云数据库 PostgreSQL
内核能力介绍

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

内核能力介绍
- 内核版本概述
- 内核版本更新动态
- 查看内核版本
- 自研内核功能介绍
  - 逻辑复制槽故障转移（Failover Slot）
  - TDE 透明数据加密
  - 异步 invalid buffer
  - 进程监控
  - SQL 限流
  - 使用 starocks_fdw 实现读写分离
  - 使用 tencentdb_ai 插件调用大模型
  - 开启 syscache/relcache 自动缓存淘汰机制

版权所有：腾讯云计算（北京）有限责任公司

内核能力介绍
内核版本概述

最近更新时间：2025-03-10 19:17:02

本文为您介绍云数据库 PostgreSQL 内核版本的背景信息和优势。

背景信息

PostgreSQL（简称 PG）是一款全球流行的企业级开源数据库，2017、2018连续两年蝉联 DB-Engine 年度数据库，2019年荣获 OSCON（The O'Reilly Open Source Convention）开源软件终身成就奖，2020年再次获得年度数据库冠军。
腾讯云数据库 PostgreSQL 是经过腾讯多年的业务沉淀和积累，在社区版本的 PostgreSQL 之上，根据不同的业务特征加入了特色功能和优化的内核版本。其不仅对 PostgreSQL 数据库引擎、权限管理、复制性能等方面进行了大量优化，同时提升了在云上的易用性和可维护性。

内核版本号

核版本号由两部分组成：**社区版本号**和**内部版本号**。

```
v{社区主版本号.社区次版本号}_r{不兼容版本号.兼容版本号}
如：v12.7_r1.1
```

v 代表 Version，r 代表 Release。如：v12.7_r1.1对应的社区版本是12.7，r1.1是代表腾讯云迭代版本，由 `r {不兼容版本号}.{兼容版本号}` 这两部分构成。

说明：

- Version：PostgreSQL 社区版本号，与社区对应。包含有两个数字，分别是 Major Version、Minor Version；Major Version 从10开始（9.3、9.4、9.5、9.6按照旧递进关系），Minor Version 从1开始。当社区次要版本 released 后，腾讯云会定时跟随社区版本进行版本更新。
- Release：云数据库 PostgreSQL 迭代用的版本号，表达云数据库对 PostgreSQL 所做的修改。包含有两个数字，分别是 Incompatible Version 和 Compatible Version；Incompatible Version 从1开始，Compatible Version 从0开始。
  - 不兼容版本号
    不兼容定义：用新的数据库内核程序启动数据库实例，若启动失败或功能无法正常使用，该修改就是不兼容的。
    当内核支持了不兼容的功能，需要将不兼容版本号增加1。不兼容版本号将用于判断能否直接进行二进制小版本升级。

- 兼容性版本号
  增加兼容性的修改时，需要将兼容版本号增加1。

版本支持列表

云数据库 PostgreSQL 支持的版本请参考 内核版本更新动态 。您可通过 PostgreSQL 控制台 实例列表，查看云数据库 PostgreSQL 的数据库版本。版本生命周期请参考 大版本生命周期说明 。

版本规则

- Version + Release 一起组成完整版本，称为数据库内核版本。
- Major Version 对应一个社区演化分支，各自迭代，如10、11、12等，分别拥有自己的版本号序列。
- 当社区宣布 Major Version 不再维护时，相应版本将停止新购。但腾讯云会一直维护此版本的数据库实例，直到用户实例主动销毁或迁移。
- Minor Version 跟随社区的变更，Release 根据腾讯云数据库内核所做功能的变更而变更。

优势

相比开源版本 PostgreSQL 具有如下特点：

- 更强的稳定性
  云数据库 PostgreSQL 在开源 PostgreSQL 的基础上，优化了部分数据库底层实现，使数据库因外界影响受到崩溃的问题极大减少，提升了数据库的稳定性。
- 主从同步性能增强
  社区版 PostgreSQL 在大量 DDL 处理时，主从同步性能变得极其差，云数据库 PostgreSQL 在此方面进行极大优化，整体提升高达3万倍。
- 更短的访问延迟
  云数据库 PostgreSQL 在大量短连接场景进行了建连优化，使数据库连接性能消耗减少80%，极大提升了短连接场景的性能体验。

内核版本更新动态

最近更新时间：2025-05-22 11:29:52

本文为您介绍云数据库 PostgreSQL 内核版本更新动态。如需升级内核小版本，请参考 升级内核小版本；如需升级大版本，请参考 升级数据库大版本 。

PostgreSQL 17

v17.4_r1.4

- 原生逻辑复制支持使用虚拟地址进行跨实例访问；
- 支持 plv8 插件；
- 允许普通用户创建可信插件；
- 新增若干内核安全防护机制。

v17.4_r1.3

- 支持进程级别 syscache/relcache 缓存淘汰功能；
- 支持 starrocks_fdw 插件；
- tencentdb_superuser 支持使用 replication origin 相关函数；
- 修复若干已知 bug；
- 支持 pglogical 插件；
- 更新至社区最新小版本。

v17.0_r1.2

支持 tencentdb_ai 插件。

v17.0_r1.1

- 内核新增安全防护机制；
- 支持 timescaledb。

v17.0_r1.0

兼容社区版本17.0。

PostgreSQL 16

v16.8_r1.10

- 原生逻辑复制支持使用虚拟地址进行跨实例访问；

- 支持 plv8 插件；
- 允许普通用户创建可信插件；
- 新增若干内核安全防护机制。

v16.8_r1.9

- 支持进程级别 syscache/relcache 缓存淘汰功能；
- 支持 starrocks_fdw 插件；
- tencentdb_superuser 支持使用 replication origin 相关函数；
- 修复若干已知 bug；
- 更新至社区最新小版本。

v16.4_r1.8

内核新增安全防护机制；

v16.4_r1.7

- 修复社区 CVE-2024-10979;
- 支持 pg_jieba 插件；
- 优化 pg_squeeze 部分问题；
- 支持 smlar 插件；
- 支持 pldebugger 插件；
- 支持 pgsphere 插件；
- 支持 pglogical 插件；
- 升级 pgvector 至0.7.3版本；
- 支持 pg_partman 插件；
- 升级社区小版本至16.4版本。

v16.2_r1.6

cos_fdw 增加安全鉴权功能，限制访问同账户购买的 COS。

v16.2_r1.5

- 修复审计极速版影响行数一直显示为0的问题；
- 修复审计精细版扩展协议下 SELECT 语句无法审计的问题；
- 支持 SQL 限流插件 tencentdb_sql_throttling;
- 修复社区 PG 逻辑复制 BUG：大量 update 导致逻辑复制分析 wal 日志速度慢；

v16.2_r1.4

- 修复 zhparser 权限问题；
- 修复 vacuum 陷入死循环问题。

v16.2_r1.3

- 兼容社区版本16.2；
- 支持 database 级别 CPU 隔离。

v16.0_r1.2

- 修复审计日志长连接导致的内存泄露；
- 修复 dblink 非法参数导致 core；
- 出于安全考虑，普通日志记录 COPY 语句；
- tencentdb_superuser 支持创建 replication origin 权限；
- 修复 cos_fdw 进行大分区表扫描时的超时报错问题。

v16.0_r1.1

修复 CVE-2023-5869。

v16.0_r1.0

兼容社区16.0版本。

PostgreSQL 15

v15.12_r1.16

- 原生逻辑复制支持使用虚拟地址进行跨实例访问；
- 支持 plv8 插件；
- 允许普通用户创建可信插件；
- 新增若干内核安全防护机制。

v15.12_r1.15

- 支持进程级别 syscache/relcache 缓存淘汰功能；
- 支持 starrocks_fdw 插件；
- tencentdb_superuser 支持使用 replication origin 相关函数；
- 修复若干已知 bug；
- 更新至社区最新小版本。

v15.8_r1.14

内核新增安全防护机制。

v15.8_r1.13

- 修复社区 CVE-2024-10979;
- 支持 pg_jieba 插件;
- 优化 pg_squeeze 部分问题;
- 支持 smlar 插件;
- 支持 pldebugger 插件;
- 支持 pgsphere 插件;
- 支持 pglogical 插件;
- 升级 pgvector 至0.7.3版本;
- 支持 pg_partman 插件;
- 升级社区小版本至15.8版本。

v15.6_r1.12

cos_fdw 增加安全鉴权功能，限制访问同账户购买的 COS。

v15.6_r1.11

- 修复 zhparser 权限问题;
- 修复审计极速版影响行数一直显示为0的问题;
- 修复审计精细版扩展协议下 SELECT 语句无法审计的问题;
- 支持 sql 限流插件 tencentdb_sql_throttling;
- 支持 pldebugger 插件。

