee_id;
```

逻辑条件 OR

逻辑条件 `OR` 表示‘或’，表示任何一个都可以。如果任一条件为 `TRUE` ，则返回 `TRUE` 。如果两者均为 `FALSE` ，则返回 `FALSE` 。否则返回 `UNKNOWN` 。

逻辑条件 OR 示例

```
SELECT * FROM employees WHERE job_id = 'PU_CLERK' OR department_id = 10 ORDER BY employee_id;
```

6.8.4. 模式匹配条件

模式匹配条件（Pattern-matching Conditions）用来比较字符数据。

LIKE 条件

`LIKE` 条件用于模式匹配。相等运算符（=）指的是一个字符值与另一个字符值完全匹配，而 `LIKE` 条件通过在第一个值中搜索由第二个字符值指定的模式来将一个字符值的一部分与另一个字符值进行匹配。

`LIKE` 使用输入字符集定义的字符来计算字符串。

语法

```
char1 [NOT] LIKE char2 [ ESCAPE esc_char ]
```

除了 `LIKE` ，特殊的模式匹配字符 `_` 表示值中的一个字符恰好匹配， `%` 表示值中的零个或多个字符匹配。模式 `%` 不能与 `NULL` 匹配。

参数

参数	说明
char1	字符表达式，例如字符列，称为搜索值。
char2	字符表达式，通常是文字，称为模式。
esc_char	字符表达式，通常是文字，`ESCAPE` 将 `esc_char` 标识为转义字符。当转义符置于模式匹配符之前时，该模式匹配符被解释为普通的字符。

示例

以下语句使用了 `LIKE`：

```
SELECT last_name FROM employees WHERE last_name LIKE '%A\_B%' ESCAPE '\' ORDER BY last_name;
```

`ESCAPE '\'` 会将 `%A\_B%` 中 `\` 后面的模式匹配符‘_’解释为普通的字符。

```
SELECT salary FROM employees WHERE 'SM%' LIKE last_name ORDER BY salary;
```

REGEXP_LIKE 条件

`REGEXP_LIKE` 用于正则表达式匹配。`REGEXP_LIKE` 使用输入字符集定义的字符评估字符串。

语法

```
REGEXP_LIKE(source_char, pattern [, match_param ])
```

参数

参数	说明
source_char	用作搜索值的字符表达式，数据类型可以是 `CHAR`，`VARCHAR2`，`NCHAR`，`NVARCHAR2` 或 `CLOB`。
pattern	正则表达式，数据类型可以是 `CHAR`，`VARCHAR2`，`NCHAR`，`NVARCHAR2` 或 `CLOB`。

参数	说明
source_char	数据类型 VARCHAR2 或 CHAR 的字符表达式，允许您更改条件的默认匹配行为。

如果模式的数据类型与 source_char 的数据类型不同，OceanBase 将模式转换为 source_char 的数据类型。

示例

创建表 employees，并向里面插入数据。执行以下语句：

```
CREATE TABLE employees(manager_id INT, first_name varchar(50), last_name varchar(50), hiredate varchar(
50),SALARY INT);
INSERT INTO employees VALUES(300, 'Steven', 'King', '2019-09-11',23600);
INSERT INTO employees VALUES(200, 'Steven', 'Markle', '2019-11-05', 23800);
INSERT INTO employees VALUES(100, 'Deven', 'Part', '2018-10-01',24000);
INSERT INTO employees VALUES(200, 'Carlos', 'Ross', '2019-06-11',23500);
INSERT INTO employees VALUES(200, 'Teven', 'Bell', '2019-05-25', 23000);
INSERT INTO employees VALUES(200, 'Stephen', 'Stiles', '2018-06-11',24500);
INSERT INTO employees VALUES(100, 'Ame', 'De Haan', '2018-05-01',11000);
INSERT INTO employees VALUES(100, 'Jon', 'Errazuriz', '2017-07-21', 1400);
COMMIT;
```

查询返回名字为 Steven 或 Stephen 的员工的名字和姓氏（其中 first_name 以 Ste 开头，以 en 结尾，中间是 v 或 ph），执行以下语句：

```
SELECT first_name, last_name FROM employees WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$')
ORDER BY first_name, last_name;
```

查询结果如下：

```
+------------+-----------+
| FIRST_NAME | LAST_NAME |
+------------+-----------+
| Stephen    | Stiles    |
| Steven     | King      |
| Steven     | Markle    |
+------------+-----------+
```

6.8.5. 空条件

空条件（Null Conditions）检测 NULL 值。这是用于检测 NULL 的唯一条件。

语法

```
expr IS [ NOT ] NULL
```

示例

执行以下语句：

```
SELECT last_name FROM employees WHERE commission_pct IS NULL
ORDER BY last_name;
```

6.8.6. 复合条件

复合条件（Compound Conditions）指的是其他条件的组合。

语法

```
{ (condition) | NOT condition | condition { AND | OR } condition }
```

更多关于条件 `NOT` 、 `AND` 和 `OR` 的信息，请参考文档 逻辑条件。

6.8.7. BETWEEN 条件

`BETWEEN` 条件（Between Condition）确定一个表达式的值是否在其他两个表达式定义的间隔内。

语法

```
expr1 [ NOT ] BETWEEN expr2 AND expr3

NOT (expr1 BETWEEN expr2 AND expr3)
```

`expr1` 、 `expr2` 和 `expr3` 这三个表达式都必须是数字、字符或日期时间表达式。在 SQL 中，可能会对 `expr1` 进行多次计算。如果在 PL/SQL 中出现表达式，则保证只计算一次 `expr1` 。如果表达式不都是相同的数据类型，那么 OceanBase 数据库将表达式隐式转换为统一的数据类型。否则返回错误。

示例

查询薪资在 2000~3000 范围内的员工信息，并按照员工号排序。

```
SELECT * FROM employees WHERE salary BETWEEN 2000 AND 3000 ORDER BY employee_id;
```

6.8.8. 存在条件

存在条件（EXISTS Conditions）用于测试子查询中是否存在指定的行。

语法

```
EXISTS (subquery)
```

如果子查询至少返回一行，则说明存在您想要的数据。

示例

```
SELECT department_id FROM departments d WHERE EXISTS (SELECT * FROM employees e
WHERE d.department_id = e.department_id) ORDER BY department_id;
```

6.8.9. IN 条件

`IN` 条件（IN Conditions）是成员资格条件。它测试值或子查询列表中成员的值。

语法

```
expr [ NOT ] IN  ({ expression_list | subquery })
|
( expr [, expr ]... ) [ NOT ]  IN  ( { expression_list [, expression_list ]... | subquery } )
```

`IN` 条件可以测试表达式是表达式列表或子查询的成员，或者多个表达式是表达式列表或子查询的成员。并且每个表达式列表中的表达式必须在数量和数据类型上与运算符 `IN` 左边的表达式匹配。

示例

IN 示例：相当于 `=ANY`，表示集合中所有的成员。

```
SELECT * FROM employees WHERE job_id IN ('PU_CLERK','SH_CLERK') ORDER BY employee_id;

SELECT * FROM employees WHERE salary IN (SELECT salary  FROM employees
WHERE department_id =30) ORDER BY employee_id;
```

NOT IN 示例：相当于 `!= ALL`。如果集合中的任何成员为 `NULL`，则计算为 `false`。

```
SELECT * FROM employees WHERE salary NOT IN (SELECT salary FROM employees
WHERE department_id = 30) ORDER BY employee_id;

SELECT * FROM employees WHERE job_id NOT IN ('PU_CLERK', 'SH_CLERK')
ORDER BY employee_id;
```

6.9. 查询和子查询

6.9.1. 查询和子查询概述

查询是指数据库中用来获取数据的方式，它可搭配条件限制的子句（如 WHERE ），排列顺序的子句（如 ORDER BY ）等语句来获取查询结果。子查询是指嵌套在一个上层查询中的查询。上层的查询一般被称为父查询或外部查询。子查询的结果作为输入传递回父查询或外部查询。父查询将这个值结合到计算中，以便确定最后的输出。SQL 语言允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询。同时，子查询可以出现在 SQL 语句中的各种子句中，比如 SELECT 语句、 FROM 语句和 WHERE 语句等。下列为 SQL 语句中常见的查询：

- 简单查询
- 层次查询
- 集合
- 连接
- 子查询

简单查询（Simple Queries）

简单查询指从 OceanBase 一个或多个选择列表或视图中检索一个或多个列数据的操作，列的数量以及它们的数据类型和长度由选择列表的元素确定。而选择列表指的是 SELECT 关键字之后和 FROM 子句之前的表达式列表。

层次查询（Hierarchical Query）

层次查询是一种具有特殊功能的查询语句，通过它能够将分层数据按照层次关系展示出来。分层数据是指关系表中的数据之间具有层次关系。

集合

您可以使用集合运算符 UNION 、 UNION ALL 、 INTERSECT 和 MINUS 来组合多个查询。 所有集合运算符都具有相同的优先级。如果 SQL 语句包含多个集合运算符，则 OceanBase 从左到右对它们进行判断，除非括号中指定了顺序。本节主要讲了以下集合运算符：

运算符	说明
UNION	返回两个结果集的并集，并且不重复。
UNION ALL	返回两个结果集的并集，并且可以重复。
INTERSECT	返回两个结果集的交集。
MINUS	返回两个结果集的差集。

连接 （Join）

连接（Join）是将来自两个或多个表、视图或实例视图的行组合在一起的查询。 每当查询的 FROM 子句中出现多个表时，OceanBase 数据库执行连接。 查询的选择列表可以从其中任何表中选择任何列。 如果这两个表都有一个列名，那么您必须用表名限定查询过程中对这些列的所有引用。本节主要讲了以下连接：

连接类型	表示	说明
等值连接	Equijoins	包含等式运算符连接条件的连接。
自连接	SELF-JOIN	表与其自身的连接。
内连接	INNER JOIN	内连接，结果为两个连接表中的匹配行的连接。
左（外）连接	LEFT [OUTER] JOIN	结果包括左表（出现在 JOIN 子句最左边）中的所有行，不包括右表中的不匹配行。
右（外）连接	RIGHT [OUTER] JOIN	结果包括右表（出现在 JOIN 子句最右边）中的所有行，不包括有左表中的不匹配的行。
全（外）连接	FULL [OUTER] JOIN	结果包括所有连接中的所有行，不论他们是否匹配。
SEMI 连接	SEMI-JOIN	SEMI-JOIN 只能通过子查询展开得到。
ANTI 连接	ANTI-JOIN	ANTI-JOIN 也只能通过子查询展开得到。
笛卡儿积	Cartesian Products	当两个表没有连接操作时，对这两个表进行查询得到的数据是这两个表的笛卡儿积。

子查询

子查询指的是 SELECT 查询语句中嵌套了另一个或者多个 SELECT 语句，可以返回单行结果、多行结果或不返回结果。 SELECT 语句的 FROM 子句中的子查询也称为内联视图。您可以在嵌入式视图中嵌套任意数量的子查询。 SELECT 语句的 WHERE 子句中的子查询也称为嵌套子查询。

6.9.2. 简单查询

简单查询指从 OceanBase 一个或多个表或视图中检索一个或多个列数据的操作，列的数量以及它们的数据类型和长度由表结构确定。而选择列表指的是 `SELECT` 关键字之后和 `FROM` 子句之前的表达式列表。

语法

```
SELECT 列名1,列名2,列名3,… FROM 表;
```

表名、字段名和关键字 `SELECT` 、 `FROM` 不区分大小写。查询的最后可以跟上分号（;），多条 SQL 可以同时执行。您可以使用 `SELECT` 语句中的 Hint 将指令或提示传递给 OceanBase 数据库优化器。优化器使用 Hint 为语句选择执行计划。

示例

创建一张员工表 employee，并向列 employee_id、first_name、last_name、manager_id 和 salary 插入数据：

```
CREATE TABLE employee (
employee_id INT,
first_name VARCHAR(50),
last_name VARCHAR(50),
manager_id INT,
salary NUMERIC
);
INSERT INTO employee VALUES(111, 'DEL', 'FA BEN', 1, 1500);
INSERT INTO employee VALUES(112, 'AXEL', 'BELL', 1, 1000);
INSERT INTO employee VALUES(113, 'CRIS',  'RACHAR', 1, 1000);
```

简单查询示例

- 查询部分列：

```
SELECT first_name, last_name, salary FROM employee;
+------------+-----------+--------+
| FIRST_NAME | LAST_NAME | SALARY |
+------------+-----------+--------+
| DEL        | FA BEN    |  1500 |
| AXEL       | BELL      |  1000 |
| CRIS       | RACHAR    |  1000 |
+------------+-----------+--------+
```

- 查询所有列：

```
SELECT * FROM employee;
+-------------+------------+-----------+------------+--------+
| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | MANAGER_ID | SALARY |
+-------------+------------+-----------+------------+--------+
|         111 | DEL        | FA BEN    |          1 |   1500 |
|         112 | AXEL       | BELL      |          1 |   1000 |
|         113 | CRIS       | RACHAR    |          1 |   1000 |
+-------------+------------+-----------+------------+--------+
```

- 对列进行数学运算：

```
SELECT salary+100 FROM employee;
+------------+
| SALARY+100 |
+------------+
|       1600 |
|       1100 |
|       1100 |
+------------+
```

- 给列取别名：

```
SELECT salary*12 年薪 FROM employee;
+--------+
| 年薪   |
+--------+
|  18000 |
|  12000 |
|  12000 |
+--------+
```

- 字符串拼接：

```
SELECT first_name || '-' || last_name  AS 姓名 FROM employee;
+-------------+
| 姓名        |
+-------------+
| DEL-FA BEN  |
| AXEL-BELL   |
| CRIS-RACHAR |
+-------------+
```

- 数据去重：

```
SELECT DISTINCT MANAGER_ID FROM employee;
+------------+
| MANAGER_ID |
+------------+
|       1|
+------------+
```

- CASE WHEN 语句：

```
SELECT salary, CASE WHEN salary >= 1000 then '高薪' WHEN salary >= 800 THEN '一般'
ELSE '继续努力' END AS typeFROM employee;
+--------+--------+
| SALARY | TYPE   |
+--------+--------+
|  1500 | 高薪   |
|  1000 | 高薪   |
|  1000 | 高薪   |
+--------+--------+
```

6.9.3. 层次查询

层次查询（Hierarchical Query）是一种具有特殊功能的查询语句，通过它能够将分层数据按照层次关系展示出来。分层数据是指关系表中的数据之间具有层次关系。这种关系在现实生活中十分常见，例如：

- 组织架构中组长和组员之间的关系。
- 企业中上下级部门之间的关系。
- Web 网页中，页面跳转的关系。

语法

```
SELECT [level], column, expr... FROM table [WHERE condition] [ START WITH start_expression ]
CONNECT BY [NOCYCLE] { PRIOR child_expr = parent_expr | parent_expr = PRIOR child_expr }
[ ORDER SIBLINGS BY …] [ GROUP BY … ] [ HAVING … ] [ ORDER BY … ]
```

参数

参数	说明
LEVEL	节点的层次，是伪列，表示等级。由查询的起点开始算起为 1，依次类推。

参数	说明
CONNECT_BY_ISLEAF	当前数据行是否是层次关系中的叶子节点，是伪列，0表示不是，1表示是。
CONNECT_BY_ISCYCLE	当前数据行是否在循环中，是伪列，0表示不是，1表示是。
CONNECT_BY_ROOT运算符	CONNECT_BY_ROOT是一元运算符，表示参数中的列来自于层次查询的根节点，与一元的 + 和 - 具有相同的优先级。
condition	条件。
CONNECT BY	指明如何来确定父子关系，这里通常使用等值表达式，但其他表达式同样支持。
START WITH	指明层次查询中的根行（Root row）。
PRIOR 运算符	PRIOR 是一元运算符，表示参数中的列来自于父行（Parent row），与一元的 + 和 - 具有相同的优先级。
NOCYCLE	当指定该关键字时，即使返回结果中有循环仍旧可以返回，并可以通过 CONNECT_BY_ISCYCLE 虚拟列来指明哪里出现循环；否则，出现循环会给客户端报错。
ORDER SIBLINGS BY	指定同一个层级行之间的排列顺序。

执行流程

使用和实现层次查询最关键是要理解其执行流程，层次查询执行流程：

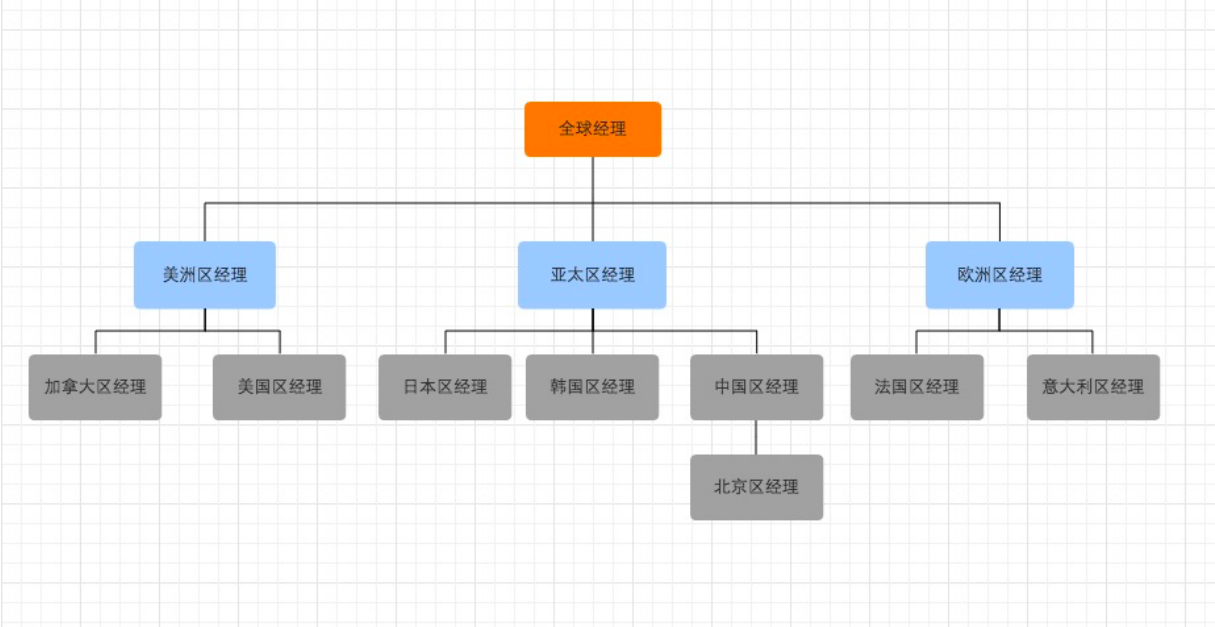
1. 执行 FROM 后面的 SCAN 或 JOIN 操作；
2. 根据 START WITH 和 CONNECT BY 的内容生成层次关系结果；
3. 按照常规查询执行流程执行剩下的子句（例如 WHERE 、 GROUP 、 ORDER BY ……）对于 2 中生成层次关系的流程可以理解为：
4. 根据 START WITH 中的表达式得到根行（Root rows）。
5. 根据 CONNECT BY 中的表达式 选择每个根行（Root rows）的子行（Child rows）。
6. 将 2 中生成的子行（Child rows）作为新的根行（Root rows）进一步生成子行（Child rows），周而复始直到没有新行生成。

示例

展示层次查询的使用，向表 emp 中的 emp_id、position 和 mgr_id 列插入数据。执行以下语句：

```
CREATE TABLE emp(emp_id INT,position VARCHAR(50),mgr_id INT);
INSERT INTO emp VALUES (1,'全球经理',NULL);
INSERT INTO emp VALUES (2,'欧洲区经理',1);
INSERT INTO emp VALUES (3,'亚太区经理',1);
INSERT INTO emp VALUES (4,'美洲区经理',1);
INSERT INTO emp VALUES (5,'意大利区经理',2);
INSERT INTO emp VALUES (6,'法国区经理',2);
INSERT INTO emp VALUES (7,'中国区经理',3);
INSERT INTO emp VALUES (8,'韩国区经理',3);
INSERT INTO emp VALUES (9,'日本区经理',3);
INSERT INTO emp VALUES (10,'美国区经理',4);
INSERT INTO emp VALUES (11,'加拿大区经理',4);
INSERT INTO emp VALUES (12,'北京区经理',7);
```

通过上面的内容可以看见列 position 具有清晰的层次关系。树状结构如下：



是按照层次结构将结果展示出来，执行以下语句：

```
SELECT emp_id, mgr_id, position, level FROM emp
START WITH mgr_id IS NULL CONNECT BY PRIOR emp_id = mgr_id;
```

查询结果如下：

```
+--------+--------+------------------+-------+
| EMP_ID | MGR_ID | POSITION         | LEVEL |
+--------+--------+------------------+-------+
|    1 |   NULL | 全球经理         |     1 |
|    2 |      1 | 欧洲区经理       |     2 |
|    5 |      2 | 意大利区经理     |     3 |
|    6 |      2 | 法国区经理       |     3 |
|    3 |      1 | 亚太区经理       |     2 |
|    7 |      3 | 中国区经理       |     3 |
|   12 |      7 | 北京区经理       |     4 |
|    8 |      3 | 韩国区经理       |     3 |
|    9 |      3 | 日本区经理       |     3 |
|    4 |      1 | 美洲区经理       |     2 |
|   10 |      4 | 美国区经理       |     3 |
|   11 |      4 | 加拿大区经理     |     3 |
+--------+--------+------------------+-------+
```

如果仅查询“亚太区”的层次结构，执行以下语句：

```
SELECT emp_id, mgr_id, position, level FROM emp START WITH position = '亚太区经理' CONNECT BY PRIOR e
mp_id = mgr_id;
```

查询结果如下：

```
+--------+--------+----------------+-------+
| EMP_ID | MGR_ID | POSITION       | LEVEL |
+--------+--------+----------------+-------+
|    3 |      1 | 亚太区经理     |     1 |
|    7 |      3 | 中国区经理     |     2 |
|   12 |      7 | 北京区经理     |     3 |
|    8 |      3 | 韩国区经理     |     2 |
|    9 |      3 | 日本区经理     |     2 |
+--------+--------+----------------+-------+
```

6.9.4. 集合

您可以使用集合运算符 `UNION` 、 `UNION ALL` 、 `INTERSECT` 和 `MINUS` 来组合多个查询。 所有集合运算符都具有相同的优先级。如果 `SQL` 语句包含多个集合运算符，则 OceanBase 从左到右对它们进行判断，除非括号中指定了顺序。

集合运算符的规则和限制

您可以指定 UNION 的属性为 ALL 和 DISTINCT 或 UNIQUE 。分别代表集合可重复，和集合不可重复。而其它几种集合操作是不能指定 ALL 属性的（它们只有 DISTINCT 属性）。所有的集合操作默认的属性是 DISTINCT 。在 Oceanbase 中，集合操作中可以指定 ORDER BY 和 LIMIT 子句，但是不允许其他子句的出现。我们仅支持 MINUS ，不支持 EXCEPT 尽管这两者的语义是相同的。参加集合操作的各查询结果的列数和相应表达式的数量必须相同，对应的数据类型也必须兼容（例如数值或字符）。

集合运算符规则

如果组件查询是字符数据，则返回值的数据类型如下：

- 查询两个长度相等的 VARCHAR2 类型值，则返回的值为相同长度的 CHAR 类型。 查询不同长度的 CHAR 类型值，则返回的值为 VARCHAR2 类型，其长度为较大的 CHAR 值。
- 其中一个或两个查询是 VARCHAR2 数据类型的值，则返回的值为 VARCHAR2 类型。

如果组件查询是数值数据，则返回值的数据类型由数值优先级决定：

- 查询 BINARY_DOUBLE 类型值，则返回的值为 BINARY_DOUBLE 类型。
- 查询都选择 BINARY_FLOAT 类型值，则返回的值为 BINARY_FLOAT 类型。
- 所有查询都选择 NUMBER 类型值，则返回的值为 NUMBER 类型。

在使用集合运算符的查询中，OceanBase 不会跨数据类型组执行隐式转换。如果组件查询的相应表达式同时解析为字符数据和数值数据，OceanBase 将返回错误。

集合运算符限制

集合运算符受到以下限制：

- 集合运算符在 BLOB 和 CLOB 的列上无效。
- 如果集合运算符前面的 SELECT 列表包含表达式，则必须为表达式提供列别名，以便在 ORDER BY 子句引用。
- 您不能用设置的运算符指定 UPDATE 语句。
- 您不能在这些运算符的子查询中指定 ORDER BY 语句。
- 不能在包含表集合表达式的 SELECT 语句中使用这些运算符 。

UNION 运算符和 UNION ALL 运算符

UNION 运算符用于合并两个或多个 SELECT 语句的结果集。 UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。默认地， UNION 运算符选取不同的值。如果允许重复的值，请使用 UNION ALL 。

语法

```
{ (< SQL- 查询语句 1>) }
UNION [ALL]
{ (< SQL- 查询语句 2>) }
```

INTERSECT 运算符

返回两个结果集的交集，即两个查询都返回的所有非重复值。

语法

```
{ (< SQL- 查询语句 1>) }
INTERSECT
{ (< SQL- 查询语句 2>) }
```

限制条件

- 所有查询中的列数和列的顺序必须相同。
- 比较的两个查询结果集中的列数据类型可以不同但必须兼容。
- 比较的两个查询结果集中不能包含不可比较的数据类型列（XML、TEXT、NTEXT、IMAGE 或非二进制 CLR 用户定义类型）。
- 返回的结果集的列名与操作数左侧的查询返回的列名相同。`ORDER BY` 子句中的列名或别名必须引用左侧查询返回的列名。
- 不能与 `COMPUTE` 和 `COMPUTE BY` 子句一起使用。
- 通过比较行来确定非重复值时，两个 NULL 值被视为相等。

MINUS 运算符

`MINUS` 运算符返回两个结果集的差，即从左查询中返回右查询没有找到的所有非重复值。

语法

```
{ (< SQL- 查询语句 1>) }
MINUS
{ (< SQL- 查询语句 2>) }
```

执行顺序

与表达式中的其他运算符一起使用时的执行顺序：

1、括号中的表达式。

2、`INTERSECT` 操作数。

3、基于在表达式中的位置从左到右求值的 `MINUS` 和 `UNION`。

如果 `MINUS` 或 `INTERSECT` 用于比较两个以上的查询集，则数据类型转换是通过一次比较两个查询来确定的，并遵循前面提到的表达式求值规则。

示例

以下语句创建了表 table_a 和表 table_b，并向表中插入数据：

```
CREATE TABLE table_a(PK INT, name VARCHAR(25));
INSERT INTO table_a VALUES(1,'福克斯');
INSERT INTO table_a VALUES(2,'警察');
INSERT INTO table_a VALUES(3,'的士');
INSERT INTO table_a VALUES(4,'林肯');
INSERT INTO table_a VALUES(5,'纽约');
INSERT INTO table_a VALUES(6,'华盛顿');
INSERT INTO table_a VALUES(7,'戴尔');
INSERT INTO table_a VALUES(10,'朗讯');
CREATE TABLE table_b(PK INT, name VARCHAR(25));
INSERT INTO table_b VALUES(1,'福克斯');
INSERT INTO table_b VALUES(2,'警察');
INSERT INTO table_b VALUES(3,'的士');
INSERT INTO table_b VALUES(6,'华盛顿');
INSERT INTO table_b VALUES(7,'戴尔');
INSERT INTO table_b VALUES(8,'微软');
INSERT INTO table_b VALUES(9,'苹果');
INSERT INTO table_b VALUES(11,'苏格兰');
```

UNION 示例：

```
SELECT PK, name FROM table_a
UNION
SELECT PK, name FROM table_b;
```

查询结果如下：

```
+------+-----------+
|PK  |NAME    |
+------+-----------+
|  1|福克斯   |
|  2|警察    |
|  3|的士    |
|  4|林肯    |
|  5|纽约    |
|  6|华盛顿   |
|  7|戴尔    |
| 10|朗讯    |
|  8|微软    |
|  9|苹果    |
| 11|苏格兰   |
+------+-----------+
```

UNION ALL 示例：

```
SELECT PK, name FROM table_a
UNION ALL
SELECT PK, name FROM table_b;
```

查询结果如下：

```
+------+-----------+
|PK  |NAME    |
+------+-----------+
|  1|福克斯  |
|  2|警察    |
|  3|的士    |
|  4|林肯    |
|  5|纽约    |
|  6|华盛顿  |
|  7|戴尔    |
| 10|朗讯    |
|  1|福克斯  |
|  2|警察    |
|  3|的士    |
|  6|华盛顿  |
|  7|戴尔    |
|  8|微软    |
|  9|苹果    |
| 11|苏格兰  |
+------+-----------+
```

INTERSECT 示例：

```
SELECT PK, NAME FROM table_a
INTERSECT
SELECT PK, NAME FROM table_b;
```

查询结果如下：

```
+------+-----------+
|PK  |NAME    |
+------+-----------+
|  1|福克斯  |
|  2|警察    |
|  3|的士    |
|  6|华盛顿  |
|  7|戴尔    |
+------+-----------+
```

MINUS 示例：

```
SELECT PK, NAME FROM table_a
MINUS
SELECT PK, NAME FROM table_b;
```

查询结果如下：

```
+------+--------+
|PK  |NAME  |
+------+--------+
|  4|林肯  |
|  5|纽约  |
| 10|朗讯  |
+------+--------+
```

6.9.5. 连接

连接（Join）是将两个或多个表、视图或物化视图的结合在一起的查询。当查询的 FROM 子句中出现多个表时OceanBase 数据库会进行连接查询，查询输出列可以从 FROM 子句任何表中选取。 如果多个表都有一个列名，那么您必须用表名限定查询过程中对这些列的所有引用。数据库中的连接类型一般包括 inner join 、 outer join 、 semi-join 和 anti-join 。其中 Semi-join 和 Anti-join 都是通过子查询改写得到，OceanBase 本身并没有表述 Anti-join 和 Semi-join 的语法。

连接条件

连接条件是将多表结合在一起的规则，存在于从句或 WHERE 子句中，用于比较来自不同的表两列，多数连接至少包含一个连接条件。连接条件可以分为等值连接（比如 t1.a = t2.b）和非等值连接（t1.a < t2.b），相比于非等值连接条件, 等值连接条件允许数据库使用高效的连接算法，比如 Hash Join 和 Merge-Sort join 。

为了执行连接，OceanBase 从不同表取行组合成对，使用连接条件进行匹配。 要执行两表以上的多表连接，OceanBase 首先根据比较它们的列的连接条件连接其中的两个表，然后根据包含连接表和新表列的连接条件将结果连接到另一个表。优化器根据连接条件、基表索引以及可用统计信息确定 OceanBase 连接顺序。

WHERE 子句除了连接条件还可能含有其他条件，这些仅引用一个表的条件可以进一步限制连接查询返回的行数。

等值连接（EQUI-JOINS）

等值连接（Equijoins）是连接条件包含等式运算符的连接。进行等值连接时将特定列满足等值条件的行进行组合输出。

自连接（SELF-JOIN）

自连接是表与其自身的连接。该表在 FROM 子句中出现两次，后跟表别名，这些别名限定联接条件中的列名。执行自连接时OceanBase数据库将组合并返回满足连接条件的行。

笛卡儿积（Cartesian Products）

如果连接查询中的两个表没有连接条件，OceanBase 数据库返回其笛卡尔乘积，使用第一个表的每一行与另一表每一个行进行组合输出。笛卡尔乘积总是生成许多行，很少有用。 例如，两个都有 100 行的表的笛卡尔积有 10,000 行。除非您特别需要笛卡尔乘积，否则始终包括一个连接条件。 如果查询连接了三个或多个表，并且没有为特定对指定连接条件，则优化器可以选择避免生成中间笛卡尔乘积的连接顺序。

内连接（INNER JOIN）

内连接（INNER JOIN）是数据库中最基本的连接操作。内连接基于连接条件将两张表（如 A 和 B）的列组合在一起，产生新的结果表。查询会将 A 表的每一行和 B 表的每一行进行比较，并找出满足连接条件的组合。当连接条件被满足，A 和 B 中匹配的行会按列组合（并排组合）成结果集中的一行。连接产生的结果集等于首先对两张表做笛卡尔积，将 A 中的每一行和 B 中的每一行组合，然后返回满足连接条件的记录。

外连接（OUTER JOIN）

外连接（OUTER JOIN） 返回满足连接条件的所有行，同时从一个表返回没有使用的行，在另一个表相应位置填充 `NULL` 。外连接可依据连接表保留左表、右表或全部表的行而进一步分为左连接、右连接和全连接。其中左连接（LEFT [OUTER] JOIN）中左表行未在右表匹配到行时，在右表自动填充 `NULL` 。右连接（RIGHT [OUTER] JOIN）中右表行未在左表匹配到行时，在左表自动填充 `NULL` 。全连接（FULL [OUTER] JOIN）左表或者右表未在其它表匹配到行时均会填充 `NULL` 。

SEMI 连接（SEMI-JOIN）

当 A 表和 B 表进行 `LEFT` 或 `RIGHT ANTI-JOIN` 的时候，它只返回 A 或 B 中所有能够在 B 或 A 中找到匹配的行。 `SEMI-JOIN` 只能通过子查询展开得到。

ANTI 连接（ANTI-JOIN）

当 A 表和 B 表进行 `LEFT` 或 `RIGHT ANTI-JOIN` 的时候，它只返回 A 或 B 中所有不能在 B 或 A 中找到匹配的行。类似于 `SEMI-JOIN` ， `ANTI-JOIN` 也只能通过子查询展开得到。

示例

建立表 table_a 和表 table_b，并插入数据。执行以下语句：

```
CREATE TABLE table_a(PK INT, name VARCHAR(25));
INSERT INTO table_a VALUES(1,'福克斯');
INSERT INTO table_a VALUES(2,'警察');
INSERT INTO table_a VALUES(3,'的士');
INSERT INTO table_a VALUES(4,'林肯');
INSERT INTO table_a VALUES(5,'亚利桑那州');
INSERT INTO table_a VALUES(6,'华盛顿');
INSERT INTO table_a VALUES(7,'戴尔');
INSERT INTO table_a VALUES(10,'朗讯');
CREATE TABLE table_b(PK INT, name VARCHAR(25));
INSERT INTO table_b VALUES(1,'福克斯');
INSERT INTO table_b VALUES(2,'警察');
INSERT INTO table_b VALUES(3,'的士');
INSERT INTO table_b VALUES(6,'华盛顿');
INSERT INTO table_b VALUES(7,'戴尔');
INSERT INTO table_b VALUES(8,'微软');
INSERT INTO table_b VALUES(9,'苹果');
INSERT INTO table_b VALUES(11,'苏格兰威士忌');
```

自连接查询（SELF-JOIN）：

```
SELECT * FROM table_a ta, table_a tb WHERE ta.NAME = tb.NAME;
```

查询结果如下：

```
+------+----------------+------+----------------+
| PK   | NAME           | PK   | NAME           |
+------+----------------+------+----------------+
|    1 | 福克斯         |    1 | 福克斯         |
|    2 | 警察           |    2 | 警察           |
|    3 | 的士           |    3 | 的士           |
|    4 | 林肯           |    4 | 林肯           |
|    5 | 亚利桑那州     |    5 | 亚利桑那州     |
|    6 | 华盛顿         |    6 | 华盛顿         |
|    7 | 戴尔           |    7 | 戴尔           |
|   10 | 朗讯           |   10 | 朗讯           |
+------+----------------+------+----------------+
```

内连接（INNER JOIN）查询：

```
SELECT A.PK AS A_PK, A.name AS A_Value, B.PK AS B_PK, B.name AS B_Value
FROM table_a A INNER JOIN table_b B ON A.PK = B.PK;
```

查询结果如下：

```
+------+-----------+------+-----------+
|A_PK|A_VALUE  |B_PK|B_VALUE  |
+------+-----------+------+-----------+
|  1|福克斯  |  1|福克斯  |
|  2|警察  |  2|警察  |
|  3|的士  |  3|的士  |
|  6|华盛顿  |  6|华盛顿  |
|  7|戴尔  |  7|戴尔  |
+------+-----------+------+-----------+
```

左连接查询：

```
SELECT  A.PK AS A_PK, A.name AS A_Value, B.PK AS B_PK, B.name AS B_Value
FROM table_a A LEFT JOIN  table_b B ON A.PK = B.PK;
```

查询结果如下：

```
+------+-----------------+------+-----------+
|A_PK|A_VALUE     |B_PK|B_VALUE  |
+------+-----------------+------+-----------+
|  1|福克斯      |  1|福克斯  |
|  2|警察        |  2|警察  |
|  3|的士        |  3|的士  |
|  6|华盛顿      |  6|华盛顿  |
|  7|戴尔        |  7|戴尔  |
|  4|林肯        |NULL|NULL  |
|  5|亚利桑那州    |NULL|NULL  |
| 10|朗讯        |NULL|NULL  |
+------+-----------------+------+-----------+
```

右连接查询：

```
obclient> SELECT  A.PK AS A_PK, A.name AS A_Value, B.PK AS B_PK, B.name AS B_Value FROM table_a A RIGH
T JOIN  table_b B ON A.PK = B.PK;
```

查询结果如下：

```
+------+-----------+------+-------------------+
|A_PK|A_VALUE  |B_PK|B_VALUE      |
+------+-----------+------+-------------------+
|  1|福克斯  |  1|福克斯       |
|  2|警察   |  2|警察        |
|  3|的士   |  3|的士        |
|  6|华盛顿  |  6|华盛顿       |
|  7|戴尔   |  7|戴尔        |
|NULL|NULL   |  8|微软        |
|NULL|NULL   | 11|苏格兰威士忌    |
|NULL|NULL   |  9|苹果        |
+------+-----------+------+-------------------+
```

全连接查询：

```
obclient> SELECT  A.PK AS A_PK,A.name AS A_Value,B.PK AS B_PK,B.name AS B_Value FROM table_a A FULL JOIN table_b B ON A.PK = B.PK;
```

查询结果如下：

```
+------+-----------------+------+-------------------+
|A_PK|A_VALUE     |B_PK|B_VALUE      |
+------+-----------------+------+-------------------+
|  1|福克斯     |  1|福克斯       |
|  2|警察      |  2|警察        |
|  3|的士      |  3|的士        |
|  6|华盛顿     |  6|华盛顿       |
|  7|戴尔      |  7|戴尔        |
|NULL|NULL      |  8|微软        |
|NULL|NULL      |  9|苹果        |
|NULL|NULL      | 11|苏格兰威士忌    |
|  4|林肯      |NULL|NULL        |
|  5|亚利桑那州   |NULL|NULL        |
| 10|朗讯      |NULL|NULL        |
+------+-----------------+------+-------------------+
```

Semi 连接（semi-join）：有依赖关系的子查询被展开改写成 SEMI-JOIN。

```
explain SELECT * FROM table_a t1 WHERE t1.PK IN (SELECT t2.PK FROM table_b t2 WHERE t2.NAME = t1.NAME );
```

查询结果如下：

```
+----------------------------------------------+
| Query Plan                                   |
+----------------------------------------------+
=======================================
|ID|OPERATOR     |NAME|EST. ROWS|COST|
---------------------------------------
|0 |HASH SEMI JOIN|   |8        |114 |
|1 | TABLE SCAN   |T1 |8        |38  |
|2 | TABLE SCAN   |T2 |8        |38  |
=======================================

Outputs & filters:
-------------------------------------
 0 - output([T1.PK], [T1.NAME]), filter(nil),
    equal_conds([T1.PK = T2.PK], [T2.NAME = T1.NAME]), other_conds(nil)
 1 - output([T1.NAME], [T1.PK]), filter(nil),
    access([T1.NAME], [T1.PK]), partitions(p0)
 2 - output([T2.NAME], [T2.PK]), filter(nil),
    access([T2.NAME], [T2.PK]), partitions(p0)
+----------------------------------------------+
```

Anti 连接（anti-join）：有依赖关系的子查询被改写成 `Anti-join` 。

```
EXPLAIN SELECT * FROM table_a t1 WHERE t1.PK NOT IN (SELECT t2.PK
FROM table_b t2 WHERE t2.name = t1.name);
```

查询结果如下：