v15.6_r1.10

- 兼容社区版本15.6;
- 支持 database 级别 CPU 隔离。

v15.1_r1.9

- 修复审计日志长连接导致的内存泄露;
- 修复 dblink 非法参数导致 core;
- 出于安全考虑，普通日志记录 COPY 语句;
- tencentdb_superuser 支持创建 replication origin 权限;
- 修复 cos_fdw 进行大分区表扫描时的超时报错问题。

v15.1_r1.8

修复 CVE-2023-5869。

版权所有：腾讯云计算（北京）有限责任公司

v15.1_r1.7

- 审计日志性能优化。
- 修复 failover slot 慢于备机 checkpoint 位点时，从备机发起备份失败的问题。

v15.1_r1.6

- 支持大版本升级。
- 更新部分插件版本。

v15.1_r1.5

- 兼容社区15.1版本。
- 修复 CVE-2023-39417 安全漏洞。
- 支持 tencentdb_system_stat 进程监控插件。
- 支持 pg_squeeze、pg_similarity 插件。

v15.1_r1.4

- 兼容社区15.1版本。
- 支持 dblink 跨实例访问。
- rds_server_handler 插件支持使用命令清空所有 cache。
- 支持数据库全审计。
- 禁止 COPY PROGRAM 行为。
- 支持 tencentdb_superuser 插件。

v15.1_r1.3

- 兼容社区15.1版本。
- 支持 pgvector 插件。

v15.1_r1.2

- 兼容社区15.1版本。
- 对部分插件在创建和升级时可能造成的安全问题进行加固。

v15.1_r1.0

- 兼容社区15.1版本。
- 新增 tencentdb_superuser 权限，允许创建 event_trigger。

PostgreSQL 14

v14.17_r1.30

- 原生逻辑复制支持使用虚拟地址进行跨实例访问；
- 支持 plv8 插件；
- 允许普通用户创建可信插件；
- 新增若干内核安全防护机制。

v14.17_r1.29

- 支持进程级别 syscache/relcache 缓存淘汰功能；
- 支持 starrocks_fdw 插件；
- tencentdb_superuser 支持使用 replication origin 相关函数；
- 修复若干已知 bug；
- 更新至社区最新小版本。

v14.13_r1.28

内核新增安全防护机制。

v14.13_r1.26

- 修复社区 CVE-2024-10979;
- 支持 pg_jieba 插件；
- 优化 pg_squeeze 部分问题；
- 支持 smlar 插件；
- 支持 pldebugger 插件；
- 支持 pgsphere 插件；
- 支持 pglogical 插件；
- 升级 pgvector 至0.7.3版本；
- 支持 pg_partman 插件；
- 升级社区小版本至14.13版本。

v14.11_r1.24

修复 plv8 安全漏洞。

v14.11_r1.23

cos_fdw 增加安全鉴权功能，限制访问同账户购买的 COS。

v14.11_r1.22

- 修复 zhparser 权限问题；
- 修复审计极速版影响行数一直显示为0的问题；
- 修复审计精细版扩展协议下 SELECT 语句无法审计的问题；
- 支持 sql 限流插件 tencentdb_sql_throttling。

v14.11_r1.21

- 兼容社区14.11版本；
- 支持数据库对象 database 级别 CPU 资源隔离。

v14.2_r1.18

- 修复审计日志长连接导致的内存泄露；
- 修复 dblink 非法参数导致 core；
- 出于安全考虑，普通日志记录 COPY 语句；
- tencentdb_superuser 支持创建 replication origin 权限；
- 修复 cos_fdw 进行大分区表扫描时的超时报错问题。
- 社区 BUG 修复：备机 promote 后时间线回切导致数据损坏。

v14.2_r1.16

修复 CVE-2023-5869。

v14.2_r1.15

审计日志性能优化；

v14.2_r1.14

- 支持大版本升级；
- 更新部分插件版本。

v14.2_r1.12

- 兼容社区14.2版本。
- 修复 CVE-2023-39417 安全漏洞。
- 支持 tencentdb_system_stat 进程监控插件。
- 支持 pg_squeeze、pg_similarity 插件。

v14.2_r1.12

- 修复 CVE-2023-39417 安全漏洞；
- 支持 tencentdb_system_stat 进程监控插件；

- 支持 pg_squeeze、pg_similarity 插件；

v14.2_r1.11

- 支持 dblink 跨实例访问；
- rds_server_handler 插件支持使用命令清空所有 cache；
- 支持审计日志；
- 禁止 COPY PROGRAM 行为；
- 支持 pgvector 插件；
- 支持 tencentdb_superuser 插件。

v14.2_r1.9

修复创建插件时提权漏洞：创建插件时 tencentdb_superuser 不再提权成 superuser。

v14.2_r1.7

增加 GUC 控制用户 COPY 行为，禁止创建插件时使用 COPY 行为。

v14.2_r1.4

- 支持 failover slot；
- tencentdb_superuser 支持创建/修改 event_trigger；
- 逻辑复制解析 drop schema 失败时抛出表 oid 信息；
- 将 session_replication_role 参数级别改为 PGC_USERSET；
- 修复 dblink 同实例跨库访问权限问题。

v14.2_r1.3

支持 tencentdb_pwdcheck 强密码认证插件。

v14.2_r1.2

- 兼容社区14.2版本。
- 支持 dblink 插件。

v14.2_r1.1

- 兼容社区14.2版本。
- 支持 cos_fdw 插件。

v14.2_r1.0

- 兼容社区14.2版本。

- 合入 TencentDB PostgreSQL 内核改动能力。
- 支持 pg_cron 插件。
- 支持 TopN 插件。
- 支持 TDE 数据透明加密。

PostgreSQL 14 babelfish 版

v14.3_r1.27

修复社区 CVE-2024-10979。

v14.3_r1.25

修复 plv8 安全漏洞。

v14.3_r1.19

社区 BUG 修复：备机 promote 后时间线回切导致数据损坏。

v14.3_r1.12

修复 CVE-2023-39417 安全漏洞。

v14.3_r1.10

修复创建插件时提权漏洞：创建插件时 tencentdb_superuser 不再提权成 superuser。

v14.3_r1.8

增加 GUC 控制用户 COPY 行为，禁止创建插件时使用 COPY 行为。

v14.3_r1.6

支持将 NUL 转义成自定义字符。

v14.3_r1.3

- 支持 pg_basebackup 双流合并和过滤文件；
- 支持 tencentdb_superuser;
- 支持 soft_limit_connection 软连接限制；
- 支持 qps 统计插件；
- 禁止 superuser 执行 owner 为普通 user 的函数；
- 支持三方插件；
- 禁用 dblink；
- 修复备机异步 invalid buffer 的 bug；

版权所有：腾讯云计算（北京）有限责任公司

- 支持 topn 和 pg_cron 插件；
- 使用 tencentdb_superuser CREATE/ALTER superuser 时，会默认 CREATE/ALTER 一个 tencentdb_superuser；
- 支持 TDE；
- 支持服务端 basebackup 时过滤文件（夹）；
- 支持 cos_fdw 冷热存储分离插件；
- 修复 pg_cron 安全问题；
- 支持 dblink 同实例跨库访问；
- 允许 superuser 执行所有 oid<16384 的函数；
- 修复社区 CIC/RIC 索引数据丢失的问题；
- 支持 babelfish 初始内核及 2.1 版本插件代码。

PostgreSQL 13

v13.20_r1.22

- 原生逻辑复制支持使用虚拟地址进行跨实例访问；
- 允许普通用户创建可信插件；
- 新增若干内核安全防护机制。

v13.20_r1.21

- 支持进程级别 syscache/relcache 缓存淘汰功能；
- 支持 starrocks_fdw 插件；
- tencentdb_superuser 支持使用 replication origin 相关函数；
- 修复若干已知 bug；
- 更新至社区最新小版本。

v13.16_r1.20

内核新增安全防护机制。

v13.16_r1.19

- 修复社区 CVE-2024-10979;
- 支持 pg_jieba 插件；
- 优化 pg_squeeze 死锁问题；
- 支持 smlar 插件；
- 支持 pldebugger 插件；
- 支持 pgsphere 插件；

- 支持 pg_pathman 插件；
- 支持 pglogical 插件；
- 升级 pgvector 至0.7.3版本；
- 支持 pg_partman 插件；
- 升级社区小版本至13.16版本。

v13.14_r1.18

修复 plv8 安全漏洞。

v13.14_r1.17

cos_fdw 增加安全鉴权功能，限制访问同账户购买的 COS。

v13.14_r1.16

- 修复审计极速版影响行数一直显示为0的问题；
- 支持 sequential_uuids 插件；
- 修复审计精细版扩展协议下 SELECT 语句无法审计的问题；
- 支持 starrocks_fdw 插件。

v13.14_r1.15

兼容社区版本13.14；

v13.3_r1.14

- 修复审计日志长连接导致的内存泄露；
- 修复 dblink 非法参数导致 core；
- 出于安全考虑，普通日志记录 COPY 语句；
- tencentdb_superuser 支持创建 replication origin 权限；
- 修复 cos_fdw 进行大分区表扫描时的超时报错问题。
- 社区 BUG 修复：备机 promote 后时间线回切导致数据损坏。

v13.3_r1.13

修复 CVE-2023-5869。

v13.3_r1.12

审计日志性能优化。

v13.3_r1.11

- 支持大版本升级；
- 更新部分插件版本。

v13.3_r1.10

- 兼容社区13.3版本。
- 修复 CVE-2023-39417 安全漏洞。
- 支持 tencentdb_system_stat 进程监控插件。
- 支持 pg_squeeze、pg_similarity 插件。

v13.3_r1.9

- 兼容社区13.3版本。
- 支持 dblink 跨实例访问。
- rds_server_handler 插件支持使用命令清空所有 cache。
- 支持数据库全审计。
- 禁止 COPY PROGRAM 行为。
- 支持 tencentdb_superuser 插件。

v13.3_r1.8

- 兼容社区13.3版本。
- 对部分插件在创建和升级时可能造成的安全问题进行加固。

v13.3_r1.6

- 兼容社区13.3版本。
- 新增 tencentdb_failover_slot 插件，支持 failover slot 功能。
- 新增 tencentdb_superuser 权限，允许使用 checkpoint 命令。
- 新增 tencentdb_superuser 权限，允许创建 event_trigger。
- 支持使用 dblink 进行同实例跨库访问。
- 优化 walsender 解析 drop publication 语句性能。

v13.3_r1.5

- 兼容社区13.3版本。
- 支持 tencentdb_pwdcheck 强密码认证插件。

v13.3_r1.4

- 兼容社区13.3版本。
- 支持 db_link 插件。

版权所有：腾讯云计算（北京）有限责任公司

v13.3_r1.3

- 兼容社区13.3版本。
- 支持 cos_fdw 插件。

v13.3_r1.2

- 兼容社区13.3版本。
- 支持 pg_cron 插件。
- 支持 TopN 插件。
- 支持 TDE 数据透明加密。

v13.3_r1.1

- 兼容社区13.3版本。
- 修复了一些插件管理方面的 BUG。

v13.3_r1.0

- 兼容社区13.3版本。
- 合入社区 PostgreSQL 13.3 版本的所有功能。
- 新增管理员角色 pg_tencentdb_superuser。
- 合入腾讯云关于 PostgreSQL 历史优化功能包。

PostgreSQL 12

v12.22_r1.27

新增若干内核安全防护机制。

v12.22_r1.26

- 修复若干已知 bug；
- 更新至社区最新小版本。

v12.20_r1.25

内核新增安全防护机制。

v12.20_r1.24

- 修复社区 CVE-2024-10979;
- 支持 pg_jieba 插件；
- 优化 pg_squeeze 部分问题；

- 支持 smlar 插件；
- 支持 pldebugger 插件；
- 支持 pgsphere 插件；
- 支持 pg_pathman 插件；
- 支持 pglogical 插件；
- 升级 pgvector 至 0.7.3 版本；
- 支持 pg_partman 插件；
- 升级社区小版本至 12.20 版本。

v12.20_r1.23

修复 plv8 安全漏洞。

v12.18_r1.22

cos_fdw 增加安全鉴权功能，限制访问同账户购买的 COS。

v12.18_r1.21

修复审计精细版扩展协议下 SELECT 语句无法审计的问题。

v12.18_r1.20

- 修复 zhparser 权限问题；
- 修复审计极速版影响行数一直显示为 0 的问题；
- 支持 sequential_uuids 插件。

v12.18_r1.19

更新社区小版本至 12.18。

v12.7_r1.18

- 修复审计日志长连接导致的内存泄露；
- 修复 dblink 非法参数导致 core；
- 出于安全考虑，普通日志记录 COPY 语句；
- tencentdb_superuser 支持创建 replication origin 权限；
- 修复 cos_fdw 进行大分区表扫描时的超时报错问题。
- 社区 BUG 修复：备机 promote 后时间线回切导致数据损坏。

v12.7_r1.17

支持 orafce 插件。

v12.7_r1.16

修复 CVE-2023-5869。

v12.7_r1.15

审计日志性能优化。

v12.7_r1.14

- 支持大版本升级；
- 更新部分插件版本。

v12.7_r1.13

- 兼容社区12.7版本。
- 修复 CVE-2023-39417 安全漏洞。
- 支持 tencentdb_system_stat 进程监控插件。
- 支持 pg_squeeze、pg_similarity 插件。

v12.7_r1.12

- 兼容社区12.7版本。
- 支持1.5.2版本 pg_squeeze 插件。
- 升级 zhparser 插件到2.2版本。

v12.7_r1.10

- 兼容社区12.7版本。
- 对部分插件在创建和升级时可能造成的安全问题进行加固。

v12.7_r1.8

- 兼容社区12.7版本。
- 新增 tencentdb_failover_slot 插件，支持 failover slot 功能。
- 新增 tencentdb_superuser 权限，允许使用 checkpoint 命令。
- 新增 tencentdb_superuser 权限，允许创建 event_trigger。
- 支持使用 dblink 进行同实例跨库访问。
- 优化 walsender 解析 drop publication 语句性能。

v12.7_r1.7

- 兼容社区12.7版本。
- 支持 tencentdb_pwdcheck 强密码认证插件。

版权所有：腾讯云计算（北京）有限责任公司

v12.7_r1.6

- 兼容社区12.7版本。
- 支持 timescaledb 插件1.7.5版本和2.6.0版本。

v12.7_r1.5

- 兼容社区12.7版本。
- 支持 dblink 插件。

v12.7_r1.3

- 兼容社区12.7版本。
- 支持 cos_fdw 插件。

v12.7_r1.2

- 兼容社区12.7版本。
- 支持 pg_cron 插件。
- 支持 TopN 插件。
- 支持 TDE 数据透明加密。

v12.7_r1.1

- 兼容社区12.7版本。
- 合入社区 PostgreSQL 12.7 版本的所有功能。

v12.4_r1.0

- 兼容社区12.4版本。
- 新增管理员角色 pg_tencentdb_superuser。
- 合入腾讯云关于 PostgreSQL 历史优化功能包。

PostgreSQL 11

v11.22_r1.24

新增若干内核安全防护机制。

v11.22_r1.23

修复若干已知 bug。

v11.22_r1.22

内核新增安全防护机制。

v11.22_r1.21

- 修复社区 CVE-2024-10979;
- 支持 pg_jieba 插件;
- 优化 pg_squeeze 部分问题;
- 支持 smlar 插件;
- 支持 pldebugger 插件;
- 支持 pgsphere 插件;
- 支持 pg_pathman 插件;
- 支持 pglogical 插件;
- 修复 cos_fdw 安全问题。

v11.22_r1.20

修复 plv8 安全漏洞。

v11.22_r1.19

修复审计日志相关问题。

v11.22_r1.18

新增 sequential_uuids 插件

v11.22_r1.17

兼容社区11.22版本。

v11.12_r1.16

社区 BUG 修复：备机 promote 后时间线回切导致数据损坏。

v11.12_r1.15

修复社区 CVE-2023-5869。

v11.12_r1.14

审计日志性能优化。

v11.12_r1.13

- 支持大版本升级;
- 更新部分插件版本。

v11.12_r1.12

- 兼容社区11.12版本。
- 修复 CVE-2023-39417 安全漏洞。
- 支持 tencentdb_system_stat 进程监控插件。
- 支持 pg_squeeze、pg_similarity 插件。

v11.12_r1.11

- 兼容社区11.12版本。
- 支持 dblink 跨实例访问。
- rds_server_handler 插件支持使用命令清空所有 cache。
- 支持数据库全审计。
- 禁止 COPY PROGRAM 行为。
- 支持 tencentdb_superuser 插件。

v11.12_r1.9

- 兼容社区11.12版本。
- 对部分插件在创建和升级时可能造成的安全问题进行加固。

v11.12_r1.7

- 兼容社区11.12版本。
- 新增 tencentdb_failover_slot 插件，支持 failover slot 功能。
- 新增 tencentdb_superuser 权限，允许使用 checkpoint 命令。
- 新增 tencentdb_superuser 权限，允许创建 event_trigger。
- 支持使用 dblink 进行同实例跨库访问。
- 优化 walsender 解析 drop publication 语句性能。

v11.12_r1.6

- 兼容社区11.12版本。
- 支持 tencentdb_pwdcheck 强密码认证插件。

v11.12_r1.5

- 兼容社区11.12版本。
- 支持 dblink 插件。

v11.12_r1.3

- 兼容社区11.12版本。
- 支持 cos_fdw 插件。

v11.12_r1.2

- 兼容社区11.12版本。
- 支持 pg_cron 插件。
- 支持 TopN 插件。
- 支持 TDE 数据透明加密。

v11.12_r1.1

兼容社区11.12版本。

v11.8_r1.0

- 兼容社区11.8版本。
- 合入社区 PostgreSQL 11.8 版本的所有功能。
- 新增管理员角色 pg_tencentdb_superuser。
- 合入腾讯云关于 PostgreSQL 历史优化功能包。