```
+----------------------------------------------+
| Query Plan                                   |
+----------------------------------------------+
=======================================
|ID|OPERATOR     |NAME|EST. ROWS|COST|
---------------------------------------
|0 |HASH ANTI JOIN|   |0        |112 |
|1 | TABLE SCAN   |T1 |8        |38  |
|2 | TABLE SCAN   |T2 |8        |38  |
=======================================
Outputs & filters:
-------------------------------------
 0 - output([T1.PK], [T1.NAME]), filter(nil),
    equal_conds([T2.NAME = T1.NAME]), other_conds([(T_OP_OR, T1.PK = T2.PK,
    (T_OP_IS, T1.PK, NULL, 0), (T_OP_IS, T2.PK, NULL, 0))])
 1 - output([T1.NAME], [T1.PK]), filter(nil),
    access([T1.NAME], [T1.PK]), partitions(p0)
 2 - output([T2.NAME], [T2.PK]), filter(nil),
    access([T2.NAME], [T2.PK]), partitions(p0)
+----------------------------------------------------------+
```

6.9.6. 子查询

子查询指的是 `SELECT` 查询语句中嵌套了另一个或者多个 `SELECT` 语句，可以返回单行结果、多行结果或不返回结果。 `SELECT` 语句的 `FROM` 子句中的子查询也称为内联视图， `SELECT` 语句的 `WHERE` 子句中的子查询也称为嵌套子查询。

子查询可以分为相关子查询和非相关子查询。相关子查询指该子查询的执行依赖了外部查询的变量，这种子查询通常会执行多次。非相关子查询指该子查询的执行不依赖外部查询的变量，这种子查询一般只需要计算一次。对于非相关子查询与部分相关子查询，可以通过改写进行子查询消除，实现嵌套子查询的展开。

语法

```
SELECT [ hint ] [ { { DISTINCT | UNIQUE } | ALL } ] select_list
FROM { table_reference | join_clause | ( join_clause ) }
 [ , { table_reference | join_clause | (join_clause) } ]
 [ where_clause ]
 [ hierarchical_query_clause ]
 [ group_by_clause ]
| subquery { UNION [ALL] | INTERSECT | MINUS } subquery [ { UNION [ALL] | INTERSECT | MINUS } subquery ]
| ( subquery ) [ order_by_clause ] [ row_limiting_clause ]
```

参数

参数	说明
select_list	查询列表
subquery	子查询
hint	注释
table_reference	要查询的目标表

如果子查询中的列与外层查询中的列具有相同列名，必须在外层查询中在重复列名前加上表名或使用别名。

当上层查询引用到子查询中相关列时，将执行子查询，上层查询可以是 `SELECT`，`UPDATE` 或 `DELETE` 语句，各语句中使用子查询方式：

- 定义要插入到 `INSERT` 或 `CREATE TABLE` 语句的目标表中的行集。
- 在 `CREATE VIEW` 或 `CREATE MATERIALIZED VIEW` 语句中定义要包含在视图或物化视图中的行集。
- 在 `UPDATE` 中定义要分配给现有行的一个或多个值。
- 在 `WHERE` 子句、`HAVING` 子句或 `START WITH` 中提供条件值。
- 定义包含查询操作的表。

嵌套子查询的展开（Unnesting of Nested Subqueries）

嵌套子查询展开是数据库的一种优化策略，它把一些子查询置于外层的父查询中，其实质是把某些子查询转化为等价的多表连接操作。这种策略带来的一个明显的好处是能够有效利用访问路径、连接方法和连接顺序，使查询的层次尽可能的减少。

以下情况数据库将进行嵌套子查询的展开：

- 不相关的 `IN` 子查询。
- `IN` 和 `EXISTS` 中的相关子查询不包含聚合函数或 `GROUP BY` 子句。

可以通过 Hint `UNNEST` 控制是否展开嵌套子查询。

示例

以下语句创建了表 table_a和表 table_b，并向表中插入数据：

```
CREATE TABLE table_a(PK INT, name VARCHAR(25));
INSERT INTO table_a VALUES(1,'福克斯');
INSERT INTO table_a VALUES(2,'警察');
INSERT INTO table_a VALUES(3,'的士');
INSERT INTO table_a VALUES(4,'林肯');
INSERT INTO table_a VALUES(5,'亚利桑那州');
INSERT INTO table_a VALUES(6,'华盛顿');
INSERT INTO table_a VALUES(7,'戴尔');
INSERT INTO table_a VALUES(10,'朗讯');
CREATE TABLE table_b(PK INT, name VARCHAR(25));
INSERT INTO table_b VALUES(1,'福克斯');
INSERT INTO table_b VALUES(2,'警察');
INSERT INTO table_b VALUES(3,'的士');
INSERT INTO table_b VALUES(6,'华盛顿');
INSERT INTO table_b VALUES(7,'戴尔');
INSERT INTO table_b VALUES(8,'微软');
INSERT INTO table_b VALUES(9,'苹果');
INSERT INTO table_b VALUES(11,'苏格兰威士忌');
```

没有依赖关系的子查询，执行以下语句：

```
SELECT * FROM TABLE_A T1 WHERE T1.PK IN (SELECT T2.PK FROM TABLE_B T2);
```

查询结果如下：

```
+------+-----------+
| PK  | NAME    |
+------+-----------+
|  1 | 福克斯  |
|  2 | 警察    |
|  3 | 的士    |
|  6 | 华盛顿  |
|  7 | 戴尔    |
+------+-----------+
```

有依赖关系的子查询，子查询中用到了外层查询变量 **T1.PK**，执行以下语句：

```
SELECT * FROM TABLE_A T1 WHERE T1.PK IN (SELECT T2.PK FROM TABLE_B T2 WHERE T2.PK = T1.PK);
```

查询结果如下：

```
+------+-----------+
| PK  | NAME    |
+------+-----------+
|  1 | 福克斯   |
|  2 | 警察    |
|  3 | 的士    |
|  6 | 华盛顿   |
|  7 | 戴尔    |
+------+-----------+
```

有依赖关系的子查询被展开改写成连接，执行以下语句：

```
EXPLAIN SELECT * FROM TABLE_A T1 WHERE T1.PK IN (SELECT T2.NAME FROM TABLE_B T2 WHERE T2.NAME = T1.NAME);
```

查询结果如下：

```
+------------------------------------+
| Query Plan                 |
+------------------------------------+
=============================================
|ID|OPERATOR        |NAME|EST. ROWS|COST|
---------------------------------------------
|0 |HASH RIGHT SEMI JOIN|   |8      |107 |
|1 | TABLE SCAN      |T2  |8      |38  |
|2 | TABLE SCAN      |T1  |8      |38  |
=============================================
Outputs & filters:
-------------------------------------
 0 - output([T1.PK], [T1.NAME]), filter(nil),
    equal_conds([T1.PK = T2.NAME], [T2.NAME = T1.NAME]), other_conds(nil)
 1 - output([T2.NAME]), filter(nil),
    access([T2.NAME]), partitions(p0)
 2 - output([T1.NAME], [T1.PK]), filter(nil),
    access([T1.NAME], [T1.PK]), partitions(p0)
+------------------------------------+
```

6.10. SQL语句

6.10.1. DDL

6.10.1.1. ALTER KEYSTORE

描述

该语句用于修改 Keystore 属性，包括打开/关闭 Keystore、修改 Keystore 密码、生成 Keystore 密钥。

格式

```
## 打开/关闭 KEYSTORE
ADMINISTER KEY MANAGEMENT SET [keystore_name] OPEN IDENTIFIED BY [password];
ADMINISTER KEY MANAGEMENT SET [keystore_name] CLOSE IDENTIFIED BY [password];

## 设置密钥
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY [password]
```

参数解释

参数	描述
keystore_name	指定 Keystore 名称
password	指定访问控制 Keystore 的密码

示例

- 打开 Keystore，在 keystore 处于 open 状态时，才能访问加密表以及 set key 操作。

```
ADMINISTER KEY MANAGEMENT SET my_keystore OPEN IDENTIFIED BY abcCBAK123;
```

- 更新 Keystore 中存放的主密钥。

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY abcCBAK123;
```

6.10.1.2. ALTER OUTLINE

描述

该语句仅支持使用 SQL_TEXT 创建的 OUTLINE，可用于添加绑定 OUTLINE 和限流规则。

格式

```
ALTER OUTLINE outline_name ADD stmt [ TO target_stmt ]
```

参数解释

参数	描述
outline_name	指定要创建的 OUTLINE 名称。
stmt	一般为一个带有 hint 和原始参数的 DML 语句。
TO target_stmt	如果不指定 TO target_stmt， 则表示如果数据库接受的SQL参数化后与stmt去掉hint参数化文本相同，则将该SQL绑定stmt中hint生成执行计划；如果期望对含有hint的语句进行固定计划，则需要 TO target_stmt 来指明原始的SQL。 **注意** 在使用target_stmt时，严格要求stmt与target_stmt在去掉hint后完全匹配。

示例

- 通过 ALTER OUTLINE 来添加限流规则。

```
OceanBase (root@oceanbase)> alter outline ol_1 add select /*+max_concurrent(1)*/ * from t1 where c1 = 1 a
nd c2 = ?;
OceanBase (root@oceanbase)> alter outline ol_1 add select /*+max_concurrent(1)*/ * from t1 where c1 = ? a
nd c2 = 1;
```

- 通过 ALTER OUTLINE 来添加执行计划。

```
OceanBase (root@oceanbase)> create outline ol_2 on select /*+max_concurrent(1)*/ * from t1,t2 where t1.c
1 = 1;
OceanBase (root@oceanbase)> alter outline ol_2 add select /*+use_nl(t2)*/ * from t1,t2 where t1.c1 = 1;
```

注意事项

- 同一个 outline_name 只能指定一个执行计划。如果通过 create outline 语句指定了执行计划，则无法通过执行 alter outline 时再添加。
- 同 create outline 类似，在 alter outline 时不能同时指定限流规则和执行计划。
- 执行 alter outline 时，需要 outline_name 和 signature 同时匹配。

6.10.1.3. ALTER SEQUENCE

描述

该语句用来修改自增列。

格式

```
ALTER SEQUENCE sequence_name
 [MINVALUE value | NOMINVALUE]
 [MAXVALUE value | NOMAXVALUE]
 [INCREMENT BY value]
 [CACHE value | NOCACHE]
 [ORDER | NOORDER]
 [CYCLE | NOCYCLE];
```

参数解释

参数	描述
MINVALUE value | NOMINVALUE	指定自增列的最小值，取值范围是 -10^{27} ~ $(10^{27}-1)$。 如果指定为 NOMINVALUE，则升序时最小值为 1，降序时最小值为 $-(10^{27}-1)$。 不指定时缺省为 NOMINVALUE。
MAXVALUE value | NOMAXVALUE	指定自增列的最大值，取值范围是 $(-10^{27}+1)$ ~ 10^{27}。 如果指定为 NOMAXVALUE，则升序时最大值为$(10^{28}-1)$，降序时最大值为 -1。 不指定时缺省为 NOMAXVALUE。
START WITH value	指定自增列的起始值。该值必须小于等于 MAXVALUE，且大于等于 MINVALUE。 如果不指定，则升序时默认取值为最小值，降序时默认取值为最大值。
INCREMENT BY value	指定自增列的自增步数。该值不可以为 0。 如果指定为正数，则自增列为升序；如果指定为负数，则自增列为降序。 不指定缺省值为 1。
CACHE value | NOCACHE	指定在内存中预分配的自增值个数，默认为20。
ORDER | NOORDER	指定自增列的值是否按顺序生成，默认为 NOORDER。

参数	描述
CYCLE | NOCYCLE	指定自增列的值是否循环生成，默认为 NOCYCLE。

示例

- 修改自增列 my_sequence 的最大值，并指定自增值循环生成。

```
OceanBase(root@oceanbase)>ALTER SEQUENCE my_sequence MAXVALUE 1024 CYCLE;
```

注意事项

- `ALTER SEQUENCE` 不支持修改 START WITH 值。如果希望修改下一个 SEQUENCE 的起始位置，可以通过修改 `INCREMENT BY`

  实现。
- 其他自增列取值注意事项可参考 CREATE SEQUENCE 语句。

6.10.1.4. ALTER SESSION

描述

该语句用来修改 SESSION 状态。

ALTER SESSION 下面对应了多种功能的语句，目前 OceanBase 支持三种：

- 切换当前 SESSION 连接的数据库
- 修改当前 SESSION 的隔离级别
- 设置 SESSION 变量，支持同一个语句设置多个变量

格式

```
ALTER SESSION SET CURRENT_SCHEMA = current_schema;
ALTER SESSION SET ISOLATION_LEVEL = [READ UNCOMMITTED|READ COMMITTED|REPEATABLE READ|SERIALIZABLE];
ALTER SESSION SET var1_name = var1_value var2_name = var2_value ...;
```

参数解释

参数	描述
CURRENT_SCHEMA	指定 SESSION 连接要切换到的数据库名称
ISOLATION_LEVEL	指定 SESSION 的隔离级别

参数	描述
var1_name = var1_value var2_name = var2_value ...	指定要设置的 SESSION 变量名称和取值。设置多个变量时，变量之间没有逗号分隔。

示例

- 修改 SESSION 的变量值 recyclebin、sql_warnings 和 tx_isolation。

```
OceanBase(TEST@TEST)>alter session set recyclebin = 'on' sql_warnings = 'on' tx_isolation = 'read-committ
ed';
Query OK, 0 rows affected (0.01 sec)
```

6.10.1.5. ALTER TABLE

描述

该语句用来修改已存在的表的结构，比如：修改表及表属性、新增列、修改列及属性、删除列等。

格式

```
alter_table_stmt:
    ALTER TABLE table_name
    alter_table_action_list;
  | RENAME TABLE rename_table_action_list;

alter_table_action_list:
  alter_table_action [, alter_table_action ...]

alter_table_action:
    ADD [COLUMN] {column_definition | (column_definition_list)}
  | MODIFY [COLUMN] column_definition
  | DROP [COLUMN] column_name
  | ADD [CONSTRAINT [constraint_name]] UNIQUE {INDEX | KEY} [index_name] index_desc
     | ADD [CONSTRAINT [constraint_name]] FOREIGN KEY (column_name_list) references_clause
  | ADD [CONSTRAINT [constraint_name]] CHECK (expr)
  | ADD {INDEX | KEY} [index_name] index_desc
  | ADD FULLTEXT [INDEX | KEY] [index_name] fulltext_index_desc
  | ALTER INDEX index_name [VISIBLE | INVISIBLE]
  | DROP {INDEX | KEY} index_name
  | ADD PARTITION (range_partition_list)
  | DROP PARTITION (partition_name_list)
  | REORGANIZE PARTITION name_list INTO partition_range_or_list
```

```
  | REORGANIZE PARTITION name_list INTO partition_range_or_list
  | TRUNCATE PARTITION name_list
  | [SET] table_option_list
  | RENAME [TO] table_name
  | DROP TABLEGROUP
  | DROP CONSTRAINT constraint_name

rename_table_action_list:
  rename_table_action [, rename_table_action ...]

rename_table_action:
  table_name TO table_name

column_definition_list:
  column_definition [, column_definition ...]

column_definition:
  column_name data_type
  [DEFAULT const_value] [AUTO_INCREMENT]
  [NULL | NOT NULL] [[PRIMARY] KEY] [UNIQUE [KEY]] comment

index_desc:
 (column_desc_list) [index_type] [index_option_list]

fulltext_index_desc:
 (column_desc_list) CTXCAT(column_desc_list) [index_option_list]

column_desc_list:
  column_desc [, column_desc ...]

column_desc:
   column_name [(length)] [ASC | DESC]

index_type:
  USING BTREE

index_option_list:
  index_option [ index_option ...]

index_option:
   [GLOBAL | LOCAL]
```

```
  | block_size
  | compression
  | STORING(column_name_list)
  | comment

table_option_list:
  table_option [ table_option ...]

table_option:
  | primary_zone
  | replica_num
  | table_tablegroup
  | block_size
  | compression
  | AUTO_INCREMENT [=] INT_VALUE
  | comment
  | DUPLICATE_SCOPE [=] "none|zone|region|cluster"

partition_option:
    PARTITION BY HASH(expression)
    [subpartition_option] PARTITIONS partition_count
  | PARTITION BY KEY([column_name_list])
    [subpartition_option] PARTITIONS partition_count
  | PARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
    [subpartition_option] (range_partition_list)

subpartition_option:
    SUBPARTITION BY HASH(expression)
    SUBPARTITIONS subpartition_count
  | SUBPARTITION BY KEY(column_name_list)
    SUBPARTITIONS subpartition_count
  | SUBPARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
    (range_subpartition_list)

range_partition_list:
  range_partition [, range_partition ...]

range_partition:
  PARTITION partition_name
  VALUES LESS THAN {(expression_list) | MAXVALUE}
```

```
range_subpartition_list:
  range_subpartition [, range_subpartition ...]

range_subpartition:
  SUBPARTITION subpartition_name
  VALUES LESS THAN {(expression_list) | MAXVALUE}

expression_list:
  expression [, expression ...]

column_name_list:
  column_name [, column_name ...]

partition_name_list:
  partition_name [, partition_name ...]

partition_count | subpartition_count:
  INT_VALUE
```

参数解释

参数	描述
ADD [COLUMN]	增加列，目前不支持增加主键列。
MODIFY [COLUMN]	修改列属性。
DROP [COLUMN]	删除列，不允许删除主键列或者包含索引的列。
ADD [UNIQUE INDEX]	增加唯一索引。
ADD [INDEX]	增加普通索引
ALTER [INDEX]	修改索引属性。
ADD [PARTITION]	增加分区。
DROP [PARTITION]	删除分区。

参数	描述
REORGANIZE [PARTITION]	分区重组。
TRUNCATE [PARTITION]	删除分区数据。
RENAME [TO] table_name	表重命名。
DROP [TABLEGROUP]	删除表组。
DROP [CONSTRAINT]	删除约束。
SET BLOCK_SIZE	设置 Partition 表 BLOCK 大小。
SET REPLICA_NUM	设置表的副本数（指表的副本总数）。
SET COMPRESSION	设置表的压缩方式。
SET USE_BLOOM_FILTER	设置是否使用 BloomFilter。
SET COMMENT	设置注释信息。
SET PROGRESSIVE_MERGE_NUM	设置渐进合并步数，取值范围是1~64。

示例

- 修改表 t2 中字段 d 的字段类型。

```
ALTER TABLE t2 MODIFY d CHAR(10);
```

- 增加、删除列。
  - 增加列前，执行 `DESCRIBE test;` 命令查看表信息，如下图所示：

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

- 执行以下命令增加 c3 列

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

- 增加列后，执行 `DESCRIBE test;` 命令查看表信息，如下图所示：

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

- 执行以下命令删除 c3 列

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

- 删除列后，执行 `DESCRIBE test;` 命令查看表信息，如下图所示：

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

- 设置表格 test 的副本数，并且增加列c5。

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

6.10.1.6. ALTER TABLEGROUP

描述

该语句用来执行以下操作：

- 对一个表组增加多张表。
- 修改表组的分区规则。
- 修改表组的 locality 和 primary zone。

格式

- 对一个表组增加多张表。

```
ALTER TABLEGROUP tablegroupname ADD [TABLE] tblname [, tblname…]
```

- 修改表组的分区规则。

```
ALTER TABLEGROUP tablegroupname alter_tg_partition_option
alter_tg_partition_option:
DROP PARTITION '(' name_list ')'
| ADD PARTITION opt_range_partition_list
| modify_tg_partition_info
```

- 修改表组的 locality 和 primary zone。

```
ALTER TABLEGROUP tablegroupname alter_tablegroup_actions
alter_tablegroup_actions:
alter_tablegroup_action
| alter_tablegroup_action, alter_tablegroup_action
alter_tablegroup_action:
SET LOCALITY [=] locality_name
|SET PRIMARY_ZONE [=] primary_zone_name
```

参数解释

参数	描述
tablegroupname	指定表组。
tblname	表名。对一个表组增加多张表时，表与表之间以逗号（‘，’）分隔。 当添加多个表时，允许表名重复。如果待添加的表已经属于 *tablegroupname* 的表格组，系统不报错。
modify_tg_partition_info	修改表组的分区规则。
LOCALITY locality_name	指定表组的 locality。
PRIMARY_ZONE primary_zone_name	指定表组的主 Zone。

示例

- 创建表组 tgh 和两张关系表 ttgh 和 ttgh2，并将表组的 locality 修改为 F@z1。

```
OceanBase(admin@test)> create tablegroup tgh locality='F,R{ALL_SERVER}@z1' partition by hash partitions
10;
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> create table ttgh(c1 int, c2 int) tablegroup = tgh locality='F,R{ALL_SERVER}@z1';
Query OK, 0 rows affected (0.55 sec)

OceanBase(admin@test)> create table ttgh2(c1 int, c2 int) tablegroup = tgh locality='F,R{ALL_SERVER}@z1';
Query OK, 0 rows affected (0.39 sec)

OceanBase(admin@test)> alter tablegroup tgh set locality ='F@z1';
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> select locality from oceanbase.__all_tablegroup where tablegroup_name ='tgh';
+------------+
| locality   |
+------------+
| FULL{1}@z1 |
+------------+
1 row in set (0.05 sec)

OceanBase(admin@test)> select locality from oceanbase.__all_table where tablegroup_id=(select tablegrou
p_id from oceanbase.__all_tablegroup where tablegroup_name ='tgh');
+------------+
| locality   |
+------------+
| FULL{1}@z1 |
| FULL{1}@z1 |
+------------+
2 rows in set (0.04 sec)
```