PostgreSQL 10

v10.23_r1.21

支持 postgis 3.0.11 版本。

v10.23_r1.20

内核新增安全防护机制。

v10.23_r1.19

修复社区 CVE-2024-10979。

v10.23_r1.18

关闭对 plv8 插件的支持。

v10.23_r1.17

支持 sequential_uuids 插件。

v10.23_r1.16

兼容社区10.23版本。

v10.17_r1.15

- 兼容社区10.17版本。
- 社区 BUG 修复：备机 promote 后时间线回切导致数据损坏。

v10.17_r1.14

- 兼容社区10.17版本。
- 修复社区 CVE-2023-5869。

v10.17_r1.13

- 兼容社区10.17版本。
- 大版本升级内核适配；
- pgrouting 插件支持3.2.2版本。

v10.17_r1.12

- 兼容社区10.17版本。
- 修复 CVE-2023-39417 安全漏洞。

v10.17_r1.11

- 兼容社区10.17版本。
- 修复创建插件时提权漏洞：创建插件时 tencentdb_superuser 不再提权成 superuser。

v10.17_r1.10

- 兼容社区10.17版本；
- 增加 GUC 控制用户 COPY 行为，禁止创建插件时使用 COPY 行为。

v10.17_r1.9

- 兼容社区10.17版本；
- 支持 dblink 跨实例访问；
- rds_server_handler 插件支持使用命令清空所有 cache。

v10.17_r1.8

- 兼容社区10.17版本；
- 支持 failover slot；
- tencentdb_superuser 支持创建/修改 event_trigger；
- 针对多对象逻辑复制 drop publication 语句解码做性能优化；
- 将 session_replication_role 参数级别改为 PGC_USERSET。

版权所有：腾讯云计算（北京）有限责任公司

v10.17_r1.7

- 兼容社区10.17版本。
- 支持 tencentdb_pwdcheck 强密码认证插件。

v10.17_r1.6

- 兼容社区10.17版本；
- 修复 tencentdb_superuser 使用 dblink 的问题。

v10.17_r1.5

- 兼容社区10.17版本；
- 修复 pg_cron 安全问题；
- 支持 dblink 同实例跨库访问。

v10.17_r1.4

- 兼容社区10.17版本。
- 支持 cos_fdw 插件。

v10.17_r1.3

- 兼容社区10.17版本。
- 支持 pg_cron 插件。
- 支持 TopN 插件。
- 支持 TDE 数据透明加密。

v10.17_r1.2

- 兼容社区10.17版本。
- 禁用 dblink；
- 修复备机异步 invalid buffer 的 bug。

v10.17_r1.1

- 兼容社区10.17版本。
- 支持 pg_basebackup 双流合并和过滤文件；
- 支持 tencentdb_superuser;
- 支持备机异步 invalid buffer；
- 支持 soft_limit_connection 软连接限制；
- 支持 qps 统计插件；

版权所有：腾讯云计算（北京）有限责任公司

- 支持三方插件。

v10.4_r1.1

- 兼容社区10.4版本。
- 修复了一些插件管理方面的 BUG。

v10.4_r1.0

- 兼容社区10.4版本。
- 新增管理员角色 pg_tencentdb_superuser。
- 合入腾讯云关于 PostgreSQL 历史优化功能包。

版权所有：腾讯云计算（北京）有限责任公司

查看内核版本

最近更新时间：2023-05-17 16:00:04

本文为您介绍如何查看云数据库 PostgreSQL 内核版本信息。

操作步骤

通过登录到 PostgreSQL 实例查看内核版本

1. 您可通过云服务器，连接到 PostgreSQL 实例查看内核版本，连接登录实例方式请参见 连接 PostgreSQL 实例 。
2. 登录实例后执行如下命令查看内核版本。

```
show tencentdb_version;
```

> **注意**
> 部分存量实例命令执行失败，或者输出结果为v1，代表 Release 版本为 r1.0，如数据库版本为 PostgreSQL 10.4，则内核版本为 v10.4_r1.0。

通过 PostgreSQL 控制台查看内核版本

1. 登录 PostgreSQL 控制台 ，在实例列表，单击实例 ID 或**操作**列的**管理**，进入实例管理页面。
2. 在实例详情页的**配置信息**下，可查看内核版本。

自研内核功能介绍
逻辑复制槽故障转移（Failover Slot）

最近更新时间：2024-05-10 09:50:52

功能背景

当前，PostgreSQL 用户在使用逻辑复制的时候会创建 logical replication slot，该信息没有记录在 catalog 中，因此不会将其信息从主库同步到备库。如果发生 HA 切换，逻辑复制的 Slot 会丢失，导致逻辑复制断开，给使用带来不便。因此我们需要将逻辑复制槽进行故障转移。

功能介绍

Failover Slot 功能可以将 Slot 信息从主库同步到备库，如此可以在 HA 切换之后避免逻辑订阅断开，用户可以无感知地继续使用逻辑订阅。我们提供了一个插件 tencent_failover_slot，用户使用该插件可以完成 Failover Slot 的创建、转换为普通 Slot、查看 Failover Slot 信息、删除 Failover Slot 等操作。同时，我们新增了 GUC 参数方便客户针对异常的情况的设置。下面详细介绍。

创建 Failover Slot

```
create extension tencentdb_failover_slot;--首先创建插件
select pg_create_logical_failover_slot('slotname','pluginname');--创建
Failover Slot
```

注意：

- Slot 是实例级别的，而插件是 database 级别的，如果切换了 database 需要调用插件里的函数，需要重新创建 tencentdb_failover_slot 插件。
- pg_create_logical_failover_slot 函数包含两个参数：
  - slotname：Failover Slot 的名称。
  - pluginname：表示 plugin 的名称。

将 Failover Slot 转换为普通 Slot

```
select transform_slot_to_nonfailover('slotname');
```

说明：

- slotname：slot 名字。

- 需要保证该 slot 当前是 inactive 的，即没有发布订阅正在使用该 slot。

查看 Failover Slot 信息

```
postgres=# select * from pg_failover_slots;
     slot_name
----------------
     fs
     fsg
   (2 rows)
```

说明：
pg_failover_slots 视图包含一列，展示了当前所有的 Failover Slot 的名字，更多 Failover Slot 信息请查看 pg_replication_slots 视图。

删除 Failover Slot

```
select * from pg_drop_replication_slot('slotname');
```

说明：
slotname：slot 名称。

参数设置

新增参数 failover_slot_timeline_diverged_option，类型为 enum，方便用户针对异常情况的处理设置，设置路径为 云数据库 PostgreSQL 控制台 > **实例详情** > **参数设置**。该参数默认值为 error，可选值 error、rewind。当极端情况下会发生备机日志接收速度比逻辑复制慢并发生 HA 的情况，此时如果设置 error，代表逻辑复制将暂停，发布端和订阅端都能收到报错，等待用户来处理。如果设置的为 rewind，那么逻辑复制将从切换时间点开始。