6.10.1.7. ALTER USER

描述

该语句主要用于执行以下操作：

- 修改 OceanBase 用户的密码。
- 修改 OceanBase 用户连接的加密方式。

> **说明**
>
> 必须拥有 UPDATE USER 的权限，才可以执行本命令。

格式

- 修改用户密码

```
ALTER USER username IDENTIFIED BY password;
```

- 修改用户连接的加密方式

```
ALTER USER user REQUIRE {NONE | SSL | x509 | tls_option_list}

tls_option_list:
 tls_option
 | tls_option_list tls_option

tls_option:
    CIPHER str_value
  | ISSUER str_value
  | SUBJECT str_value
```

参数解释

参数	描述
REQUIRE	指定加密协议。

示例

- 修改用户 sqluser01 的密码为 abc123

```
ALTER USER sqluser01 IDENTIFIED BY abc123;
```

- 修改用户连接加密协议为 SSL

```
ALTER USER sqluser REQUIRE SSL;
```

6.10.1.8. CREATE INDEX

描述

该语句用来创建索引。索引是创建在表上的，对数据库表中一列或多列的值进行排序的一种结构。其作用主要在于提高查询的速度，降低数据库系统的性能开销。

格式

```
CREATE [UNIQUE] INDEX indexname
            ON tblname (index_col_name,...)
    [index_type] [index_options]
index_type:
    USING BTREE

index_options:
    index_option [index_option…]

index_option:
  GLOBAL | LOCAL
  | COMMENT 'string'
  | COMPRESSION [=] {NONE | LZ4_1.0 | LZO_1.0 | SNAPPY_1.0 | ZLIB_1.0}
  | BLOCK_SIZE [=] size
  | STORING(columname_list)
  | VISIBLE | INVISIBLE

index_col_name:
  colname [(length)] [ASC | DESC]

columname_list:
  colname [, colname…]
```

参数解释

参数	描述
indexname	指定要创建的索引名称。
tblname	指过索引所属的表名。
index_col_name	指定索引的列名，每个列名后都支持ASC（升序），不支持DESC（降序）。默认为升序。 建立索引的排序方式为：首先以index_col_name中第一个列的值排序；该列值相同的记录，按下一列名的值排序；以此类推。

参数	描述
index_type	索引类型，只支持USING BTREE，以B树为索引。
UNIQUE	指定为唯一索引。
index_option	指定索引选项，多个index_option以空格分隔。
GLOBAL | LOCAL	指定该索引是全局索引或局部索引，默认是GLOBAL。
COMMENT	指定注释。
COMPRESSION	指定压缩算法。
BLOCK_SIZE	指定微块大小。
STORING	表示索引表中冗余存储某些列，以提高系统查询性能。

示例

1. 执行以下命令，创建表test。

```
CREATE TABLE test (c1 int primary key, c2 VARCHAR(10));
```

2. 执行以下命令，创建表test的索引。

```
CREATE INDEX test_index ON test (c1, c2 DESC);
```

3. 执行以下命令，查看表test的索引。

```
SHOW INDEX FROM test;
```

6.10.1.9. CREATE KEYSTORE

描述

该语句用来创建 Keystore 对象，此对象用于存储密钥。在透明加密场景中，需要创建 Keystore 对象，每个租户最多只能创建一个 Keystore 对象。

格式

```
ADMINISTER KEY MANAGEMENT CREATE KEYSTORE [KEYSTORE_NAME] IDENTIFIED BY [PASSSWORD] ;
```

参数解释

参数	描述
KEYSTORE_NAME	指定 Keystore 名称
PASSWORD	指定访问控制 Keystore 的密码

示例

创建一个 Keystore 对象。

```
OceanBase(admin@test)>ADMINISTER KEY MANAGEMENT CREATE KEYSTORE my_keystore IDENTIFIED BY abcCBAK123;
```

6.10.1.10. CREATE OUTLINE

描述

该语句用来创建 OUTLINE。可以通过两种方式创建，一种是通过 SQL_TEXT（用户执行的带参数的原始语句），另一种是通过 SQL_ID 创建。

> 注意
>
> 创建 OUTLINE 需要进入对应的用户下执行。

格式

- 使用 SQL_TEXT 创建 OUTLINE

```
CREATE [OR REPLACE] OUTLINE outline_name ON stmt [ TO target_stmt ]
```

- 使用 SQL_ID 创建 OUTLINE

```
CREATE OUTLINE outline_name ON sql_id USING HINT  hint;
```

参数解释

参数	描述
outline_name	指定要创建的 OUTLINE 名称。
OR REPLACE	指定 OR REPLACE 后，如果要创建的 OUTLINE 已存在，则会替换原有的 OUTLINE。
stmt	一般为一个带有 hint 和原始参数的 DML 语句。
TO target_stmt	如果不指定 TO target_stmt， 则表示如果数据库接受的SQL参数化后与stmt去掉hint参数化文本相同，则将该SQL绑定stmt中hint生成执行计划；如果期望对含有hint的语句进行固定计划，则需要 TO target_stmt 来指明原始的SQL。 **注意** 在使用target_stmt时，严格要求stmt与target_stmt在去掉hint后完全匹配。
sql_id	如果sql_id对应的SQL语句已经有hint，则创建OUTLINE指定的hint会覆盖原始语句中所有hint。
hint	格式为 /*+ xxx */ 。

示例

- 使用 SQL_TEXT 创建 OUTLINE

```
CREATE OUTLINE otl_idx_c2
    ON SELECT/*+ index(t1 idx_c2)*/ * FROM t1 WHERE c2 = 1;
```

- 使用 SQL_ID 创建 OUTLINE

```
CREATE OUTLINE otl_idx_c2
ON 'ED570339F2C856BA96008A29EDF04C74'
USING HINT /*+ index(t1 idx_c2)*/;
```

注意事项

使用 SQL_TEXT 方式创建的 OUTLINE 会覆盖 SQL_ID 方式创建的 OUTLINE，SQL_TEXT 方式创建的优先级高。

6.10.1.11. CREATE SEQUENCE

描述

该语句用来创建自增列。

格式

```
CREATE SEQUENCE sequence_name
 [MINVALUE value | NOMINVALUE]
 [MAXVALUE value | NOMAXVALUE]
 [START WITH value]
 [INCREMENT BY value]
 [CACHE value | NOCACHE]
 [ORDER | NOORDER]
 [CYCLE | NOCYCLE];
```

参数解释

参数	描述
MINVALUE value | NOMINVALUE	指定自增列的最小值，取值范围是 -10^{27} ~ $(10^{27}-1)$。 如果指定为 NOMINVALUE，则升序时最小值为 1，降序时最小值为 $-(10^{27}-1)$。 不指定时缺省为 NOMINVALUE。
MAXVALUE value | NOMAXVALUE	指定自增列的最大值，取值范围是 $(-10^{27}+1)$ ~ 10^{27}。 如果指定为 NOMAXVALUE，则升序时最大值为$(10^{28}-1)$，降序时最大值为 -1。 不指定时缺省为 NOMAXVALUE。
START WITH value	指定自增列的起始值。该值必须小于等于 MAXVALUE，且大于等于 MINVALUE。 如果不指定，则升序时默认取值为最小值，降序时默认取值为最大值。
INCREMENT BY value	指定自增列的自增步数。该值不可以为 0。 如果指定为正数，则自增列为升序；如果指定为负数，则自增列为降序。 不指定缺省值为 1。

参数	描述
CACHE value | NOCACHE	指定在内存中预分配的自增值个数，默认为20。
ORDER | NOORDER	指定自增列的值是否按顺序生成，默认为 NOORDER。
CYCLE | NOCYCLE	指定自增列的值是否循环生成，默认为 NOCYCLE。

示例

- 创建一个自增列 s1。

```
OceanBase(ADMIN@TEST)>create sequence S1 minvalue 1 maxvalue 5 nocycle noorder cache 10240000;
```

注意事项

- 同时指定 MINVALUE 和 MAXVALUE 时，MINVALUE 必须小于 MAXVALUE。
- MAXVALUE 和 MINVALUE 之差必须大于等于 INCREMENT BY value。
- CACHE value 的值必须大于 1，如果 CACHE value 的值为 1，则等价于 NOCACHE。
- 指定了 CYCLE，且 INCREMENT BY value 小于 0 时，必须指定 MINVALUE。
- 指定了 CYCLE 时， CACHE 的数量必须不能超过一个 cycle。

6.10.1.12. CREATE SYNONYM

描述

该语句用来创建一个同义词。

格式

```
CREATE [ OR REPLACE ] [ PUBLIC ]
SYNONYM [ schema. ]synonym
FOR [ schema. ]object;
```

参数说明

参数	描述
OR REPLACE	表示如果要创建的同义词名称已存在，则使用新的定义重新创建同义词。

参数	描述
PUBLIC	指定 PUBLIC 来创建公共同义词，所有用户都可以使用。用户必须对基础对象具有相应的权限才能使用该同义词。 在解析对象的引用时，仅当对象没有指定 schema 时，才会使用公共同义词。 如果不指定 PUBLIC，则同义词是私有的，只能当前 schema访问，并且同义词名称在当前 schema 中必须唯一。
[schema.]synonym	schema 指定当前同义词属于哪个 schema，如果指定了 PUBLIC，则对同义词不需要指定 schema；synonym 表示同义词的名称。
[schema.]object	表示同义词对应对象的名称。

示例

- 创建一个同义词。

```
OceanBase(TEST@TEST)>create table t1(c1 int);
Query OK, 0 rows affected (0.11 sec)

OceanBase(TEST@TEST)>create synonym s1 for t1;
Query OK, 0 rows affected (0.04 sec)

OceanBase(TEST@TEST)>insert into s1 values(1);
Query OK, 1 row affected (0.04 sec)

OceanBase(TEST@TEST)>select * from s1;
+------+
| C1  |
+------+
|  1|
+------+
1 row in set (0.02 sec)
```

- 创建一个 PUBLIC 同义词

```
OceanBase(TEST@TEST)>create public synonym syn_pub for t1;
Query OK, 0 rows affected (0.04 sec)

OceanBase(TEST@TEST)>select * from syn_pub;
+------+
| C1   |
+------+
|    1 |
+------+
1 row in set (0.01 sec)
```

注意事项

创建同义词时，需要满足如下权限要求：

- 在当前的 schema 下创建私有的 synonym，需要 CREATE SYNONYM 权限
- 在非当前 schema 下创建私有的 synonym，需要 CREATE ANY SYNONYM 权限
- 创建 PUBLIC 的 synonym，需要 CREATE PUBLIC SYNONYM 权限
- 对于需要创建 synonym 的对象不需要存在，也不需要具有访问权限

```
// 创建synonym_user
OceanBase(TEST@TEST)>CREATE USER synonym_user IDENTIFIED BY synonym_user;
Query OK, 0 rows affected (0.04 sec)

OceanBase(TEST@TEST)>grant CREATE on test.* to synonym_user;
Query OK, 0 rows affected (0.05 sec)

OceanBase(TEST@TEST)>grant SELECT on test.* to synonym_user;
Query OK, 0 rows affected (0.02 sec)

// 连接synonym_user
OceanBase(SYNONYM_USER@TEST)>create or replace synonym s1 for t1;
ERROR-00600: internal error code, arguments: -5036, Access denied; you need (at least one of) the CREATE SY
NONYM privilege(s) for this operation

// 重新给定CREATE SYNONYM权限
OceanBase(TEST@TEST)>grant CREATE SYNONYM on *.* to synonym_user;
Query OK, 0 rows affected (0.03 sec)

OceanBase(SYNONYM_USER@TEST)>create synonym s1 for t1;
Query OK, 0 rows affected (0.07 sec)
```

6.10.1.13. CREATE TABLE

描述

该语句用来在数据库中创建一张新表。

格式

```
CREATE [GLOBAL TEMPORARY] TABLE table_name
    (table_definition_list) [table_option_list] [partition_option] [on_commit_option]
CREATE [GLOBAL TEMPORARY] TABLE table_name
    (table_definition_list) [table_option_list] [partition_option] [AS] select;

table_definition_list:
  table_definition [, table_definition ...]

table_definition:
    column_definition
  | INDEX [index_name] index_desc
  | [CONSTRAINT [constraint_name]] [PRIMARY KEY|UNIQUE]  (column_desc_list) [USING INDEX index_option
_list]
  | [CONSTRAINT [constraint_name]] FOREIGN KEY (column_name, column_name ...) references_clause cons
tranit_state
  | [CONSTRAINT [constraint_name]] CHECK(expression) constranit_state


column_definition_list:
  column_definition [, column_definition ...]

column_definition:
  column_name data_type
  [VISIBLE|INVISIBLE]
  {
  [DEFAULT expression]
  [NULL | NOT NULL]
  [CONSTRAINT [constraint_name] [PRIMARY] KEY] [UNIQUE [KEY]]
  [CONSTRAINT [constraint_name] CHECK(expression) constranit_state]
  [CONSTRAINT [constraint_name] references_clause
  |
  [GENERATED ALWAYS] AS (expression) [VIRTUAL]
  [NULL | NOT NULL] [UNIQUE KEY] [[PRIMARY] KEY] [UNIQUE LOWER_KEY] [COMMENT string]
  }
```

```
references_clause:
   REFERENCES table_name (column_name, column_name ...) [ON DELETE {CASCADE|SET NULL}]]

constranit_state:
  [RELY|NORELY] [USING INDEX index_option_list] [ENABLE|DISABLE] [VALIDATE|NOVALIDATE]

index_desc:
 (column_desc_list) [index_option_list]

column_desc_list:
  column_desc [, column_desc ...]

column_desc:
   column_name [ASC | DESC][NULL LAST|NULL FIRST]

index_option_list:
  index_option [ index_option ...]

index_option:
    [GLOBAL | LOCAL]
  | block_size
  | compression
  | STORING(column_name_list)
  | comment

table_option_list:
  table_option [ table_option ...]

table_option:
  primary_zone
  | replica_num
  | table_tablegroup
  | block_size
  | compression
  | comment
  | DUPLICATE_SCOPE [=] "none|zone|region|cluster"
  | LOCALITY [=] "locality description"
  | ENABLE ROW MOVEMENT
  | DISABLE ROW MOVEMENT
  | physical_attribute
```

```
physical_attribute_list:
  physical_attribute [physical_attribute]

physical_attribute:
  PCTFREE [=] num
  | PCTUSED num
  | INITRANS num
  | MAXTRANS num
  | STORAGE(storage_option [storage_option] ...)
  | TABLESPACE tablespace


compression:
  NOCOMPRESS
  | COMPRESS { BASIC | FOR OLTP | FOR QUERY [LOW|HIGH]  | FOR ARCHIVE [LOW|HIGH]}

storage_option:
  INITIAL_ num [K|M|G|T|P|E]
  | NEXT num [K|M|G|T|P|E]
  | MINEXTENTS num [K|M|G|T|P|E]
  | MAXEXTENTS num [K|M|G|T|P|E]

partition_option:
    PARTITION BY HASH(column_name_list)
    [subpartition_option] PARTITIONS partition_count [TABLESPACE tablespace] [compression]
  | PARTITION BY RANGE (column_name_list)
    [subpartition_option] (range_partition_list)
  | PARTITION BY LIST (column_name_list)
    [subpartition_option] (list_partition_list)

subpartition_option:
    SUBPARTITION BY HASH (column_name_list) SUBPARTITIONS subpartition_count
  | SUBPARTITION BY RANGE (column_name_list) SUBPARTITION TEMPLATE
    (range_subpartition_list)
  | SUBPARTITION BY LIST (column_name_list) SUBPARTITION TEMPLATE (list_subpartition_list)

range_partition_list:
  range_partition [, range_partition ...]

range_partition:
```

```
range_partition:
  PARTITION [partition_name]
  VALUES LESS THAN {(expression_list) | MAXVALUE} [ID = num] [physical_attribute_list] [compression]

range_subpartition_list:
  range_subpartition [, range_subpartition ...]

range_subpartition:
  SUBPARTITION subpartition_name
  VALUES LESS THAN {(expression_list) | MAXVALUE} [physical_attribute_list]

list_partition_list:
  list_partition [, list_partition] ...

list_partition:
  PARTITION [partition_name] VALUES (DEFAULT|expression_list) [ID num] [physical_attribute_list] [compression]

list_subpartition_list:
  list_subpartition [, list_subpartition] ...

list_subpartition:
  SUBPARTITION [partition_name] VALUES (DEFAULT|expression_list) [physical_attribute_list]

expression_list:
  expression [, expression ...]

column_name_list:
  column_name [, column_name ...]

partition_name_list:
  partition_name [, partition_name ...]

partition_count | subpartition_count:
  INT_VALUE

on_commit_option:
  ON COMMIT DELETE ROWS
  | ON COMMIT PRESERVE ROWS
```

参数说明

参数	描述
DUPLICATE_SCOPE	用来指定复制表属性，取值如下： • none：表示该表是一个普通表 • zone：表示该表是一个复制表，leader需要将事务复制到本zone的所有F副本及R副本 • region：表示该表是一个复制表，leader需要将事务复制到本region的所有F副本及R副本 • cluster：表示该表是一个复制表，leader需要将事务复掉到cluster的所有F副本及R副本 不指定DUPLICATE_SCOPE的情况下，默认值为none。
BLOCK_SIZE	指定表的微块大小
COMPRESSION	指定存储格式flat/encoding以及压缩方法，对应如下： • nocompress: flat格式, none 压缩 • compress [basic]: flat格式, lz4_1.0 压缩 • compress for oltp: flat格式, zstd_1.0 压缩 • query [low|high]: encoding格式, lz4_1.0压缩 • archive [low|high]: encoding格式, zstd_1.0压缩
primary_zone	指定主Zone（副本Leader所在 Zone）。
replica_num	指定副本数。
table_tablegroup	指定表所属的talegroup。
comment	注释。
LOCALITY	描述副本在Zone间的分布情况，如：F@z1,F@z2,F@z3,R@z4 表示z1, z2, z3为全功能副本，z4为只读副本。
physical_attribute	PCTFREE: 指定宏块保留空间百分比 其它属性：STORAGE, TABLESPACE 等仅为了语法兼容方便迁移，不生效

参数	描述
ENABLE/DISABLE ROW MOVEMENT	是否允许更会致行在不同分区间移动的分区键更新
ON COMMIT DELETE ROWS	事务级临时表：提交时删除数据
ON COMMIT PRESERVE ROWS	会话级临时表：会话结束时删除数据

示例

- 创建数据库表。

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

- 创建一个复制表。

```
CREATE TABLE item() locality = 'F,R{all_server}@hz1, F,R{all_server}@hz2,
F,R{all_server}@hz3' DUPLICATE_SCOPE="cluster"
```

- 创建带索引的表。

```
create table t1 (c1 int primary key, c2 int, c3 int, index i1 (c2));
```

- 创建 hash 分区，分区数为 8 的表。

```
create table t1 (c1 int primary key, c2 int) partition by hash(c1) partitions 8;
```

- 创建一级分区为 range 分区，二级分区为 hash 分区的表。

```
create table t1 (c1 int, c2 int, c3 int)
  partition by range(c1) subpartition by hash(c2) subpartitions 5
  (partition p0 values less than(0), partition p1 values less than(100));
```

- 开启 encoding 并使用 zstd 压缩，宏块保留空间为 5%。

```
create table t1 (c1 int, c2 int, c3 varchar(64))
 COMPRESS FOR ARCHIVE
 PCTFREE 5;
```

- 创建一个事务级临时表。

```
create global temporary table t1 (c1 int) on commit delete rows ;
```

- 创建一个带约束的表。

```
create table t1 (c1 int, c2 int, c3 int, CONSTRAINT equal_check CHECK(c2 = c3 * 2) ENABLE VALIDATE);
```