注意事项

当前只支持逻辑复制槽（logical replication slot）故障转移，不支持物理复制槽（physical slot）故障转移。

TDE 透明数据加密

最近更新时间：2024-05-09 15:16:31

功能背景

随着数据安全保护的要求越来越严格，各国各地域的信息安全保护法都提到要求数据库进行加密存储。通过数据加密防止意外数据文件遗失导致的数据泄露问题。

功能介绍

云数据库 PostgreSQL 提供透明数据加密（Transparent Data Encryption，TDE）功能，透明加密指数据的加解密操作对用户透明，支持对数据文件进行实时 I/O 加密和解密，在数据写入磁盘前进行加密，从磁盘读入内存时进行解密，可满足静态数据加密的合规性要求。加密使用的密钥由密钥管理服务 KMS 产生和管理。
KMS 是腾讯云一项保护数据及密钥安全的密钥服务，服务涉及的各个流程均采用高安全性协议通信，保证服务高安全，提供分布式集群管理和热备份，保证服务高可靠和高可用。
KMS 采用的是两层密钥体系，涉及两类密钥，即用户主密钥（CMK）与数据密钥（DATA KEY）。用户主密钥用于加密数据密钥或密码、证书、配置文件等小包数据（最多4KB）。数据密钥用于加密业务数据。海量的业务数据在存储或通信过程中使用数据密钥以对称加密的方式加密，而数据密钥又通过用户主密钥采用非对称加密方式加密保护。通过两层密钥体系，确保数据文件加密。

支持版本

内核对应小版本等于或高于 v10.17_r1.2、v11.12_r1.2、v12.7_r1.2、v13.3_r1.2、v14.2_r1.0，支持加密功能。

适用场景

透明数据加密指数据的加解密操作对用户透明，支持对数据文件进行实时 I/O 加密和解密，在数据写入磁盘前进行加密，从磁盘读入内存时进行解密，可满足静态数据加密的合规性要求。

使用说明

开启透明数据加密功能与数据库透明加密的详细介绍，请参见 透明数据加密 。

异步 invalid buffer

最近更新时间：2025-05-15 17:02:02

背景介绍

PostgreSQL 主从复制的逻辑在具有大量数据表的 DDL 变更时会导致从库恢复数据极其缓慢。为了避免此类问题，TencentDB 针对此类场景进行了优化和修改。

原理介绍

PostgreSQL 是通过物理复制实现主从复制。日志同步到备机之后，备机会解析 wal 日志，来与主库保持数据一致。在 PostgreSQL 从库在恢复一条 drop 类语句时要做的操作如下：

1. 恢复系统表，例如 pg_class，pg_attribute，pg_type 等，相当于移除表的元信息。
2. close 表对应的文件。
3. 遍历 buffer 中的页面，如果缓存的是该表的页面，则标记为 invalid，后面其他进程可以使用该页面。
4. 发异步失效消息给其他 backend，通知该表已删除。
5. 删除表对应的物理文件。

但在 PostgreSQL 内核中第3步将会执行 DropRelFileNodesAllBuffers 函数。其函数需要遍历整个 shard_buffer，以查看 buffer 是否缓存有将要删除的表的数据，将其标记为失效。而 PostgreSQL 中页面大小默认为8K，以 shard_buffer 大小16GB为例，则一共有16GB/8K = 200W个 page，每删除一个表这里需要循环200万次，如果表上面有索引，每个索引也要循环200万次。
所以从业务上看，当存在大量表变更并且快速删除表操作的时候，由于主库可以并发执行所以感觉不出性能的影响，但是因为 PostgreSQL 的备库是单进程的 recovery，就会出现主备同步日志堆积，数据延迟的问题。

问题解决

腾讯云数据库 PostgreSQL 在 recover drop table 操作时，将表信息写入一个共享的 hash 表中。当存在表文件删除场景下，不再直接进行物理删除，将删除实际文件的动作作为异步动作存储其中。
当 invalid buffer 结束时将表从 hash 表中移除，这样，如果在此过程中发生打开文件失败，检查是否存在此 hash 表中即可。并且如果在新创建文件的时候也去遍历一下此队列，若队列中存在同名文件正在 invalid buffer，则等待即可。
而 PostgreSQL 关于表文件命名是一个 uint32 整数保存，采用的是“全局分配，局部存储”的方式，即一个实例下的所有数据库使用一个计数器生成文件号，生成的文件保存在各自库的目录下，分配时，如果当前库下已有同名文件，则尝试下一个，直到没有冲突为止，计数器绕圈后重新开始。
经过优化后，可以明显发现同类场景下主从同步性能增强了3W多倍。

进程监控

最近更新时间：2025-04-29 16:42:01

功能背景

在一些通用场景下，客户应用不仅需要数据库实例整体的监控信息，同时也需要通过实例中的进程监控信息来判断业务健康度，从而进一步做业务逻辑的决策。典型的，在 SaaS 场景中，用户的一个租户单独使用一个数据库实例的 database。业务方通过监控连接到 database 进程的资源使用情况，能进一步判断租户当前的健康度等情况。

功能介绍

云数据库 PostgreSQL 提供了数据库实例的进程监控信息。系统提供了一个视图 tencentdb_process_system_usage，该视图包含如下信息，说明如下：

字段名	说明
pid	进程号。
username	用户名。
datname	数据库名。
backend_type	进程类型。可能的类型为 autovacuum launcher, autovacuum worker, logical replication launcher, logical replication worker, parallel worker, background writer, client backend, checkpointer, startup, walreceiver, walsender 和 walwriter。除此以外，由扩展注册的后台 Worker 可能有额外的类型。
query	当前正在执行的查询语句。
cpu_usage	当前查询采样期间的 CPU 使用率，对于采样时间间隔，系统提供参数 tencentdb_system_stat.sampling_interval 可以设置，该参数的单位为毫秒，默认为1000。
memory_bytes	当前采样期间内存使用率，单位字节。

使用说明

1. 新建插件，具体语句如下：

```
postgres=> create extension tencentdb_system_stat;
CREATE EXTENSION
postgres=>
```

2. 视图 tencentdb_process_system_usage 的使用跟其他视图一样，只需要登录数据库 PostgreSQL 实例，进行查询即可，如下语句所示：

```
select * from tencentdb_process_system_usage;
```

SQL 限流

最近更新时间：2025-05-15 17:02:02

使用限制

注意：

- 重启和 HA 及大版本升级后，规则不在内存，将失效，需要重新加载。
- 不支持直接配置扩展协议（Extended Query Protocol）的 SQL 配置限流规则，更多扩展协议的说明请参考 官方文档 。
- SQL 限流功能当前仅支持内核版本在 v14.11_r1.23、v15.6_r1.12、v16.2_r1.6及以上的云数据库 PostgreSQL 实例，如您需要，请 升级小版本 或 升级大版本 。
- 当限流规则被加载到内存时才生效，新增规则时默认将规则持久化到表 tencentdb_sql_throttling.persistent_rules_table 中，重启或者 HA 后可重新将规则加载进内存。

创建插件

说明：
如您需要使用 SQL 限流功能，请您 提交工单 联系我们添加 shared_preload_libraries 参数。修改 shared_preload_libraries 参数会重启实例，请确保业务有重连机制。

检查 shared_preload_libraries 参数中已经加入 tencentdb_sql_throttling：

```
postgres=> show shared_preload_libraries;

shared_preload_libraries

------------------------------------------------------------------------
------------------------------
---------------------------------------------

pg_stat_statements,pg_stat_log,wal2json,decoderbufs,decoder_raw,pg_hint_
plan,rds_server_handler,tencen
tdb_pwdcheck,pgaudit,tencentdb_sql_throttling
(1 row)
```

创建 tencentdb_sql_throttling 插件，并检查创建完成：

```
postgres=> create extension tencentdb_sql_throttling;
CREATE EXTENSION
postgres=> select * from pg_extension where
extname='tencentdb_sql_throttling';
  oid  |          extname          | extowner | extnamespace |
extrelocatable | extversion | extconfig | extcondition
-------+---------------------------+----------+--------------+-----------
-----+------------+-----------+--------------
 16415 | tencentdb_sql_throttling |    16385 |         2200 | t
| 1.0        |           |
(1 row)
```

新增限流规则

云数据库 PostgreSQL 支持通过 query_id 和 SQL 来新增限流规则。其中 query_id 是内核将 query 语句规范化后哈希得到的64位整数，是一个用于标识查询的唯一标识符，更多 query_id 的详细说明请参考 官方文档 。

通过 query_id 新增限流规则

如您需要获取 SQL 语句的 query_id，您可通过查询 pg_stat_statements 或 pg_stat_activity 得到。本文是通过查询 pg_stat_activity 得到。

```
postgres=> select datname,username,client_addr,state,query_id,query from
pg_stat_activity where username='dbadmin' and client_addr='172.16.76.48'
limit 10;
 datname  | username |  client_addr  | state  |      query_id      |
query
----------+---------+---------------+--------+--------------------+-----
---------------------------
 postgres | dbadmin | 172.16.76.48 | active | 4979398629351213 43 |
SELECT id FROM t WHERE id = 1
 postgres | dbadmin | 172.16.76.48 | active | 4979398629351213 43 |
SELECT id FROM t WHERE id = 53
 postgres | dbadmin | 172.16.76.48 | active | 4979398629351213 43 |
SELECT id FROM t WHERE id = 54
 postgres | dbadmin | 172.16.76.48 | active | 4979398629351213 43 |
SELECT id FROM t WHERE id = 4
 postgres | dbadmin | 172.16.76.48 | active | 4979398629351213 43 |
SELECT id FROM t WHERE id = 52
 postgres | dbadmin | 172.16.76.48 | active | 4979398629351213 43 |
SELECT id FROM t WHERE id = 51
 postgres | dbadmin | 172.16.76.48 | active | 4979398629351213 43 |
```

```
SELECT id FROM t WHERE id = 7
 postgres | dbadmin | 172.16.76.48 | active | 497939862935121343 |
SELECT id FROM t WHERE id = 10
 postgres | dbadmin | 172.16.76.48 | active | 497939862935121343 |
SELECT id FROM t WHERE id = 11
 postgres | dbadmin | 172.16.76.48 | active | 497939862935121343 |
SELECT id FROM t WHERE id = 8
(10 rows)
```

在获取到 query_id 之后，我们可基于 query_id 新增限流规则。云数据库 PostgreSQL 提供了一个函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.add_rule_with_queryid(
    queryid bigint,
    work_node int,
    max_concurrency int,
    is_enabled boolean,
    memory_only boolean,
    query_string text
);
```

add_rule_with_queryid 函数的参数解释如下：

说明：

- queryid：将 query 语句规范化后哈希得到的64位整数，可通过 pg_stat_statements 或 pg_stat_activity 查询得到；
- work_node：限流规则生效的节点，0表示主实例和只读实例都生效，1表示只有主实例生效，2表示只有只读实例生效；
- max_concurrency：该限流规则允许的最大并发数量；
- is_enabled：当前规则是否生效，取值为 true 和 false；
- memory_only：可选，是否需要将规则持久化到表中，重启或者 HA 后可使用命令重新将规则加载进内存以生效；
- query_string：可选，可以将 query 语句加入规则中，此时不对此 query_string 计算 queryid 进行校验，该限流规则以 queryid 为准。若不提供此参数，内核将自动在 pg_stat_statements 视图中查找该 queryid 对应的 sql 语句，并将结果记录在该规则中。

- work_node：限流规则生效的节点，0表示主实例和只读实例都生效，1表示只有主实例生效，2表示只有只读实例生效；

- max_concurrency：该限流规则允许的最大并发数量；
- is_enabled：当前规则是否生效；
- memory_only：可选，是否需要将规则持久化到表中，重启或者 HA 后可使用命令重新将规则加载进内存以生效；
- query_string：可选，可以将 query 语句加入规则中，此时不对此 query_string 计算 queryid 进行校验，该限流规则以 queryid 为准。若不提供此参数，内核将自动在 pg_stat_statements 视图中查找该 queryid 对应的 sql 语句，并将结果记录在该规则中。SQL 限流示例如下 SQL 限流示例如下：

```
postgres=> SELECT
tencentdb_sql_throttling.add_rule_with_queryid(4979398629351213 43,0,10
,true);
 add_rule_with_queryid
----------------------

(1 row)
```

通过 SQL 语句新增限流规则

> **说明：**
> - 暂时不支持扩展协议的 SQL 语句的限流。
> - 如果 SQL 语句中的数据库对象已经存在限流规则，则无法新增规则。

云数据库 PostgreSQL 提供了限流的函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.add_rule_with_query(
        query_string text,
        work_node int,
        max_concurrency int,
        is_enabled boolean,
        memory_only boolean
)
```

add_rule_with_query 函数的参数解释如下，该函数返回值为限流的 SQL 语句的 query_id：

> **说明：**
> - query_string：需要限流的 SQL 语句；
> - work_node：限流规则生效的节点，0表示主实例和只读实例都生效，1表示只有主实例生效，2表示只有只读实例生效；

- max_concurrency：该限流规则允许的最大并发数量；
- is_enabled：当前规则是否生效，取值为 true 和 false；
- memory_only：可选，是否需要将规则持久化到表中，重启或 HA 后可使用命令重新将规则加载进内存；

SQL 限流示例如下：

```
postgres=# select tencentdb_sql_throttling.add_rule_with_query('select
pg_sleep(1);',0,1,true);
 add_rule_with_query
---------------------
 -3416356442043621232
(1 row)
```

限流规则设置完成后可查看限流规则，请参考 查看限流规则 。

删除限流规则

通过 query_id 删除规则

在获取到 query_id 之后，我们可基于 query_id 删除限流规则。云数据库 PostgreSQL 提供了一个函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.drop_rule(
        queryid bigint,
        memory_only boolean
);
```

drop_rule 函数的参数解释如下：

说明：

- queryid：将 query 语句规范化后哈希得到的64位整数，可通过 pg_stat_statements 或 pg_stat_activity 查询得到；
- memory_only
  - memory_only 为 true 时，只删除内存中的规则，持久化表中的规则不受影响，可看作从内存中卸载限流规则；
  - memory_only 为 false 时，将同时删除内存和表中的规则，如果无法删除，将打印 WARNING 日志。只读实例只能删除内存中的规则。

删除示例如下：

```
postgres=> SELECT
tencentdb_sql_throttling.drop_rule(497939862935121343,true);
-[ RECORD 1 ]
drop_rule |
```

当需要删除某些没有加载到内存中的 SQL 限流规则时，您可直接删除持久化表格中对应的 SQL 限流规则数据即可。

删除所有规则

当您需要删除所有 SQL 限流规则时，云数据库 PostgreSQL 提供了一个函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.drop_all_rules(
        memory_only boolean
);
```

drop_all_rules 函数的参数解释如下：

> **说明：**
> - memory_only
> - memory_only 为 true 时，只删除内存中的规则，持久化表中的规则不受影响，可看作从内存中卸载限流规则；
> - memory_only 为 false 时，将同时删除内存和表中的规则，如果无法删除，将打印 WARNING 日志。只读实例只能删除内存中的规则。

删除示例如下：

```
postgres=> select tencentdb_sql_throttling.drop_all_rules(true);
 drop_all_rules
----------------

(1 row)
```

加载限流规则

加载限流规则是指将存储在磁盘中的 SQL 限流规则加载到内存中，并使之生效。

通过 query_id 加载限流规则

在获取到 query_id 之后，我们可基于 query_id 加载限流规则。云数据库 PostgreSQL 提供了一个函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.load_rule(queryid bigint);
```

load_rule 函数的参数解释如下：

> **说明：**
> queryid：将 query 语句规范化后哈希得到的64位整数，可通过 pg_stat_statements 或 pg_stat_activity 查询得到；

加载示例如下：

```
postgres=> SELECT
tencentdb_sql_throttling.load_rule(497939862935121343);
-[ RECORD 1 ]
load_rule |

postgres=> SELECT * FROM tencentdb_sql_throttling.rules;
-[ RECORD 1 ]-------+----------------------------------
queryid             | 497939862935121343
work_node           | 0
max_concurrency     | 10
is_enabled          | t
query_string        | SELECT id FROM t WHERE id = 594017
current_concurrency | 10
total_hit_count     | 4760
reject_count        | 4713

postgres=>
```

加载所有限流规则

当您需要加载所有限流规则时，云数据库 PostgreSQL 提供了一个函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.load_all_rules();
```

加载示例如下：

```
postgres=> select tencentdb_sql_throttling.load_all_rules();
```

```
 load_all_rules
----------------

(1 row)
```

持久化限流规则

持久化是指在将内存中的 SQL 限流规则存储到表格中，以便后续可以直接加载生效，不需要重新设置。

通过 query_id 持久化限流规则

在获取到 query_id 之后，我们可基于 query_id 持久化限流规则。云数据库 PostgreSQL 提供了一个函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.dump_rule(queryid bigint);
```

dump_rule 函数的参数解释如下：

> **说明：**
> queryid：将 query 语句规范化后哈希得到的64位整数，可通过 pg_stat_statements 或 pg_stat_activity 查询得到；

持久化示例：

```
postgres=> select
tencentdb_sql_throttling.dump_rule(440101247839410938);
 dump_rule
-----------

(1 row)
```

持久化所有限流规则

如您需要持久化所有的限流规则，云数据库 PostgreSQL 提供了一个函数供您使用，函数定义如下：

```
tencentdb_sql_throttling.dump_all_rules();
```

持久化所有 SQL 限流规则示例：

```
postgres=> select tencentdb_sql_throttling.dump_all_rules();;
 dump_all_rules
```