6.10.1.14. CREATE TABLEGROUP

描述

该语句用来创建表组。

> **说明**
>
> 只有租户下的管理员权限才可以创建表组。

格式

```
CREATE TABLEGROUP [IF NOT EXISTS] tablegroupname [opt_tablegroup_option_list] [opt_tg_partition_option]

opt_tablegroup_option_list:
tablegroup_option [tablegroup_option]

tablegroup_option:
LOCALITY [=] locality_name
| PRIMARY_ZONE [=] primary_zone_name

opt_tg_partition_option:
PARTITION BY
 HASH COLUMN_NUM [tg_subpartition_option] PARTITIONS INTNUM
| RANGE COLUMNS COLUMN_NUM [tg_subpartition_option] {PARTITION partition_name VALUES LESS THAN range_partition_expr, ...}
| LIST COLUMNS COLUMN_NUM [tg_subpartition_option] {PARTITION partition_name VALUES in list_partition_expr, ...}

tg_subpartition_option:
SUBPARTITION BY
RANGE COLUMN_NUM SUBPARTITION TEMPLATE {SUBPARTITION partition_name VALUES LESS THAN range_partition_expr, ...}
| HASH COLUMN_NUM [SUBPARTITIONS INTNUM]
| LIST COLUMN_NUM SUBPARTITION TEMPLATE {SUBPARTITION partition_name VALUES in list_partition_expr, ...}
```

参数解释

参数	描述
tablegroupname	表组名称，最长 64 个字符，字符只能有大小写英文字母，数字和下划线，而且必须以字母或下划线开头，并且不能使用 OceanBase 的关键字。 如果要创建的表组名称已存在，并且没有指定 IF NOT EXISTS，则会出现错误。
opt_tablegroup_option_list	表组的分区方式、locality、primary zone 必须和组内的表完全一致。 组内的表信息不能独立变更某一项，只可以通过操作表组进行批量操作。 相同的 locality：副本类型、个数、位置要完全一致 相同的 primary zone：leader 位置及其优先级要完全一致 相同的分区方式： • 分区类型相同（例如，都是 hash+range 分区）。 • 如果是 hash 分区，要求引用的列数相同，且分区个数相同。 • 如果是 range 分区，要求引用的列数相同，且分区数相同，且 range 分割点相同。 • 对于二级分区，根据分区类型，要求和上述一致
opt_tg_partition_option	表组的分区规则定义，与 CREATE TABLE 使用同样的分区方式。 表组没有具体的列定义，所以 HASH、RANGE、LIST 不需要写出具体的列，只需要指定列个数（COLUMN_NUM）。

示例

- 创建名为 myTableGroup1 的表组。

```
OceanBase(admin@test)> CREATE TABLEGROUP myTableGroup1;
Query OK, 0 rows affected (0.07 sec)

OceanBase(admin@test)> create table myt1 (c1 int, c2 int ) tablegroup = myTableGroup1;
Query OK, 0 rows affected (0.28 sec)

OceanBase(admin@test)> create table myt2 (c1 int, c2 int ) tablegroup = myTableGroup1;
Query OK, 0 rows affected (0.26 sec)
```

- 创建 hash 分区的表组 tgh，同时创建 hash 分区的表 ttgh，且分区个数相同。

```
OceanBase(admin@test)> create tablegroup tgh locality='F,R{ALL_SERVER}@z1' partition by hash partitions 10;
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> create table ttgh(c1 int, c2 int) locality='F,R{ALL_SERVER}@z1' partition by hash(c1) partitions 10;
Query OK, 0 rows affected (0.55 sec)

OceanBase(admin@test)> create table ttgh2(c1 int, c2 int) locality='F,R{ALL_SERVER}@z1' partition by hash(c2) partitions 10;
Query OK, 0 rows affected (0.39 sec)
```

6.10.1.15. CREATE TABLESPACE

描述

该语句用来创建 TableSpace 逻辑对象，TableSpace 属性支持加密。

格式

```
CREATE TABLESPACE tablespace_name [ENCRYPTION USING 'AES-256|AES-128|AES-192|SM4']
```

参数解释

参数	描述
tablespace_name	指定要创建的 TableSpace 对象名称
ENCRYPTION USING	使用加密的关键字
AES-256|AES-128|AES-192|SM4	分别代表不同的加密算法

示例

- 创建 TableSpace 对象。

```
CREATE TABLESPACE ob_tablespace;

CREATE TABLESPACE ob_tablespace ENCRYPTION USING 'AES-256';
```

6.10.1.16. CREATE USER

描述

该语句用于创建新的 OceanBase 用户。创建新用户后，可以使用该用户连接 OceanBase。

> 说明
>
> 必须拥有 CREATE USER 的系统权限，才可以使用 CREATE USER 命令。

格式

```
create_user_stmt:
  CREATE USER user_name [host_name] IDENTIFIED BY password [REQUIRE {NONE | SSL | X509 | tls_option_list}]
 [PROFILE user_profile] [DEFAULT TABLESPACE table_space]

password:
  STR_VALUE

tls_option_list:
    tls_option
  | tls_option_list, tls_option

tls_option:
      CIPHER STR_VALUE
  | ISSUER STR_VALUE
  | SUBJECT STR_VALUE
```

参数解释

参数	描述
user_name	用户名，新建用户后，“dba_users”表会新增一行该用户的表项。如果同名用户已经存在，则报错。
host_name	用户所在的主机名，为 @xxx.xxx.xx.x格式。
IDENTIFIED BY	必须使用 IDENTIFIED BY 子句，可以为账户指定一个密码。

参数	描述
REQUIRE	指定用户使用的加密协议为 NONE, SSL, X509 和 tls_option_list 中的一种。
PROFILE user_profile	指定用户使用的 Profile。
DEFAULT TABLESPACE table_space	指定用户的默认表空间。

示例

1. 执行以下命令创建名为 sqluser 的用户，密码为 123456。

```
CREATE USER sqluser IDENTIFIED BY 123456;
```

2. 执行以下命令查看创建的用户。

```
SELECT username FROM dba_users;
```

执行结果如下所示。

```
OceanBase(TEST@TEST)>CREATE USER sqluser IDENTIFIED BY 123456;
Query OK, 0 rows affected (0.05 sec)
OceanBase(TEST@TEST)>SELECT username FROM dba_users;
+------------+
| USERNAME   |
+------------+
| SYS        |
| LBACSYS    |
| ORAAUDITOR |
| ROOT       |
| TEST       |
| ADMIN      |
| SQLUSER    |
+------------+
7 rows in set (0.01 sec)
```

6.10.1.17. CREATE VIEW

描述

该语句用于创建视图语句，如果指定了 OR REPLACE 子句，该语句能够替换已有的视图。

视图在数据库中实际上并不是以表的形式存在，每次使用时他们就会派生。视图是通过 CREATE VIEW 语句中指定的 SELECT 语句的结果生成的。

支持可更新视图。

格式

```
create_view_stmt:
 CREATE [OR REPLACE] VIEW view_name [(column_name_list)] AS select_stmt;

column_name_list:
  column_name [, column_name ...]
```

参数解释

参数	描述
OR REPLACE	表示如果要创建的视图名称已存在，则使用新的定义重新创建视图。
view_name	视图名。
select_stmt	是一种 SELECT 语句。它给出了视图的定义。该语句可以从基表或其他视图进行选择。
column_name_list	视图必须具有唯一的列名，不得有重复，就像基表那样。默认情况下，由 SELECT 语句检索的列名将用作视图列名。 要想为视图列定义明确的名称，可使用可选的 column_name_list 子句，列出由逗号隔开的 ID。column_name_list 中的名称数目必须等于SELECT语句检索的列数。 SELECT 语句检索的列可以是对表列的简单引用。也可以是使用函数、常量值、操作符等的表达式。

示例

- 选择表 t 中的 c1、c2 列来创建视图 v。

```
create or replace view v(vc1, vc2) as select c1, c2 from t;
```

6.10.1.18. DROP INDEX

描述

该语句用来删除索引。当索引过多时，维护开销增大，因此，需要删除不必要的索引。

删除索引的时候需要等待一段时间才能完全删除。

格式

```
DROP INDEX [schema.]indexname;
```

参数解释

参数	描述
schema	指定 schema 名称
indexname	指定索引名称

示例

- 删除索引 test_index。

```
DROP INDEX test_index;
```

6.10.1.19. DROP OUTLINE

描述

该语句用于删除 OceanBase 数据库中的 OUTLINE。

格式

```
DROP OUTLINE outline_name;
```

参数解释

参数	描述
outline_name	指定要删除的 OUTLINE 名称。

示例

- 删除 OUTLINE ol_1

```
DROP OUTLINE ol_1;
```

6.10.1.20. DROP SEQUENCE

描述

该语句用来删除自增列。

格式

```
DROP SEQUENCE sequence_name
```

参数解释

参数	描述
sequence_name	指定要删除的自增列名称

示例

- 删除自增列 S1

```
OceanBase(TEST@TEST)>drop sequence S1;
Query OK, 0 rows affected (0.16 sec)
```

6.10.1.21. DROP SYNONYM

描述

该语句用来删除一个同义词。

格式

```
DROP [PUBLIC] SYNONYM [ schema. ]synonym;
```

参数说明

参数	描述
PUBLIC	指定 PUBLIC 来删除公共同义词；如果不指定 PUBLIC，则删除私有同义词。
[schema.]synonym	schema 指定当前同义词属于哪个 schema，如果指定了 PUBLIC，则对同义词不需要指定 schema；synonym 表示同义词的名称。

举例

- 删除一个同义词。

```
OceanBase(TEST@TEST)>drop synonym test.s1;
Query OK, 0 rows affected (0.04 sec)
```

- 删除一个 PUBLIC 同义词。

```
OceanBase(TEST@TEST)>drop public synonym syn_pub;
Query OK, 0 rows affected (0.03 sec)
```

注意事项

1. 删除同义词时，需要满足如下权限：
   - 删除私有同义词时：
     - 要删除的 synonym 在对应 schema 下
     - 要有 DROP ANY SYNONYM 权限
   - 删除公共同义词时，需要有 DROP PUBLIC SYNONYM 权限
2. 删除 public synonym 时，必须指定 PUBLIC 关键字，且不能指定 schema。

6.10.1.22. DROP TABLE

描述

该语句用于删除 OceanBase 数据库中的表。

格式

```
DROP TABLE table_name [CASCADE CONSTRAINTS] [PURGE]
```

参数解释

参数	描述
table_name	指定要删除的表名
CASCADE CONSTRAINTS	级联删除跟 table_name 关联的约束
PURGE	彻底删除表 (不进回收站)

示例

- 删除表 test

```
DROP TABLE test;
```

6.10.1.23. DROP TABLEGROUP

描述

该语句用来删除表组。

格式

```
DROP TABLEGROUP [IF EXISTS] tablegroupname
```

参数解释

参数	描述
tablegroupname	表组名称。如果要删除的表组名称不存在，并且没有指定 IF EXISTS，则会出现错误。

示例

- 删除名为 myTableGroup1 的表组。

```
OceanBase(admin@test)> DROP TABLEGROUP myTableGroup1;
```

6.10.1.24. DROP TABLESPACE

描述

该语句用来删除 TableSpace 逻辑对象，仅在 TableSpace下没有表时可以删除成功。

格式

```
DROP TABLESPACE tablespace_name;
```

参数解释

参数	描述
tablespace_name	指定要删除的 TableSpace 名称

示例

- 删除 TableSpace 对象 ts

```
OceanBase(root@oceanbase)>drop tablesapce ts;
```

6.10.1.25. DROP USER

描述

该语句用于删除一个 OceanBase 用户。

> 说明
>
> - 必须拥有 CREATE USER 的系统权限，才可以使用 DROP USER 命令。
> - CASCADE 会指定在删除用户前，删除该用户拥有的所有 Object，包括权限，数据库，表等。

格式

```
DROP USER username CASCADE;
```

参数解释

参数	描述
username	用户名，只支持删除单个用户。

示例

- 执行以下命令，删除名为 sqluser 的用户。

```
oceanBase(admin@TEST)>DROP USER sqluser CASCADE;
Query OK, 0 rows affected (0.06 sec)
```

6.10.1.26. DROP VIEW

描述

该语句用于删除一个或多个视图。

> 说明
>
> 当前用户必须在每个视图上有 DROP 权限。

格式

```
drop_view_stmt:
  DROP VIEW view_name [CASCADE | RESTRICT];
```

参数解释

参数	描述
view_name	删除的视图名
CASCADE | RESTRICT	CASCADE 和 RESTRICT 会被解析和忽略。

示例

- 删除视图 v1

```
OceanBase(root@oceanbase)>drop view v1;
```

6.10.1.27. RENAME

描述

该语句用来对对象进行重命名。

格式

```
RENAME obj_name TO new_obj_name;
```

参数解释

参数	描述
obj_name	原对象名。
new_obj_name	新对象名。

示例

1. 创建表 t1 和 t2。

```
create table t1(c1 int);
```

2. 将表 t1 重命名为 t11。

```
rename t1 to t11;
```

注意事项

重命名操作可以对表、视图、私有同义词、sequence 进行。

6.10.1.28. TRUNCATE TABLE

描述

该语句用于完全清空指定表，但是保留表结构，包括表中定义的 Partition 信息。从逻辑上说，该语句与用于删除所有行的DELETE FROM语句相同。执行TRUNCATE语句，必须具有表的删除和创建权限。

TRUNCATE TABLE 语句与 DELETE FROM 语句有以下不同：

- 删减操作会取消并重新创建表，这比一行一行的删除行要快很多。
- TRUNCATE TABLE 语句执行结果显示影响行数始终显示为 0 行。

格式

```
TRUNCATE [TABLE] table_name;
```

参数解释

参数	描述
table_name	指定表名。

示例

- 完全清空表 tb1。

```
OceanBase(root@oceanbase)>TRUNCATE TABLE tb1;
```

6.10.2. DML

6.10.2.1. DELETE

描述

该语句用来删除表中符合条件的行，包括单表删除及多表删除两种方式。

格式

```
DELETE [hint_options] [FROM] table_factor
    [WHERE where_condition]
    [{ RETURNING | RETURN } returning_exprs [into_clause]]

table_factor:
  {tbl_name | table_subquery | '(' table_reference ')' }

where_condition:
  expression

returning_exprs:
  projection [, ...]

into_clause:
{ INTO into_var_list | BULK COLLECT INTO into_var_list}

into_var_list:
{ USER_VARIABLE | ref_name } [, ...]
```

参数解释

参数	描述
hint_options	指定 hint 选项。
table_factor	指定需要删除的表名（基表、可更新视图、特殊子查询）。
where_condition	删除的表需要满足的过滤条件。
returning_exprs	返回删除数据前的投影列。
into_clause	将删除数据前的投影列插入到指定列表。

注意

特殊子查询指的类似于可更新视图对应的子查询，这类子查询不应该包含复杂的算子（比如group by/distinct/window function等）

示例

示例表及数据基于以下定义：

```
OceanBase(admin@test)>create table t1(c1 int primary key, c2 int);
Query OK, 0 rows affected (0.16 sec)
OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.06 sec)
```

- 单表删除：删除 “c1=2” 的行，其中 c1 列为表 t1 中的 Primary Key。

```
OceanBase(admin@test)>DELETE FROM t1 WHERE c1 = 2;
Query OK, 1 row affected (0.02 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  3 |    3 |
|  4 |    4 |
+----+------+
3 rows in set (0.01 sec)
```

- 单表删除：直接操作子查询进去删除操作。

```
OceanBase(admin@test)>DELETE FROM (SELECT * FROM t1);
Query OK, 4 rows affected (0.04 sec)

OceanBase(admin@test)>select * from t1;
Empty set (0.01 sec)
```

- 单表删除：包含returning 子句。

```
OceanBase(admin@test)>DELETE FROM t1 RETURNING c1;
+----+
| C1 |
+----+
| 1 |
| 2 |
| 3 |
| 4 |
+----+
4 rows in set (0.01 sec)

OceanBase(admin@test)>select * from t1;
Empty set (0.01 sec)
```

6.10.2.2. INSERT

描述

该语句用于添加一个或多个记录到表中。

格式

```
INSERT [hint_options] single_table_insert

single_table_insert:
{ INTO insert_table_clause opt_nologging '(' column_list ')' values_clause [{ RETURNING | RETURN } returning
_exprs [into_clause]]
| INTO insert_table_clause opt_nologging '(' ')' values_clause [{ RETURNING | RETURN } returning_exprs [into
_clause]]
| INTO insert_table_clause opt_nologging values_clause [{ RETURNING | RETURN } returning_exprs [into_clau
se]]
}

opt_nologging: { NOLOGGING | /*EMPTY*/ }

returning_exprs:
  projection [, ...]

into_clause:
{ INTO into_var_list | BULK COLLECT INTO into_var_list}

into_var_list:
{ USER_VARIABLE | ref_name } [, ...]
```

参数解释

参数	描述
hint_options	指定 hint 选项。
insert_table_clause	指定的插入的表，可以是基表、可更新视图、特殊子查询。
opt_nologging	尽量减少插入时的日志信息
column_list	指定要插入的列名。
returning_exprs	返回插入数据之后的投影列。
into_clause	将插入数据之后的列值插入到指定列表中。

> **注意**
>
> 特殊子查询指的类似于可更新视图对应的子查询，这类子查询不应该包含复杂的算子（比如group by/distinct/window function等）

示例

示例表及数据基于以下定义：

```
OceanBase(admin@test)>create table t1(c1 int primary key, c2 int);
Query OK, 0 rows affected (0.16 sec)
OceanBase(admin@test)>select * from t1;
Empty set (0.02 sec)
```

- 单表插入：向表 t1 中插入一行数数据。

```
OceanBase(admin@test)>insert into t1 values(1,1);
Query OK, 1 row affected (0.01 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.04 sec)
```

- 单表插入：直接向子查询中插入数据

```
OceanBase(admin@test)>insert into (select * from t1) values(1,1);
Query OK, 1 row affected (0.01 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| C1 | C2   |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.01 sec)
```

- 单表插入：包含returning子句。

```
OceanBase(admin@test)>insert into t1 values(1,1) returning c1;
+----+
| C1 |
+----+
| 1 |
+----+
1 row in set (0.02 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| C1 | C2  |
+----+------+
| 1 |  1 |
+----+------+
1 row in set (0.01 sec)
```

6.10.2.3. MERGE

描述

该语句用来根据一张源表更新一张目标表的数据，可以对目标表进行插入、更新或者删除。

格式

```
MERGE [hint_options]
      INTO table_factor [opt_alias]
     USING table_factor [opt_alias]
       ON '(' expr ')'
  [merge_update_clause]
    [merge_insert_clause]

merge_update_clause:
  WHEN MATCHED THEN UPDATE SET update_asgn_list [WHERE expr] [DELETE WHERE expr]

merge_insert_clause:
  WHEN NOT MATCHED THEN INSERT opt_insert_columns VALUES '(' insert_vals ')' [WHERE expr]
```

参数解释

参数	描述
hint_options	指定 hint 选项
table_factor	指定源表和目标表的表名
ON expr	源表和目标表的连接条件
update_asgn_list	更新语句的赋值操作
WHERE expr	触发更新、删除或者插入操作时的需要满足的条件

示例

示例表和数据基于以下定义：

```
create table t1 (c1 int, c2 int);
create table t2 (c1 int, c2 int);

insert into t1 values (0, 0);
insert into t1 values (1, null);
insert into t1 values (2, null);
insert into t2 values (1, 1);
insert into t2 values (2, 20);
insert into t2 values (3, 3);
insert into t2 values (4, 40);
```

根据 T2 的数据更新 T1 的数据，

1. 如果 t1.c1 在 t2 中能够找到 c1 取值相同的行
    i. 如果 t1.c2 取值为空，那么使用 t2.c2 更新 t1.c2
    ii. 如果更新之后，t1.c2 >= 0，那么删除这行
2. 如果 t2.c1 在 t1 中找不到取值相同的行
    i. 将 t2 中的这行插入到 t1 中，且只插入满足 t2.c2 < 10 的行

```
merge into t1 using t2 on (t1.c1 = t2.c1)
when matched then update set c2 = t2.c2 where t1.c2 is null delete where t1.c2 >= 10
when not matched then insert values (t2.c1, t2.c2) where t2.c2 < 10;
Query OK, 3 rows affected (0.02 sec)

select * from t1;
+------+------+
|C1  |C2  |
+------+------+
|  0|  0|
|  1|  1|
|  3|  3|
+------+------+
```

6.10.2.4. PURGE DATABASE

描述

该语句用于删除放在回收站的数据库。

格式

```
PURGE DATABASE object_name;
```

参数解释

参数	描述
object_name	对象进入回收站后的新名字

示例

- 删除放在回收站的数据库 __recycle_$_1_1597384386029184。

```
OceanBase(admin@test)> create database db1;
Query OK, 1 row affected (0.03 sec)

OceanBase(admin@test)> drop database db1;
Query OK, 0 rows affected (0.04 sec)

OceanBase(admin@test)> show recyclebin;
+--------------------------------+---------------+----------+----------------------------+
| OBJECT_NAME                    | ORIGINAL_NAME | TYPE     | CREATETIME                 |
+--------------------------------+---------------+----------+----------------------------+
| __recycle_$_1_1597384386029184 | db1           | DATABASE | 2020-08-14 13:53:06.029367 |
+--------------------------------+---------------+----------+----------------------------+
1 row in set (0.01 sec)

OceanBase(admin@test)> purge database __recycle_$_1_1597384386029184;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)> show recyclebin;
```

6.10.2.5. PURGE INDEX

描述

该语句用来删除放在回收站中的索引表。

格式

```
PURGE INDEX object_name;
```

参数解释

参数	描述
object_name	对象进入回收站后的新名字

示例

- 删除放在回收站中的索引表 __recycle_$_1_1597387726700872。

```
OceanBase(admin@test)> create table t1(c1 int);
Query OK, 0 rows affected (0.09 sec)

OceanBase(admin@test)> create index idx on t1(c1);
Query OK, 0 rows affected (0.48 sec)

OceanBase(admin@test)> drop table t1;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)> show recyclebin;
+-------------------------------+---------------------------+-------+---------------------------+
| OBJECT_NAME                   | ORIGINAL_NAME             | TYPE  | CREATETIME                |
+-------------------------------+---------------------------+-------+---------------------------+
| __recycle_$_1_1597387726700872 | __idx_1101710651081557_idx | INDEX | 2020-08-14 14:48:46.699145 |
| __recycle_$_1_1597387726712976 | t1                        | TABLE | 2020-08-14 14:48:46.712643 |
+-------------------------------+---------------------------+-------+---------------------------+
5 rows in set (0.01 sec)

OceanBase(admin@test)> purge index __recycle_$_1_1597387726700872;
Query OK, 0 rows affected (0.04 sec)
```

6.10.2.6. PURGE RECYCLEBIN

描述

该语句用于 root 用户清空回收站。

> **注意**
>
> PURGE RECYCLEBIN会清除回收站中的所有东西，只有 root 用户才能够执行这条语句，操作时请谨慎。

格式

```
PURGE RECYCLEBIN;
```

参数解释

无。

示例

- root 用户清空回收站。

```
OceanBase(admin@test)> purge recyclebin;
Query OK, 0 rows affected (0.03 sec)
```

6.10.2.7. PURGE TABLE

描述

该语句用于删除放在回收站的表。

格式

```
PURGE TABLE object_name;
```

参数解释

参数	描述
object_name	对象进入回收站后的新名字

示例

- 删除放在回收站中的表 __recycle_$_1_1099511628776_1099511677778。

```
OceanBase(admin@test)> create table test(c1 int);
Query OK, 0 rows affected (0.16 sec)

OceanBase(admin@test)> drop table test;
Query OK, 0 rows affected (0.03 sec)

OceanBase(admin@test)> show recyclebin;
+-----------------------------------------+--------------+-------+---------------------------+
| OBJECT_NAME                             | ORIGINAL_NAME | TYPE | CREATETIME                |
+-----------------------------------------+--------------+-------+---------------------------+
| __recycle_$_1_1099511628776_1099511677778 | test       | TABLE | 2017-10-20 17:40:22.304025 |
+-----------------------------------------+--------------+-------+---------------------------+
1 row in set (0.02 sec)

OceanBase(admin@test)> purge table __recycle_$_1_1099511628776_1099511677778;
Query OK, 0 rows affected (0.04 sec)
```

6.10.2.8. SELECT

SELECT 的语法相对比较复杂。本节首先会介绍普通的 SELECT 语法结构，然后介绍集合类 SELECT 的语法结构，最后介绍带有 with clause 的 SELECT。

SIMPLE SELECT

描述

该语句用于查询表中的内容。

格式

```
simple_select:
SELECT [/*+ hint statement */] [DISTINCT | UNIQUE | ALL]
  select_expr_list FROM from_list [WHERE condition]
 [GROUP BY group_expression_list] [ROLLUP group_expression_list] [HAVING condition]]
 [ORDER BY order_expression_list]
 [FOR UPDATE]

select_expr:
  table_name.*
 | table_alias_name.*
 | expr [[AS] column_alias_name]

from_list:
  table_reference [, table_reference ...]

table_reference:
   simple_table
  | joined_table

simple_table:
  table_factor [partition_option] [[AS] table_alias_name]
  | (select_stmt) [AS] table_alias_name
  | (table_reference_list)

joined_table:
   table_reference [INNER] JOIN simple_table [join_condition]
  | table_reference outer_join_type JOIN simple_table join_condition

partition_option:
  PARTITION (partition_name_list)

partition_name_list:
  partition_name [, partition_name ...]
```

```
outer_join_type:
  {LEFT | RIGHT | FULL} [OUTER]

join_condition:
  ON expression

condition:
  expression

group_expression_list:
  group_expression [, group_expression ...]

group_expression:
  expression [ASC | DESC]

order_expression_list:
  order_expression [, order_expression ...]

order_expression:
  expression [ASC | DESC]
```

参数解释

参数	描述
DISTINCT UNIQUE ALL	在数据库表中，可能会包含重复值。 • 指定“DISTINCT”，则在查询结果中相同的行只显示一行； • 指定 “UNIQUE”，则在查询结果中相同的行只显示一行； • 指定“ALL”，则在查询结果中显示所有的行（包含重复），默认为 ALL
select_expr	列出要查询的表达式或列名，用“,”隔开。也可以用“*”表示所有列。
AS othername	为输出字段重新命名。
FROM table_references	指名了从哪个表或哪些表中读取数据（支持多表查询）。

参数	描述
WHERE where_conditions	可选项，WHERE 字句用来设置一个筛选条件，查询结果中仅包含满足条件的数据。where_conditions 为表达式。
GROUP BY group_by_list	按一些字段进行分组，产生统计值
ROLLUP group_expression_list	合并 Group By 的分组，产生统计值
HAVING search_confitions	HAVING 字句与 WHERE 字句类似，但是 HAVING 字句可以使用累计函数（如 SUM，AVG 等）。
ORDER BY order_list order_list： colname [ASC | DESC] [,colname [ASC | DESC]...]	用来按升序（ASC）或者降序（DESC）显示查询结果。不指定 ASC 或者 DESC 时，默认为 ASC。
FOR UPDATE	对查询结果所有行上排他锁，以阻止其他事务的并发修改，或阻止在某些事务隔离级别时的并发读取。
PARTITION(partition_list)	指定查询表的分区信息。例如：partition(p0,p1...)

示例

以如下表 a 为例。

表a

id	name	num
1	a	100
2	b	200
3	a	50

- 从表 a 中读取 name 数据。

```
SELECT name FROM a;
```

```
+------+
| name |
+------+
| a    |
| b    |
| a    |
+------+
3 rows in set (0.01 sec)
```

- 在查询结果中对 name 进行去重处理。

```
SELECT DISTINCT name FROM a;
```

```
+------+
| name |
+------+
| a    |
| b    |
+------+
2 rows in set (0.01 sec)
```

- 从表 a 中查询 id，name，num，然后把 num 列除以2输出，输出的列名为 avg。

```
SELECT id, name, num/2 AS avg FROM a;
```

```
+------+------+------+
| id   | name | avg  |
+------+------+------+
|    1 | a    |   50 |
|    2 | b    |  100 |
|    3 | a    |   25 |
+------+------+------+
3 rows in set (0.00 sec)
```

- 从表 a 中根据筛选条件 “ name = 'a' ” ，输出对应的 id 、name 和 num 。

```
SELECT id, name, num FROM a WHERE name = 'a';
```

```
+------+------+------+
| id   | name | num  |
+------+------+------+
|    1 | a    |  100 |
|    3 | a    |   50 |
+------+------+------+
2 rows in set (0.01 sec)
```

- 从表 a 中查询 id，name，按照 name 分组对 num 求和，并输出。

```
SELECT id, name, SUM(num) FROM a GROUP BY name;
```

```
+------+------+----------+
| id   | name | SUM(num) |
+------+------+----------+
|    1 | a    |      150 |
|    2 | b    |      200 |
+------+------+----------+
2 rows in set (0.00 sec)
```

- 从表 a 中查询 id，name，按照 name 分组对 num 求和，查询 num 总和小于160的行，并输出。

```
SELECT id, name, SUM(num) as sum FROM a GROUP BY name HAVING SUM(num) < 160;
```

```
+------+------+------+
| id   | name | sum  |
+------+------+------+
|    1 | a    |  150 |
+------+------+------+
1 row in set (0.01 sec)
```

- 从表 a 中查询 id，name，num，根据 num 按升序（ASC）输出查询结果。

```
SELECT * FROM a ORDER BY num ASC;
```

```
+------+------+------+
| id   | name | num  |
+------+------+------+
|    3 | a    |   50 |
|    1 | a    |  100 |
|    2 | b    |  200 |
+------+------+------+
3 rows in set (0.00 sec)
```

- 从表 a 中查询 id，name，num，根据 num 按降序（DESC）输出查询结果。

```
SELECT * FROM a ORDER BY num DESC;
```

```
+------+------+------+
| id   | name | num  |
+------+------+------+
|    2 | b    |  200 |
|    1 | a    |  100 |
|    3 | a    |   50 |
+------+------+------+
3 rows in set (0.01 sec)
```

集合类 SELECT

描述

该语句用于对多个 SELECT 查询的结果进行 UNION，MINUS，INTERSECT。

格式

```
select_clause_set:
 simple_select [ UNION | UNION ALL | | INTERSECT] select_clause_set_right
 [ORDER BY sort_list_columns]


select_clause_set_right:
 simple_select |
 select_caluse_set
```

参数解释

参数	描述
UNION ALL	合并两个查询的结果
UNION	合并两个查询的结果，并去重
MINUS	从左查询结果集中去重出现在右查询中的结果，并去重
INTERSECT	保留左查询结果集中出现在右查询中的结果，并去重

示例

以如下两表的数据为例：

```
create table t1 (c1 int, c2 int);
create table t2 (c1 int, c2 int);
insert into t1 values (1, -1), (2, -2);
insert into t2 values (1, 1), (2, -2), (3, 3);
```

- 计算 T1, T2 的所有的记录

```
SELECT C1, C2 FROM T1 UNION ALL SELECT C1, C2 FROM T2;
+------+------+
| C1  | C2  |
+------+------+
|  1|  -1|
|  2|  -2|
|  1|   1|
|  2|  -2|
|  3|   3|
+------+------+
```

- 计算 T1, T2 的去重后的所有记录

```
SELECT C1, C2 FROM T1 UNION SELECT C1, C2 FROM T2;
+------+------+
|C1  |C2  |
+------+------+
|  1|  -1|
|  2|  -2|
|  1|   1|
|  3|   3|
+------+------+
```

- 计算 T1 和 T2 的交集

```
SELECT C1, C2 FROM T1 INTERSECT SELECT C1, C2 FROM T2;
+------+------+
|C1  |C2  |
+------+------+
|  2|  -2|
+------+------+
```

- 计算 T1 和 T2 的差集

```
SELECT C1, C2 FROM T1 MINUS SELECT C1, C2 FROM T2;
+------+------+
|C1  |C2  |
+------+------+
|  1|  -1|
+------+------+
```

带有 with clause 的 SELECT

描述

如果查询语句中有多个相同的子查询，可以把相同的子查询放在 with clause 作为公共表达式，在主体查询中直接引用即可。

格式

```
with_clause_select:
  with_clause simple_select

with_clause:
  WITH table_name [opt_column_alias_name_list] AS ( select_clause )

select_clause:
  simple_select | select_clause_set

opt_column_alias_name_list:
  (column_name_list)

column_name_list:
  column_name | column_name , column_name_list
```

参数解释

无

示例

- 以如下表格数据和 SELECT 查询为例。

```
create table t1(c1 int, c2 int, c3 int);
create table t2(c1 int);
insert into t1 values(1,1,1);
insert into t1 values(2,2,2);
insert into t1 values(3,3,3);
insert into t2 values(4);

select * from t1 where c1 > (select count(*) from t2)
              and c2 > (select count(*) from t2)
              and c3 > (select count(*) from t2);
+------+------+------+
| C1   | C2   | C3   |
+------+------+------+
|  2|  2|  2|
|  3|  3|  3|
+------+------+------+
```

可以抽取相同子查询为 with clause：

```
with temp(cnt) as (select count(*) from t2)
select t1.* from t1, temp where c1 > temp.cnt and c2 > temp.cnt and c3 > temp.cnt;
+------+------+------+
| C1   | C2   | C3   |
+------+------+------+
|    2 |    2 |    2 |
|    3 |    3 |    3 |
+------+------+------+
```

6.10.2.9. UPDATE

描述

该语句用于修改表中的字段值。

格式

```
UPDATE [hint_options] dml_table_clause
    SET update_asgn_list
    [WHERE where_condition]
    [{ RETURNING | RETURN } returning_exprs [into_clause]]

dml_table_clause:
  dml_table_name opt_table_alias

update_asgn_list:
  column_name = expr [, ...]

where_condition:
  expression

returning_exprs:
  projection [, ...]

into_clause:
{ INTO into_var_list | BULK COLLECT INTO into_var_list}

into_var_list:
{ USER_VARIABLE | ref_name } [, ...]
```

参数解释

参数	描述
hint_options	指定 hint 选项。
dml_table_clause	指定修改的表名（基表、可更新视图、特殊子查询）
where_condition	指定过滤条件。
update_asgn_list	指定更新列表。
returning_exprs	返回修改数据后的投影列。
into_clause	将修改数据后的投影列插入到指定列表

> **注意**
>
> 特殊子查询指的类似于可更新视图对应的子查询，这类子查询不应该包含复杂的算子（比如group by/distinct/window function等等）

示例

创建示例表 t1 和 t2。

```
OceanBase(admin@test)>create table t1(c1 int primary key, c2 int);
Query OK, 0 rows affected (0.16 sec)
OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.06 sec)
```

- 单表更新：将表 t1 中 "t1.c1=1" 对应的那一行数据的 c2 列值修改为 100。

```
OceanBase(admin@test)>update t1 set t1.c2 = 100 where t1.c1 = 1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 |  100 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.01 sec)
```

- 单表更新：直接操作子查询，将子查询 中 "v.c1=1" 对应的那一行数据的 c2 列值修改为 100。

```
OceanBase(admin@test)>update (select * from t1)v set v.c2 = 100 where v.c1 = 1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

OceanBase(admin@test)>select * from t1;
+----+------+
| C1 | C2   |
+----+------+
|  1 |  100 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
+----+------+
4 rows in set (0.01 sec)
```

- 单表更新：包含 returning 子句。

```
OceanBase(admin@test)>update t1 set t1.c2 = 100 where t1.c1 = 1 returning c2;
+------+
| C2   |
+------+
| 100 |
+------+
1 row in set (0.02 sec)

OceanBase(admin@test)>select * from t1;
+----+------+
| C1 | C2   |
+----+------+
| 1 | 100 |
| 2 |   2 |
| 3 |   3 |
| 4 |   4 |
+----+------+
4 rows in set (0.01 sec)
```

6.10.3. DCL

6.10.3.1. EXPLAIN

描述

该语句用于解释 SQL 语句的执行计划，可以是 SELECT、DELETE、INSERT、REPLACE 或 UPDATE 语句。提供正则过滤功能，通过 session 变量 explain_regex 对输出行 /JSON 进行过滤。

格式

```
获取表或列的信息：
{EXPLAIN | DESCRIBE | DESC} tbl_name [col_name | wild]

获取SQL计划信息：
{EXPLAIN}
[BASIC | OUTLINE | EXTENDED | EXTENDED_NOADDR | PARTITIONS | FORMAT = {TRADITIONAL| JSON}]
{SELECT statement | DELETE statement | INSERT statement | UPDATE statement | MERGE statement}
```

参数解释

参数	描述
tbl_name	指定表名。
col_name	指定表的列名。
BASIC	指定输出计划的基础信息，如算子ID、算子名称、所引用的表名。
OUTLINE	指定输出的计划信息包含outline信息。
EXTENDED	EXPLAIN产生附加信息，包括：每个算子的输入列和输出列，访问表的分区信息，当前使用的filter信息，如果当前算子使用了索引，显示所使用的索引列及抽取的query range。
EXTENDED_NOADDR	以简约的方式展示附加信息。
PARTITIONS	显示分区相关信息。
FORMAT = {TRADITIONAL| JSON}	指定EXPALIN的输出格式： • TRADITIONAL：表格输出格式 • JSON：KEY:VALUE输出格式，JSON显示为JSON字符串，包括EXTENDED和PARTITIONS信息。

示例

- 省略 explain_type

```
OceanBase(admin@test)>explain select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: ====================================
|ID|OPERATOR  |NAME|EST. ROWS|COST  |
------------------------------------
|0 |HASH JOIN |  |9801000 |5933109|
|1 | TABLE SCAN|t2 |10000  |6219  |
|2 | TABLE SCAN|t1 |100000  |68478 |
====================================

Outputs & filters:
-------------------------------------
 0 - output([t1.c1], [t1.c2], [t2.c1], [t2.c2]), filter(nil),
    equal_conds([t1.c2 = t2.c2]), other_conds(nil)
 1 - output([t2.c2], [t2.c1]), filter(nil),
    access([t2.c2], [t2.c1]), partitions(p0)
 2 - output([t1.c2], [t1.c1]), filter(nil),
    access([t1.c2], [t1.c1]), partitions(p0)
```

- EXTENDED

```
OceanBase(admin@test)>explain extended_noaddr select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: ====================================
|ID|OPERATOR  |NAME|EST. ROWS|COST   |
------------------------------------
|0 |HASH JOIN |    |9801000  |5933109|
|1 | TABLE SCAN|t2  |10000    |6219   |
|2 | TABLE SCAN|t1  |100000   |68478  |
====================================

Outputs & filters:
-------------------------------------
 0 - output([t1.c1], [t1.c2], [t2.c1], [t2.c2]), filter(nil),
    equal_conds([t1.c2 = t2.c2]), other_conds(nil)
 1 - output([t2.c2], [t2.c1]), filter(nil),
    access([t2.c2], [t2.c1]), partitions(p0),
    is_index_back=false,
    range_key([t2.c1]), range(4 ; MAX),
    range_cond([t2.c1 > 4])
 2 - output([t1.c2], [t1.c1]), filter(nil),
    access([t1.c2], [t1.c1]), partitions(p0),
    is_index_back=false,
    range_key([t1.__pk_increment], [t1.__pk_cluster_id], [t1.__pk_partition_id]), range(MIN,MIN,MIN ; MAX,MAX,MAX)always true
```

- TRADITIONAL格式

```
OceanBase(admin@test)>explain format=TRADITIONAL select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: ======================================
|ID|OPERATOR  |NAME|EST. ROWS|COST  |
--------------------------------------
|0 |HASH JOIN |  |9801000  |5933109|
|1 | TABLE SCAN|t2  |10000  |6219  |
|2 | TABLE SCAN|t1  |100000  |68478 |
======================================

Outputs & filters:
-------------------------------------
 0 - output([t1.c1], [t1.c2], [t2.c1], [t2.c2]), filter(nil),
    equal_conds([t1.c2 = t2.c2]), other_conds(nil)
 1 - output([t2.c2], [t2.c1]), filter(nil),
    access([t2.c2], [t2.c1]), partitions(p0)
 2 - output([t1.c2], [t1.c1]), filter(nil),
    access([t1.c2], [t1.c1]), partitions(p0)
```

- JSON格式

```
OceanBase(admin@test)>explain format=JSON select * from t1,t2 where t1.c2=t2.c2 and t2.c1 > 4\G
*************************** 1. row ***************************
Query Plan: {
 "ID":2,
 "OPERATOR":"JOIN",
 "NAME":"JOIN",
 "EST.ROWS":9800999,
 "COST":5933108,
 "output": [
  "t1.c1",
  "t1.c2",
  "t2.c1",
  "t2.c2"
 ],
 "TABLE SCAN": {
  "ID":0,
  "OPERATOR":"TABLE SCAN",
  "NAME":"TABLE SCAN",
  "EST.ROWS":10000,
  "COST":6218,
  "output": [
   "t2.c2",
   "t2.c1"
  ]
 },
 "TABLE SCAN": {
  "ID":1,
  "OPERATOR":"TABLE SCAN",
  "NAME":"TABLE SCAN",
  "EST.ROWS":100000,
  "COST":68477,
  "output": [
   "t1.c2",
   "t1.c1"
  ]
 }
}
```