```
----------------

(1 row)
```

如果已持久的规则被再次持久化会报错，请知悉，具体报错如下图所示。

```
postgres=> SELECT tencentdb_sql_throttling.dump_all_rules();
NOTICE:  duplicate key value violates unique constraint
"persistent_rules_table_pkey"
NOTICE:  duplicate key value violates unique constraint
"persistent_rules_table_pkey"
 dump_all_rules
----------------

(1 row)
```

查看限流规则

您可以使用 SQL 语句 `SELECT * FROM tencentdb_sql_throttling.rules;` 来查看当前内存中的限流规则，输出的字段解释如下：

说明：

- queryid：限流 SQL 的 queryid，将 query 语句规范化后哈希得到的64位整数；
- work_node：规则生效的节点，0表示主实例和只读实例都生效，1表示只有主实例生效，2表示只有只读实例生效；
- max_concurrency：允许的最大并发数；
- is_enabled：限流规则是否生效，取值为 true 和 false；
- query_string：限流规则对应的 SQL 语句；
- current_concurrency：当前该 SQL 语句的并发量；
- total_hit_count：该限流规则被命中的次数，该值为建立 SQL 限流规则以来的累积值，只有删除规则之后重新加载才能清空；
- reject_count：该规则总共限流了多少次，该值为建立 SQL 限流规则以来的累积值，只有删除规则之后重新加载才能清空；

查看内存中的限流规则示例：

```
postgres=> SELECT * FROM tencentdb_sql_throttling.rules;
-[ RECORD 1 ]-------+-------------------------------
```

```
queryid              | 497939862935121343
work_node            | 0
max_concurrency      | 10
is_enabled           | t
query_string         | SELECT id FROM t WHERE id = 553
current_concurrency | 0
total_hit_count      | 91
reject_count         | 81

postgres=>
```

您可以使用 SQL 语句 `SELECT * FROM tencentdb_sql_throttling.persistent_rules_table;` 来查看当前存储中的限流规则，输出的字段解释如下：

> **说明：**
> - queryid：限流 SQL 的 queryid，将 query 语句规范化后哈希得到的64位整数；
> - work_node：规则生效的节点，0表示主实例和只读实例都生效，1表示只有主实例生效，2表示只有只读实例生效；
> - max_concurrency：允许的最大并发数；
> - is_enabled：限流规则是否生效，取值为 true 和 false；
> - query_string：限流规则对应的 SQL 语句。

查看持久化到表格中的规则示例：

```
postgres=> SELECT * FROM
tencentdb_sql_throttling.persistent_rules_table;
-[ RECORD 1 ]---+-----------------------------------
queryid         | 497939862935121343
work_node       | 0
max_concurrency | 10
is_enabled      | t
query_string    | SELECT id FROM t WHERE id = 594017
```

修改限流规则

云数据库 PostgreSQL 提供了函数 change_rule_status 来修改限流规则，函数定义如下：

```
tencentdb_sql_throttling.change_rule_status(
        queryid bigint,
        is_enabled boolean,
```

```
        max_concurrency int
);
```

函数参数解释如下：

> **说明：**
> - queryid：限流 SQL 的 queryid，将 query 语句规范化后哈希得到的64位整数；
> - is_enabled：限流规则是否生效，取值为 true 和 false；
> - max_concurrency：允许的最大并发数,默认值为-1，代表不修改。

修改示例：

```
postgres=> select
tencentdb_sql_throttling.change_rule_status(440101247839410938,true,300)
;
 change_rule_status
--------------------

(1 row)
```

当系统出现异常，您可以调用 change_rule_current_concurrency 函数来手动修改某个规则的当前并发数，保障规则继续运行，函数定义如下：

```
tencentdb_sql_throttling.change_rule_current_concurrency(
    queryid bigint,
    new_currenct_concurrency int
);
```

change_rule_current_concurrency 参数说明如下：

> **说明：**
> - queryid：限流 SQL 的 queryid，将 query 语句规范化后哈希得到的64位整数；
> - new_currenct_concurrency：该规则新的并发数。

调用示例如下：

```
postgres=> SELECT
tencentdb_sql_throttling.change_rule_current_concurrency(497939862935121
343,20);
```

```
-[ RECORD 1 ]--------------------+-
change_rule_current_concurrency |

postgres=> SELECT * FROM tencentdb_sql_throttling.rules;
-[ RECORD 1 ]-------+--------------------------------
queryid             | 497939862935121343
work_node           | 0
max_concurrency     | 10
is_enabled          | t
query_string        | SELECT id FROM t WHERE id = 553
current_concurrency | 20
total_hit_count     | 286028
reject_count        | 283788

postgres=>
```

修改规则生效状态且增加并发数，生效：

```
postgres=> SELECT * FROM tencentdb_sql_throttling.rules;
-[ RECORD 1 ]-------+--------------------------------
queryid             | 497939862935121343
work_node           | 0
max_concurrency     | 10
is_enabled          | t
query_string        | SELECT id FROM t WHERE id = 553
current_concurrency | 20
total_hit_count     | 286028
reject_count        | 283788

postgres=> SELECT
tencentdb_sql_throttling.change_rule_status(497939862935121343,false,20)
;
-[ RECORD 1 ]------+-
change_rule_status |

postgres=> SELECT * FROM tencentdb_sql_throttling.rules;
-[ RECORD 1 ]-------+--------------------------------
queryid             | 497939862935121343
work_node           | 0
max_concurrency     | 20
is_enabled          | f
query_string        | SELECT id FROM t WHERE id = 553
```

```
current_concurrency | 20
total_hit_count     | 358931
reject_count        | 356691
```

使用 starocks_fdw 实现读写分离

最近更新时间：2024-09-23 14:46:41

云数据库 PostgreSQL 支持自研的 starrocks_fdw 插件，让您能够将 starrocks 中的表作为 PostgreSQL 的外部表，从而实现冷热存储分离。该插件的使用兼容 mysql_fdw 插件，以下为一个使用示例，更多详细的参数及使用请参考 mysql_fdw 官网 。本文的前提是您已经部署完毕 starrocks ，且可以访问。同时您已经购买了云数据库 PostgreSQL 实例。
本文中，已经购买一个云数据库 PostgreSQL 实例，且在同一个 VPC 下的 CVM 上已经自建一个 starrocks。下面将为您一一描述操作过程。

> **说明：**
> starocks_fdw 插件当前只支持 v13.14_r1.17及以上的 PG13版本。

在 starrocks 上准备数据

新建表格 sr_m1：

```
mysql> CREATE TABLE sr_m1 (
    ->     id int not null,
    ->     str INT
    -> );
Query OK, 0 rows affected (0.02 sec)
```

插入数据：

```
mysql> insert into sr_m1(id,str)values(1,1),(2,2),(3,3);
Query OK, 3 rows affected (0.48 sec)
{'label':'insert_61cc249a-758a-11ef-a99a-5254006f765d',
'status':'VISIBLE', 'txnId':'1024'}
```

检查数据插入成功：

```
mysql> select * from sr_m1;
+------+------+
| id   | str  |
+------+------+
|    1 |    1 |
|    3 |    3 |
|    2 |    2 |
```

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

在云数据库 PostgreSQL 上创建插件

新建 database 并切换至该数据库，如您已有数据库可以忽略：

```
postgres=> create database am_sr;
CREATE DATABASE
postgres=> \c am_sr
psql (14.11, server 13.14)
You are now connected to database "am_sr" as user "dbadmin".
am_sr=>
```

创建插件：

```
am_sr=> create extension starrocks_fdw;
CREATE EXTENSION
```

在云数据库 PostgreSQL 上创建外部服务和用户映射

创建外部服务器，其参数请参考 插件参数文档 。

```
am_sr=> CREATE SERVER server_sr FOREIGN DATA WRAPPER starrocks_fdw
OPTIONS (host '10.21.0.11', port '9030',instanceid 'ins-
xxx3tfi5',access_type '2',region 'ap-guangzhou', uin '100033123456',
own_uin '100033456789', vpcid 'vpc-fqxagfew', subnetid 'vpc-fqxawe23');
CREATE SERVER
```

创建用户映射，假设云数据库 PostgreSQL 的用户是 `dbadmin` ，希望使用用户名 `root` 和密码 `123456` 连接到 starrocks，则命令如下：

```
am_sr=> CREATE USER MAPPING for dbadmin SERVER server_sr OPTIONS
(password '123456', username 'root');
CREATE USER MAPPING
```

您可以使用如下命令查看当前已经新建的外部服务：

```
am_sr-> FROM pg_foreign_server;
  srvname   | srvowner | srvfdw | srvtype | srvversion |
```