EXPLAIN 的每个输出行提供一个表的相关信息，并且每个行包括下面的列：

列名	说明
ID	计划执行序列号。
OPERATOR	执行算子。
NAME	算子所引用的表。
EST.ROWS	估计执行到当前算子输出的行数。
COST	执行到当前算子的CPU时间。

6.10.3.2. FLASHBACK TABLE BEFORE DROP

描述

用于从回收站中恢复被删除的 TABLE。

前置条件

回收站需要处于开启状态，可以通过 `show variables like 'recyclebin';` 来查看回收站是否开启。

```
obclient> show variables like 'recyclebin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| recyclebin    | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

如果回收站处于关闭状态，可以通过 `set recyclebin = on;` 来开启。回收站中的表没有被实际删除，仍然会占用资源，如果需要彻底删除，可以执行 `purge recyclebin;`

格式

```
FLASHBACK TABLE object_name TO BEFORE DROP [RENAME to db_name.table_name];
```

参数解释

参数	描述
object_name	指定要恢复的object名称或表名，只有在表所在的database中才能执行。恢复TABLE时，也会将和TABLE相关的INDEX也恢复出来。
RENAME to	修改表名和表所属的库。

示例

- 从回收站中恢复被删除的表 t。

```
obclient> create table t(id int primary key, k int);
Query OK, 0 rows affected (0.04 sec)

obclient> insert into t values(1,1);
Query OK, 1 row affected (0.00 sec)

obclient> select * from t;
+----+------+
| id | k    |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.00 sec)

obclient>> drop table t;
Query OK, 0 rows affected (0.01 sec)

obclient> select * from t;
ORA-00942: table or view 'SYS.T' does not exist
obclient> show recyclebin;
+--------------------------------+---------------+-------+----------------------------+
| OBJECT_NAME                    | ORIGINAL_NAME | TYPE  | CREATETIME                 |
+--------------------------------+---------------+-------+----------------------------+
| __recycle_$_1_1597028971700936 | T             | TABLE | 2020-08-10 11:09:31.701033 |
+--------------------------------+---------------+-------+----------------------------+
1 row in set (0.00 sec)

obclient> flashback table t to before drop;
Query OK, 0 rows affected (0.01 sec)

obclient> select * from t;
+----+------+
| id | k    |
+----+------+
|  1 |    1 |
+----+------+
1 row in set (0.00 sec)
```

6.10.3.3. GRANT

描述

该语句用于系统管理员授予 User 某些权限，包括对象权限，系统权限和角色。

> 说明
>
> - 当授予对象权限时，当前用户必须是对象的所有者，或者拥有被授予的权限（例如，user1 把表 t1 的 SELECT 权限授予 user2，则 user1 必须拥有表 t1 的 SELECT 的权限），并且拥有 GRANT OPTION 权限，才能授予成功。
> - 当授予系统权限和角色时，当前用户必须拥有被授予的权限或角色，并且拥有 GRANT OPTION 权限，才能授予成功。
> - 用户授权后，该用户只有重新连接 OceanBase，权限才能生效。

格式

```
/*授予对象权限*/
GRANT obj_with_col_priv_list
  ON obj_clause TO grant_user_list [WITH GRANT OPTION];

obj_with_col_priv_list:
 obj_with_col_priv
 | obj_with_col_priv_list, obj_with_col_priv

obj_with_col_priv:
 obj_privilege [column_list]

obj_privilege:
  ALTER
 | AUDIT
 | COMMENT
 | DELETE
 | GRANT
 | INDEX
 | INSERT
 | LOCK
 | RENAME
 | SELECT
 | UPDATE
 | REFERENCES
 | EXECUTE
 | CREATE
 | FLASHBACK
 | READ
```

```
 | WRITE
 | DEBUG

obj_clause:
 relation_name
 | relation_name '.' relation_name
 | DIRECTORY relation_name

grant_user_list:
  grant_user [, grant_user ...]

/*授予系统权限*/
GRANT {system_privilege_list | ALL PRIVILEGES}
  TO grantee_user [IDENTIFIED BY password] [WITH {GRANT | ADMID} OPTION];

system_privilege_list:
 system_privilege [, system_privilege ...]

system_privilege:
 CREATE SESSION
 | EXEMPT REDACTION POLICY
 | SYSDBA
 | SYSOPER
 | SYSBACKUP
 | CREATE TABLE
 | CREATE ANY TABLE
 | ALTER ANY TABLE
 | BACKUP ANY TABLE
 | DROP ANY TABLE
 | LOCK ANY TABLE
 | COMMENT ANY TABLE
 | SELECT ANY TABLE
 | INSERT ANY TABLE
 | UPDATE ANY TABLE
 | DELETE ANY TABLE
 | FLASHBACK ANY TABLE
 | CREATE ROLE
 | DROP ANY ROLE
 | GRANT ANY ROLE
 | ALTER ANY ROLE
 | AUDIT ANY
```

```
| GRANT ANY PRIVILEGE
| GRANT ANY OBJECT PRIVILEGE
| CREATE ANY INDEX
| ALTER ANY INDEX
| DROP ANY INDEX
| CREATE ANY VIEW
| DROP ANY VIEW
| CREATE VIEW
| SELECT ANY DICTIONARY
| CREATE PROCEDURE
| CREATE ANY PROCEDURE
| ALTER ANY PROCEDURE
| DROP ANY PROCEDURE
| EXECUTE ANY PROCEDURE
| CREATE SYNONYM
| CREATE ANY SYNONYM
| DROP ANY SYNONYM
| CREATE PUBLIC SYNONYM
| DROP PUBLIC SYNONYM
| CREATE SEQUENCE
| CREATE ANY SEQUENCE
| ALTER ANY SEQUENCE
| DROP ANY SEQUENCE
| SELECT ANY SEQUENCE
| CREATE TRIGGER
| CREATE ANY TRIGGER
| ALTER ANY TRIGGER
| DROP ANY TRIGGER
| CREATE PROFILE
| ALTER PROFILE
| DROP PROFILE
| CREATE USER
| ALTER USER
| DROP USER
| CREATE TYPE
| CREATE ANY TYPE
| ALTER ANY TYPE
| DROP ANY TYPE
| EXECUTE ANY TYPE
| UNDER ANY TYPE
```

```
| PURGE DBA_RECYCLEBIN
| CREATE ANY OUTLINE
| ALTER ANY OUTLINE
| DROP ANY OUTLINE
| SYSKM
| CREATE TABLESPACE
| ALTER TABLESPACE
| DROP TABLESPACE
| SHOW PROCESS
| ALTER SYSTEM
| CREATE DATABASE LINK
| CREATE PUBLIC DATABASE LINK
| DROP DATABASE LINK
| ALTER SESSION
| ALTER DATABASE

/*授予角色*/
GRANT role_list TO grantee_user [IDENTIFIED BY password] [WITH {GRANT | ADMID} OPTION];

role_list:
role [, role ...]
```

参数解释

参数	描述
priv_type	指定授予的权限类型。具体的权限类型及其说明请参见下方权限类型说明表。 同时把多个权限赋予用户时，权限类型用“,”隔开。
system_privilege	指定授予的系统权限类型。 同时把多个权限赋予用户时，权限类型用“,”隔开。
obj_clause	指定授予权限的层级。权限可以分为以下几个层级： • 全局层级：适用于所有的数据库。 • 数据库层级：适用于一个给定数据库中的所有目标。 • 表层级：表权限适用于一个给定表中的所有列
WITH GRANT OPTION	指定权限是否允许转授，取消授权时级联。

参数	描述
WITH ADMIN OPTION	指定权限是否允许转授，取消授权时不级联。

可以授予的权限类型如下表所示。

权限类型说明表

权限	说明
ALL PRIVILEGES	除GRANT OPTION以外所有权限。
ALTER	ALTER TABLE的权限。
CREATE	CREATE TABLE的权限。
DELETE	DELETE的权限。
DROP	DROP的权限。
GRANT OPTION	GRANT OPTION的权限。
INSERT	INSERT的权限。
UPDATE	UPDATE的权限。
SELECT	SELECT的权限。
INDEX	CREATE INDEX, DROP INDEX的权限
SHOW VIEW	SHOW CREATE VIEW权限。
SHOW DATABASES	全局 SHOW DATABASES的权限。
SUPER	SET GLOBAL修改全局系统参数的权限。
REFERENCES	创建指向表的约束的权限。

权限	说明
EXECUTE	执行预处理程序的权限。
FLASHBACK	FLASHBACK的权限。
READ	READ的权限。
WRITE	WRITE的权限。
CREATE SESSION	连接到数据库的权限。
EXEMPT REDACTION POLICY	绕过任意现有 REDACTION POLICY 并查看数据的权限。
SYSDBA	SYSDBA的权限。
SYSOPER	SYSOPER的权限。
SYSBACKUP	SYSBACKUP的权限。
CREATE TABLE	在指定用户SCHEMA内创建表的权限。
CREATE ANY TABLE	在除SYS外所有用户SCHEMA内创建表的权限。
ALTER ANY TABLE	在除SYS外所有用户SCHEMA内修改表的权限。
BACKUP ANY TABLE	在除SYS外所有用户SCHEMA内创建表的权限。
DROP ANY TABLE	在除SYS外所有用户SCHEMA内备份表的权限。
LOCK ANY TABLE	在除SYS外所有用户SCHEMA内锁定表的权限。
COMMENT ANY TABLE	在除SYS外所有用户SCHEMA内评论表的权限。
SELECT ANY TABLE	在除SYS外所有用户SCHEMA内查看表的权限。

权限	说明
INSERT ANY TABLE	在除SYS外所有用户SCHEMA内的表插入行的权限。
UPDATE ANY TABLE	在除SYS外所有用户SCHEMA内的表更新行的权限。
DELETE ANY TABLE	在除SYS外所有用户SCHEMA内删除表的权限。
FLASHBACK ANY TABLE	在除SYS外所有用户SCHEMA内FLASHBACK表的权限。
CREATE ROLE	创建角色的权限。
DROP ANY ROLE	删除任意角色的权限。
GRANT ANY ROLE	授予任意角色的权限。
ALTER ANY ROLE	修改任意角色的权限。
AUDIT ANY	在除SYS外所有用户SCHEMA内修改对象的权限。
GRANT ANY PRIVILEGE	授予任意系统权限的权限。
GRANT ANY OBJECT PRIVILEGE	授予任意对象权限的权限。
CREATE ANY INDEX	在除SYS外所有用户SCHEMA内创建索引的权限。
ALTER ANY INDEX	在除SYS外所有用户SCHEMA内修改索引的权限。
DROP ANY INDEX	在除SYS外所有用户SCHEMA内删除索引的权限。
CREATE ANY VIEW	在除SYS外所有用户SCHEMA内创建视图的权限。
DROP ANY VIEW	在除SYS外所有用户SCHEMA内删除索引的权限。
CREATE VIEW	在指定用户SCHEMA内创建视图的权限。

权限	说明
SELECT ANY DICTIONARY	在指定用户SCHEMA内查询DICTIONARY的权限。
CREATE PROCEDURE	在指定用户SCHEMA内创建PROCEDURE的权限。
CREATE ANY PROCEDURE	在除SYS外所有用户SCHEMA内创建PROCEDURE的权限。
ALTER ANY PROCEDURE	在除SYS外所有用户SCHEMA内修改PROCEDURE的权限。
DROP ANY PROCEDURE	在除SYS外所有用户SCHEMA内删除PROCEDURE的权限。
EXECUTE ANY PROCEDURE	在除SYS外所有用户SCHEMA内执行PROCEDURE的权限。
CREATE SYNONYM	在指定用户SCHEMA内创建SYNONYM的权限。
CREATE ANY SYNONYM	在除SYS外所有用户SCHEMA内创建SYNONYM的权限。
DROP ANY SYNONYM	在除SYS外所有用户SCHEMA内删除SYNONYM的权限。
CREATE PUBLIC SYNONYM	创建公共SYNONYM的权限。
DROP PUBLIC SYNONYM	删除公共SYNONYM的权限。
CREATE SEQUENCE	在指定用户SCHEMA内创建SEQUENCE的权限。
CREATE ANY SEQUENCE	在除SYS外所有用户SCHEMA内创建SEQUENCE的权限。
ALTER ANY SEQUENCE	在除SYS外所有用户SCHEMA内修改SEQUENCE的权限。
DROP ANY SEQUENCE	在除SYS外所有用户SCHEMA内删除SEQUENCE的权限。
SELECT ANY SEQUENCE	在除SYS外所有用户SCHEMA内查询SEQUENCE的权限。
CREATE TRIGGER	在指定用户SCHEMA内创建TRIGGER的权限。

权限	说明
CREATE ANY TRIGGER	在除SYS外所有用户SCHEMA内创建TRIGGER的权限。
ALTER ANY TRIGGER	在除SYS外所有用户SCHEMA内修改TRIGGER的权限。
DROP ANY TRIGGER	在除SYS外所有用户SCHEMA内删除TRIGGER的权限。
CREATE PROFILE	创建PROFILE的权限。
ALTER PROFILE	修改PROFILE的权限。
DROP PROFILE	删除PROFILE的权限。
CREATE USER	创建用户的权限。
ALTER USER	修改用户的权限。
DROP USER	删除用户的权限。
CREATE TYPE	在指定用户SCHEMA内创建TYPE的权限。
CREATE ANY TYPE	在除SYS外所有用户SCHEMA内创建TYPE的权限。
ALTER ANY TYPE	在除SYS外所有用户SCHEMA内修改TYPE的权限。
DROP ANY TYPE	在除SYS外所有用户SCHEMA内删除TYPE的权限。
EXECUTE ANY TYPE	在除SYS外所有用户SCHEMA内执行TYPE的权限。
UNDER ANY TYPE	在除SYS外所有用户SCHEMA内TYPE的基础上创建SUBTYPE的权限。
PURGE DBA_RECYCLEBIN	从系统回收站中删除所有对象的权限。
CREATE ANY OUTLINE	在除SYS外所有用户SCHEMA内创建OUTLINE的权限。

权限	说明
ALTER ANY OUTLINE	在除SYS外所有用户SCHEMA内修改OUTLINE的权限。
DROP ANY OUTLINE	在除SYS外所有用户SCHEMA内删除OUTLINE的权限。
SYSKM	SYSKM的权限。
CREATE TABLESPACE	创建表空间的权限。
ALTER TABLESPACE	修改表空间的权限。
DROP TABLESPACE	删除表空间的权限。
ALTER SYSTEM	ALTER SYSTEM的权限
CREATE DATABASE LINK	在指定用户SCHEMA内创建DATABASE LINK的权限。
CREATE PUBLIC DATABASE LINK	创建PUBLIC DATABASE LINK的权限。
DROP DATABASE LINK	在指定用户SCHEMA内删除DATABASE LINK的权限。
ALTER SESSION	修改SESSION的权限。
ALTER DATABASE	修改DATABASE的权限。

示例

- 执行以下命令给用户 obsqluser01 赋予所有权限。

```
OceanBase(admin@TEST)>GRANT ALL PRIVILEGES ON *.* TO obsqluser01 with grant option;
Query OK, 0 rows affected (0.03 sec)
```

6.10.3.4. KILL

描述

该语句用来终止一个会话。

> **说明**
>
> 如果您拥有 PROCESS 权限，则您可以查看所有会话。如果您拥有 SUPER 权限，您可以终止所有会话和语句。否则，您只能查看和终止您自己的会话和语句。

格式

```
KILL [CONNECTION | QUERY] 'sessionid'
```

参数解释

参数	描述
KILL	与 KILL CONNECTION 一样，可终止给定的threadid。
KILL CONNECTION	与不含修改符的KILL一样，可终止给定的threadid。
KILL QUERY	可终止连接当前正在执行的语句，但是会保持连接的原状。

示例

- 终止 sessionid 为 3221638213 的会话连接当前正在执行的语句后，再终止该会话。

```
OceanBase(admin@test)>show processlist;
+------------+-------+--------------------+------+---------+------+--------+------------------+
| Id         | User  | Host               | db   | Command | Time | State  | Info             |
+------------+-------+--------------------+------+---------+------+--------+------------------+
| 3221638212 | admin | 1.11.111.127:11161 | test | Query   |    0 | ACTIVE | show processlist |
| 3221638213 | admin | 1.11.111.127:11161 | test | Query   |    0 | ACTIVE | select "abcedfg" |
+------------+-------+--------------------+------+---------+------+--------+------------------+
1 row in set (0.03 sec)

OceanBase(admin@test)>kill query 3221638213;
Query OK, 0 rows affected (0.01 sec)

OceanBase(admin@test)>kill 3221638212;
Query OK, 0 rows affected (0.01 sec)
```

6.10.3.5. REVOKE

描述

该语句用于系统管理员撤销 User 的某些权限。

使用说明如下：

- 用户必须拥有被撤销的权限（例如，user1 要撤销 user2 对表 t1 的 SELECT 权限，则 user1 必须拥有表 t1 的 SELECT 的权限），并且拥有 GRANT OPTION 权限。
- 撤销 ALL PRIVILEGES 和 GRANT OPTION 权限时，当前用户必须拥有全局 GRANT OPTION 权限，或者对权限表的 UPDATE 及 DELETE 权限。
- 当权限没有 GRANT OPTION 时，撤销操作不会级联。例如，用户 user1 给 user2 授予了某些权限，撤回 user1 的权限不会同时也撤回 user2 的相应权限。当拥有 GRANT OPTION 时，撤销操作会级联。

格式

```
/*撤销对象权限*/
REVOKE obj_privileges
 ON obj_clause FROM user_list;

user_list:
 user [, user ...]

obj_privileges:
 obj_privilege [, obj_privilege ...]

obj_privilege:
  ALTER
 | AUDIT
 | COMMENT
 | DELETE
 | GRANT
 | INDEX
 | INSERT
 | LOCK
 | RENAME
 | SELECT
 | UPDATE
 | REFERENCES
 | EXECUTE
 | CREATE
 | FLASHBACK
 | READ
 | WRITE
 | DEBUG
```

```
obj_clause:
 relation_name
 | relation_name '.' relation_name
 | DIRECTORY relation_name

relation_name:
 STR_VALUE

/*撤销系统权限*/
REVOKE {system_privilege_list | ALL PRIVILEGES}
 FROM user_list;
REVOKE ALL [PRIVILEGES], GRANT_OPTION FROM user_list;

system_privilege_list:
 system_privilege [, system_privilege ...]

system_privilege:
 CREATE SESSION
 | EXEMPT REDACTION POLICY
 | SYSDBA
 | SYSOPER
 | SYSBACKUP
 | CREATE TABLE
 | CREATE ANY TABLE
 | ALTER ANY TABLE
 | BACKUP ANY TABLE
 | DROP ANY TABLE
 | LOCK ANY TABLE
 | COMMENT ANY TABLE
 | SELECT ANY TABLE
 | INSERT ANY TABLE
 | UPDATE ANY TABLE
 | DELETE ANY TABLE
 | FLASHBACK ANY TABLE
 | CREATE ROLE
 | DROP ANY ROLE
 | GRANT ANY ROLE
 | ALTER ANY ROLE
 | AUDIT ANY
 | GRANT ANY PRIVILEGE
```

```
| GRANT ANY OBJECT PRIVILEGE
| CREATE ANY INDEX
| ALTER ANY INDEX
| DROP ANY INDEX
| CREATE ANY VIEW
| DROP ANY VIEW
| CREATE VIEW
| SELECT ANY DICTIONARY
| CREATE PROCEDURE
| CREATE ANY PROCEDURE
| ALTER ANY PROCEDURE
| DROP ANY PROCEDURE
| EXECUTE ANY PROCEDURE
| CREATE SYNONYM
| CREATE ANY SYNONYM
| DROP ANY SYNONYM
| CREATE PUBLIC SYNONYM
| DROP PUBLIC SYNONYM
| CREATE SEQUENCE
| CREATE ANY SEQUENCE
| ALTER ANY SEQUENCE
| DROP ANY SEQUENCE
| SELECT ANY SEQUENCE
| CREATE TRIGGER
| CREATE ANY TRIGGER
| ALTER ANY TRIGGER
| DROP ANY TRIGGER
| CREATE PROFILE
| ALTER PROFILE
| DROP PROFILE
| CREATE USER
| ALTER USER
| DROP USER
| CREATE TYPE
| CREATE ANY TYPE
| ALTER ANY TYPE
| DROP ANY TYPE
| EXECUTE ANY TYPE
| UNDER ANY TYPE
| PURGE DBA_RECYCLEBIN
| CREATE ANY OUTLINE
```

```
| ALTER ANY OUTLINE
| DROP ANY OUTLINE
| SYSKM
| CREATE TABLESPACE
| ALTER TABLESPACE
| DROP TABLESPACE
| SHOW PROCESS
| ALTER SYSTEM
| CREATE DATABASE LINK
| CREATE PUBLIC DATABASE LINK
| DROP DATABASE LINK
| ALTER SESSION
| ALTER DATABASE

/*撤销角色*/
REVOKE role_list FROM user;

role_list:
role [, role ...]
```

参数解释

参数	描述
obj_privileges	指定撤销的对象权限类型。具体的权限类型及其说明请参见下方权限类型说明表。 同时撤销多个权限时，权限类型用“,”隔开。
system_privilege	指定撤销的系统权限类型。 同时撤销多个权限时，权限类型用“,”隔开。
obj_clause	指定撤销权限的层级，relation_name指定具体对象的名称。权限可以分为以下几个层级： • 全局层级：适用于所有的数据库。 • 数据库层级：适用于一个给定数据库中的所有目标。 • 表层级：表权限适用于一个给定表中的所有列。

可以撤销的权限类型如下表所示。

权限类型说明表

权限	说明
ALL PRIVILEGES	除GRANT OPTION以外所有权限。
ALTER	ALTER TABLE的权限。
CREATE	CREATE TABLE的权限。
DELETE	DELETE的权限。
DROP	DROP的权限。
GRANT OPTION	GRANT OPTION的权限。
INSERT	INSERT的权限。
UPDATE	UPDATE的权限。
SELECT	SELECT的权限。
INDEX	CREATE INDEX, DROP INDEX的权限
SHOW VIEW	SHOW CREATE VIEW权限。
SHOW DATABASES	全局 SHOW DATABASES的权限。
SUPER	SET GLOBAL修改全局系统参数的权限。
REFERENCES	创建指向表的约束的权限。
EXECUTE	执行预处理程序的权限。
FLASHBACK	FLASHBACK的权限。
READ	READ的权限。

权限	说明
WRITE	WRITE的权限。
CREATE SESSION	连接到数据库的权限。
EXEMPT REDACTION POLICY	绕过任意现有 REDACTION POLICY 并查看数据的权限。
SYSDBA	SYSDBA的权限。
SYSOPER	SYSOPER的权限。
SYSBACKUP	SYSBACKUP的权限。
CREATE TABLE	在指定用户SCHEMA内创建表的权限。
CREATE ANY TABLE	在除SYS外所有用户SCHEMA内创建表的权限。
ALTER ANY TABLE	在除SYS外所有用户SCHEMA内修改表的权限。
BACKUP ANY TABLE	在除SYS外所有用户SCHEMA内创建表的权限。
DROP ANY TABLE	在除SYS外所有用户SCHEMA内备份表的权限。
LOCK ANY TABLE	在除SYS外所有用户SCHEMA内锁定表的权限。
COMMENT ANY TABLE	在除SYS外所有用户SCHEMA内评论表的权限。
SELECT ANY TABLE	在除SYS外所有用户SCHEMA内查看表的权限。
INSERT ANY TABLE	在除SYS外所有用户SCHEMA内的表插入行的权限。
UPDATE ANY TABLE	在除SYS外所有用户SCHEMA内的表更新行的权限。
DELETE ANY TABLE	在除SYS外所有用户SCHEMA内删除表的权限。

权限	说明
FLASHBACK ANY TABLE	在除SYS外所有用户SCHEMA内FLASHBACK表的权限。
CREATE ROLE	创建角色的权限。
DROP ANY ROLE	删除任意角色的权限。
GRANT ANY ROLE	授予任意角色的权限。
ALTER ANY ROLE	修改任意角色的权限。
AUDIT ANY	在除SYS外所有用户SCHEMA内修改对象的权限。
GRANT ANY PRIVILEGE	授予任意系统权限的权限。
GRANT ANY OBJECT PRIVILEGE	授予任意对象权限的权限。
CREATE ANY INDEX	在除SYS外所有用户SCHEMA内创建索引的权限。
ALTER ANY INDEX	在除SYS外所有用户SCHEMA内修改索引的权限。
DROP ANY INDEX	在除SYS外所有用户SCHEMA内删除索引的权限。
CREATE ANY VIEW	在除SYS外所有用户SCHEMA内创建视图的权限。
DROP ANY VIEW	在除SYS外所有用户SCHEMA内删除索引的权限。
CREATE VIEW	在指定用户SCHEMA内创建视图的权限。
SELECT ANY DICTIONARY	在指定用户SCHEMA内查询DICTIONARY的权限。
CREATE PROCEDURE	在指定用户SCHEMA内创建PROCEDURE的权限。
CREATE ANY PROCEDURE	在除SYS外所有用户SCHEMA内创建PROCEDURE的权限。

权限	说明
ALTER ANY PROCEDURE	在除SYS外所有用户SCHEMA内修改PROCEDURE的权限。
DROP ANY PROCEDURE	在除SYS外所有用户SCHEMA内删除PROCEDURE的权限。
EXECUTE ANY PROCEDURE	在除SYS外所有用户SCHEMA内执行PROCEDURE的权限。
CREATE SYNONYM	在指定用户SCHEMA内创建SYNONYM的权限。
CREATE ANY SYNONYM	在除SYS外所有用户SCHEMA内创建SYNONYM的权限。
DROP ANY SYNONYM	在除SYS外所有用户SCHEMA内删除SYNONYM的权限。
CREATE PUBLIC SYNONYM	创建公共SYNONYM的权限。
DROP PUBLIC SYNONYM	删除公共SYNONYM的权限。
CREATE SEQUENCE	在指定用户SCHEMA内创建SEQUENCE的权限。
CREATE ANY SEQUENCE	在除SYS外所有用户SCHEMA内创建SEQUENCE的权限。
ALTER ANY SEQUENCE	在除SYS外所有用户SCHEMA内修改SEQUENCE的权限。
DROP ANY SEQUENCE	在除SYS外所有用户SCHEMA内删除SEQUENCE的权限。
SELECT ANY SEQUENCE	在除SYS外所有用户SCHEMA内查询SEQUENCE的权限。
CREATE TRIGGER	在指定用户SCHEMA内创建TRIGGER的权限。
CREATE ANY TRIGGER	在除SYS外所有用户SCHEMA内创建TRIGGER的权限。
ALTER ANY TRIGGER	在除SYS外所有用户SCHEMA内修改TRIGGER的权限。
DROP ANY TRIGGER	在除SYS外所有用户SCHEMA内删除TRIGGER的权限。

权限	说明
CREATE PROFILE	创建PROFILE的权限。
ALTER PROFILE	修改PROFILE的权限。
DROP PROFILE	删除PROFILE的权限。
CREATE USER	创建用户的权限。
ALTER USER	修改用户的权限。
DROP USER	删除用户的权限。
CREATE TYPE	在指定用户SCHEMA内创建TYPE的权限。
CREATE ANY TYPE	在除SYS外所有用户SCHEMA内创建TYPE的权限。
ALTER ANY TYPE	在除SYS外所有用户SCHEMA内修改TYPE的权限。
DROP ANY TYPE	在除SYS外所有用户SCHEMA内删除TYPE的权限。
EXECUTE ANY TYPE	在除SYS外所有用户SCHEMA内执行TYPE的权限。
UNDER ANY TYPE	在除SYS外所有用户SCHEMA内TYPE的基础上创建SUBTYPE的权限。
PURGE DBA_RECYCLEBIN	从系统回收站中删除所有对象的权限。
CREATE ANY OUTLINE	在除SYS外所有用户SCHEMA内创建OUTLINE的权限。
ALTER ANY OUTLINE	在除SYS外所有用户SCHEMA内修改OUTLINE的权限。
DROP ANY OUTLINE	在除SYS外所有用户SCHEMA内删除OUTLINE的权限。
SYSKM	SYSKM的权限。

权限	说明
CREATE TABLESPACE	创建表空间的权限。
ALTER TABLESPACE	修改表空间的权限。
DROP TABLESPACE	删除表空间的权限。
ALTER SYSTEM	ALTER SYSTEM的权限
CREATE DATABASE LINK	在指定用户SCHEMA内创建DATABASE LINK的权限。
CREATE PUBLIC DATABASE LINK	创建PUBLIC DATABASE LINK的权限。
DROP DATABASE LINK	在指定用户SCHEMA内删除DATABASE LINK的权限。
ALTER SESSION	修改SESSION的权限。
ALTER DATABASE	修改DATABASE的权限。

示例

执行以下命令撤销用户 obsqluser01 的所有权限。

```
OceanBase(admin@TEST)>REVOKE ALL PRIVILEGES FROM sqluser;
Query OK, 0 rows affected (0.10 sec)
```

6.10.3.6. SAVEPOINT

描述

SAVEPOINT 语句用来实现事务的部分回滚。

格式

1. 创建savepoint：

```
SAVEPOINT spname
```

2. 回滚到savepoint：

```
ROLLBACK [WORK] to [SAVEPOINT] spname
```

3. 删除savepoint：

```
RELEASE SAVEPOINT spname
```

参数说明

参数	描述
spname	指定savepoint的名称。savepoint是事务范围内唯一的，一个savepoint会覆盖前一个同名的savepoint。创建savepoint后可以将事务回滚到指定savepoint，也可以使用 `ROLLBACK` 语句回滚整个事务。

示例

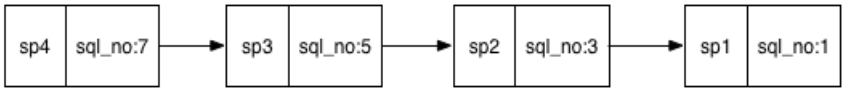
假设一个事务执行了如下语句：

sql_no	语句	分区
1	update...	p1, p4
	savepoint sp1	
2	update...	p2, p4
3	update...	p3, p5
	savepoint sp2	
4	update...	p1, p3, p6
5	update...	p1, p5
	savepoint sp3	

6	select...	
7	update...	p5, p6
	savepoint sp4	

记录savepoint

用户在提交事务之前可以创建savepoint，需要根据savepoint创建的顺序，将事务的savepoint串成链表。以上事务包含了7条sql和4个savepoint，记录savepoint的链表如下图所示，其中每个节点记录了<spname, sql_no>的映射关系：



事务参与者列表

事务为了支持回滚某条sql之后的所有修改，需要将每条语句涉及的参与者以及对应的sql_no记录下来，以上事务执行了7条sql，涉及p1~p6共6个partition：

p1	p2	p3	p4	p5	p6
1,4,5	2	2,3,4	1,2	3,5,7	4,7

savepoint回滚过程

1. 根据savepoint链表查询spname对应的sql_no

假设用户执行 `ROLLBACK to SAVEPOINT sp2` ，根据savepoint链表查询到sp2对应的sql_no为3。

2. 根据事务参与者列表查询sql_no对应的partition

根据事务参与者列表查询到sql_no大于3的语句操作的分区涉及p1、p3、p5、p6。

3. 回滚分区数据

根据第2步查询到的分区，调度程序向这些分区发起回滚请求，回滚当前事务在这些分区上sp2之后的所有修改。其中p1、p3、p5上关于本事务的部分修改被回滚掉，p6上关于本事务的所有修改都被回滚掉。

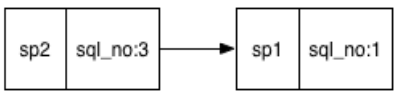
4. 更新事务参与者列表信息

修改事务参与者列表，将sql_no大于3的操作信息从事务参与者列表中删除，由于p6上的所有修改都被回滚掉，因此p6可以从参与者列表中删除。

p1	p2	p3	p4	p5	p6
1	2	2, 3	1, 2	3	

5. 删除无效的savepoint

用户执行 `ROLLBACK to SAVEPOINT sp2` 成功后，系统会删除sp3和sp4的savepoint，不允许再回滚到sp3和sp4。



6.10.3.7. SET NAMES

描述

该语句用来设置当前连接的数据库的字符集编码。

格式

```
SET NAMES 'charset_name' [COLLATE 'collation_name']
```

参数解释

参数	描述
charset_name	指定字符集
collation_name	指定字符集排序规则。如未指定，则使用字符集默认排序规则。

> 说明
>
> 该语句将三个会话系统变量 `character_set_client` ， `character_set_connection` 和 `character_set_results` 设置为给定的字符集。将 `character_set_connection` 设置为 `charset_name` ，将 `collation_connection` 设置为 `charset_name` 的默认排序规则。

示例

- 设置数据库的字符集和排序规则。

```
OceanBase(admin@test)>SET NAMES 'utf8mb4' COLLATE 'utf8mb4_general_ci';
Query OK, 0 rows affected (0.00 sec)

OceanBase(admin@test)>show variables like 'character_set_c%';
+--------------------------+---------+
| Variable_name            | Value   |
+--------------------------+---------+
| character_set_client     | utf8mb4 |
| character_set_connection | utf8mb4 |
+--------------------------+---------+
2 rows in set (0.01 sec)

OceanBase(admin@test)>show variables like 'collation_connection';
```

```
+----------------------+--------------------+
| Variable_name      | Value          |
+----------------------+--------------------+
| collation_connection | utf8mb4_general_ci |
+----------------------+--------------------+
1 row in set (0.05 sec)


OceanBase(admin@test)>SET NAMES 'gbk';
Query OK, 0 rows affected (0.02 sec)

OceanBase(admin@test)>show variables like 'character_set_c%';
+--------------------------+-------+
| Variable_name          | Value |
+--------------------------+-------+
| character_set_client     | gbk   |
| character_set_connection | gbk   |
+--------------------------+-------+
2 rows in set (0.01 sec)


OceanBase(admin@test)>show variables like 'collation_connection';
+----------------------+----------------+
| Variable_name      | Value        |
+----------------------+----------------+
| collation_connection | gbk_chinese_ci |
+----------------------+----------------+
1 row in set (0.01 sec)
```

6.10.3.8. SET PASSWORD

描述

该语句用于修改 OceanBase 当前登录用户或其他用户的密码。

格式

```
SET PASSWORD [FOR user] = PASSWORD(password);
ALTER USER user IDENTIFIED BY password;

password:
 STR_VALUE
```

参数解释

参数	描述
FOR user	如果没有 For user 子句，表示修改当前用户的密码。任何成功登录的用户都可以修改当前用户的密码。 如果有 For user 子句，表示修改指定用户的密码。必须拥有 CREATE USER 的系统权限，才可以修改指定用户的密码。

示例

- 执行以下命令将 sqluser 用户的密码修改为 abc123。

```
oceanBase(admin@TEST)>SET PASSWORD for sqluser = PASSWORD(abc123);
Query OK, 0 rows affected (0.02 sec)
```

6.10.3.9. SET VARIABLE

描述

该语句用来设置用户自定义和系统定义的 session 变量。

格式

```
SET var1_name = var1_value, var2_name = var2_value, ...;
```

参数解释

参数	描述
var1_name	指定要设置的变量，可以是用户自定义的变量，也可以是系统变量。如果要设置多个变量，中间用 , 分隔。

示例

```
OceanBase(TEST@TEST)>set @a = 1, @b = 2, @c = 3;
Query OK, 0 rows affected (0.02 sec)

OceanBase(TEST@TEST)>set @@tx_isolation = 'read-what';
ORA-00600: internal error code, arguments: -5145, Variable 'tx_isolation' can't be set to the value of 'read-w
hat'

OceanBase(TEST@TEST)>set @@tx_isolation = 'read-committed';
Query OK, 0 rows affected (0.01 sec)

OceanBase(TEST@TEST)>select @a,@b,@c,@@tx_isolation from dual;
+------+------+------+----------------+
|@A  |@B  |@C  |@@TX_ISOLATION|
+------+------+------+----------------+
|  1|  2|  3|read-committed|
+------+------+------+----------------+
1 row in set (0.01 sec)
```

6.10.3.10. SHOW

描述

`show parameters` 用于显示系统变量。

`show errors` 用于显示当前对象的错误信息。

格式

```
show parameters;
show errors;
```

参数解释

无

示例

- 显示所有系统变量。

```
show parameters;
```

6.10.3.11. SHOW RECYCLEBIN

描述

该语句用于查看回收站中的内容。

格式

```
SHOW RECYCLEBIN;
```

参数解释

无。

示例

- 查看回收站中的内容。

```
OceanBase(admin@test)> create table t1(c1 int);
Query OK, 0 rows affected (0.24 sec)

OceanBase(admin@test)> drop table t1;
Query OK, 0 rows affected (0.07 sec)

OceanBase(admin@test)> show recyclebin;
+-----------------------------------------+--------------+-------+---------------------------+
| OBJECT_NAME                             | ORIGINAL_NAME | TYPE | CREATETIME                |
+-----------------------------------------+--------------+-------+---------------------------+
| __recycle_$_1_1099511628776_1099511677777 | t1           | TABLE | 2017-10-20 17:27:40.881506 |
+-----------------------------------------+--------------+-------+---------------------------+
1 row in set (0.02 sec)
```

6.10.3.12. SHRINK

描述

该语句会将表的数据重整，即在合并的过程中重新写一遍。

格式

```
ALTER TABLE table_name SHRINK SPACE
```

参数解释

参数	描述
table_name	表名

示例

- 将表 t1 的数据重整

```
OceanBase(admin@test)>ALTER TABLE t1 SHRINK SPACE;
```

6.10.3.13. TRANSACTION

描述

该语句用于开启事务。

数据库事务（Database Transaction）是指作为单个逻辑工作单元执行的一系列操作。事务处理可以用来维护数据库的完整性，保证成批的 SQL 操作全部执行或全部不执行。

显示事务是用户自定义或用户指定的事务。通过 BEGIN TRANSACTION，或 BEGIN 和 BEGIN WORK（被作为 START TRANSACTION 的别名受到支持）语句显示开始，以 COMMIT 或 ROLLBACK 语句显示结束。

格式

```
transaction_stmt:
    START TRANSACTION [READ ONLY | READ WRITE];
  | BEGIN [WORK];
  | COMMIT [WORK];
  | ROLLBACK [WORK];
  | SET TRANSACTION {READ ONLY | READ WRITE};
```

参数解释

参数	描述
START TRANSACTION [READ ONLY | READ WRITE]	开启事务语句。一旦开启事务，则随后的 SQL 数据操作语句（即INSERT、UPDATE、DELETE等）直到显式提交时才会生效。 READ ONLY 子句表示事务是只读方式开启，事务内不允许执行修改操作。 READ WRITE 子句表示事务是读写方式开启，默认就是此种模式。
BEGIN	BEGIN 和 BEGIN WORK 被作为 START TRANSACTION 的别名受到支持
COMMIT	提交当前事务。
ROLLBACK	回滚当前事务。

参数	描述
SET TRANSACTION {READ ONLY | READ WRITE}	将当前事务设置成 READ ONLY 或者 READ WRITE 模式。

示例

假设现有表 a 如下所示。

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43
2	b	200	2013-06-21 13:07:21
3	a	50	2013-06-21 13:08:15

1. 依次执行以下命令开始执行事务，将 id 为 3 的的 name 改为 c，并插入一行当前卖出 a 的记录。

```
OceanBase(admin@test)> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

OceanBase(admin@test)> UPDATE a SET name = 'c' WHERE id = 3;
Query OK, 1 rows affected (0.00 sec)

OceanBase(admin@test)> INSERT INTO a VALUES (4, 'a', 30, '2013-06-21 16:09:13');
Query OK, 1 rows affected (0.00 sec)

OceanBase(admin@test)> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

2. 事务提交后，执行命令查看表 a 信息。

```
SELECT * FROM a;
```

结果如下：

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43

2	b	200	2013-06-21 13:07:21
3	c	50	2013-06-21 13:08:15
4	a	30	2013-06-21 16:09:13

注意

在事务还没有 COMMIT 之前，您可以查看下本事务中的操作是否已经生效，比如可以在 COMMIT 前，加一句 `SELECT * FROM a;` 。本事务 session 的访问能读到最新的结果，本事务 session 之外的访问结果肯定是没有生效，在事务还没有 COMMIT 前，你之前做的操作除当前事务连接之外都是不可见的。如果您想回滚该事务，直接用 `ROOLBACK` 代替 `COMMIT` 。