```
                 srvoptions
-----------+----------+--------+---------+------------+-----------------
-----------------------------------------------
-----------------------------------------------------------------------
---------------------------------------
 server_sr |    16385 |  16495 |         |            |
{host=10.21.0.11,port=9030,instanceid=ins-xxx3tfi5,access_type=2
,region=ap-guangzhou,uin=100033123456,own_uin=100033456789,vpcid=vpc-
fqxagfew,subnetid=vpc-fqxawe23,linkid=352}
(1 row)
```

您同时可以使用如下命令查看当前已经建立的用户映射：

```
am_sr=> SELECT um.umid, um.srvid, fs.srvname, um.umuser, r.rolname AS
usename, um.umoptions FROM pg_user_mappings um JOIN pg_foreign_server fs
ON um.srvid = fs.oid JOIN pg_roles r ON um.umuser = r.oid;
 umid  | srvid |  srvname  | umuser | usename |            umoptions
-------+-------+-----------+--------+---------+------------------------
--------
 16499 | 16498 | server_sr |  16385 | dbadmin |
{password=123456,username=root}
(1 row)
```

在云数据库 PostgreSQL 上创建外部表

创建表格：

```
am_sr=> CREATE TABLE sr (id INT NOT NULL,str INT) PARTITION BY RANGE
(id);
CREATE TABLE
```

创建外部表：

```
am_sr=> CREATE foreign TABLE sr_m PARTITION OF sr FOR VALUES FROM (1) TO
(10) SERVER server_sr OPTIONS (dbname 'my_database', table_name
'sr_m1');
CREATE FOREIGN TABLE
```

查询数据：

```
am_sr=> select * from sr;
```

```
 id | str
----+-----
  1 |   1
  3 |   3
  2 |   2
(3 rows)
```

使用 tencentdb_ai 插件调用大模型

最近更新时间：2025-02-19 18:48:12

当前无论是基础大模型还是 AI 应用开发都发展迅速，云数据库 PostgreSQL 提供了 tencentdb_ai 插件，方便您在云数据库 PostgreSQL 实例中轻松调用网络可通的大模型 API，完成各种场景的应用开发。本插件支持大版本为 PostgreSQL 17 的实例。

> **说明：**
> - 使用 `tencentdb_ai` 的前提是您已经创建完成了大版本为 PostgreSQL 17 的实例。
> - 创建 `tencentdb_ai` 插件会关联创建 `pgcrypto` 插件，请知悉。

tencentdb_ai 插件函数说明

新增模型

`add_model` 函数定义如下：

```
tencentdb_ai.add_model(
    model_name NAME,
    version TEXT,
    region TEXT,
    json_path JSONPATH
);
```

参数解释如下：

- model_name：模型名称，如 `hunyuan-lite`，`deepseek-r1` 等。
- version：该模型的公共参数 `version`，如该模型不需要 `version` 参数则可以不填写。
- region：该模型的公共参数 `region`，如该模型不需要 `region` 参数则可以不填写。
- json_path：为调用该模型时，指定如何对模型的 `JSON` 响应进行提取。相当于对模型输出调用 `jsonb_path_query_first` 函数。如果不指定该参数，默认原样输出模型的原始响应。

查看已注册模型

`list_models` 函数定义如下：

```
tencentdb_ai.list_models(void);
```

更新模型

`update_model_attr` 函数定义如下：

```
tencentdb_ai.update_model_attr(
    model_name NAME,
    attr_name text,
    attr_value text
);
```

参数解释如下：

- model_name：模型名称。
- attr_name：需要更新的模型属性。可更新的模型属性有 `version`、`region`、`json_path`、`SecretId`、`SecretKey` 。
- attr_value：需要更新的属性的值。

具体 SecretId 和 SecretKey 请查看 访问管理 。

删除模型

`delete_model` 函数定义如下：

```
tencentdb_ai.delete_model(
    model_name NAME
);
```

参数解释如下：

- model_name：模型名称。

模型调用

call_model 函数定义如下：

```
tencentdb_ai.call_model(
    model_name NAME,
    common_params TEXT[],
    api_params TEXT[]
);
```

参数解释如下：

- model_name：模型名称。
- common_params：需要调用模型的公共参数，例如 `ARRAY['Action: ChatCompletions', 'Version: 2023-09-01']`

- **api_params：需要调用模型的专有参数，例如**
  `ARRAY['"Stream": false', '"Model": "hunyuan-lite"', '"Messages": [{"Role": "user", "Content": "您好"}]']`

除了使用 `call_model` 函数来进行模型调用之外，您还可以使用如下几个接口来进行特定场景的模型调用。

固定场景接口

对话

`chat_completions` **函数定义如下：**

```
tencentdb_ai.chat_completions(
    model_name  NAME,
    content TEXT,
    args TEXT[] default NULL
);
```

参数解释如下：

- **model_name：模型名称。**
- **content：对话内容。**
- **args：调用模型的其他参数，默认为 NULL。**

调用示例：

```
SELECT tencentdb_ai.chat_completions('hunyuan-lite', '您好');

         chat_completions

-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------
----------------------------------
 {"Response":{"RequestId":"*******-****-****-****-***********","Note":"以
上内容为AI生成，不代表开发者立场，请勿删除或修改本标记","Choices":
[{"Index":0,"Message":{"Role":"assistant","Co
ntent":"您好呀！很高兴能和您聊天，今天有什么想和我分享的
吗"},"FinishReason":"stop"}],"Created":1739786393,"Id":"*******-****-
****-****-*********","Usage":{"PromptTokens":3,"Completio
nTokens":17,"TotalTokens":20}}}
```

```
(1 row)
```

文本转向量

`get_embedding` 函数定义如下：

```
tencentdb_ai.get_embedding(
    model_name NAME,
    content TEXT[]
);
```

参数解释如下：

- model_name：模型名称。
- content：转化内容。

调用示例：

```
damoxing1=> SELECT tencentdb_ai.get_embedding('hunyuan-
embedding',ARRAY['您好', 'PostgreSQL']);
```

文本排序

`run_rerank` 函数定义如下：

```
tencentdb_ai.run_rerank(
    model_name NAME,
    query TEXT,
    documents TEXT[],
    args TEXT[] DEFAULT NULL
);
```

参数解释如下：

- model_name：模型名称。
- query：文本排序的问题。
- documents：待文本排序的内容。
- args：特定模型的指定参数。

```
damoxing1=> SELECT COUNT(*) FROM tencentdb_ai.run_rerank('lke-reranker-
base', '知识引擎大模型', ARRAY['混元大模型', '腾讯知识引擎']);
 count
-------
```

```
     2
(1 row)
```

插件使用示例

下面为您一一描述插件使用过程：

1. 创建插件

```
damoxing=> CREATE EXTENSION tencentdb_ai CASCADE;
NOTICE:  installing required extension "pgcrypto"
CREATE EXTENSION
```

检查插件创建完成

```
damoxing=> SELECT * FROM pg_extension;
  oid  |   extname     | extowner | extnamespace | extrelocatable |
extversion | extconfig | extcondition
-------+--------------+----------+--------------+----------------+----
--------+-----------+--------------
 14275 | plpgsql      |       10 |           11 | f              | 1.0
|           |
 16440 | pgcrypto     |    16437 |         2200 | t              | 1.3
|           |
 16477 | tencentdb_ai |    16437 |         2200 | t              | 1.0
| {16479}   | {""}
(3 rows)
```

2. 添加大模型

```
damoxing=> SELECT tencentdb_ai.add_model('hunyuan-lite','2023-09-01',
NULL, NULL);
 add_model
-----------

(1 row)
```

3. 添加大模型 API 调用的 SecretId 和 SecretKey

   具体 SecretId 和 SecretKey 请查看 访问管理 。

```
damoxing=> SELECT tencentdb_ai.update_model_attr('hunyuan-lite',
'SecretId', 'AKID***************');
 update_model_attr
-------------------

(1 row)
damoxing=> SELECT tencentdb_ai.update_model_attr('hunyuan-lite',
'SecretKey', '********************');
 update_model_attr
-------------------

(1 row)
```

4. 查看当前添加的模型列表

```
damoxing=> SELECT * FROM tencentdb_ai.list_models();
-[ RECORD 1 ]-------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------
model_name | hunyuan-lite
json_path  |
secretid   | **************************
secretkey  | **************************
version    |
region     |
id_random  | 516234453022
key_random | 134577899689
```

5. 调用大模型

聊天场景下，您可以使用 `call_model` 接口。

```
damoxing=> SELECT tencentdb_ai.call_model('hunyuan-lite',
                           ARRAY['Action: ChatCompletions', 'Version:
2023-09-01'],
                           ARRAY['"Stream": false', '"Model":
"hunyuan-lite"', '"Messages": [{"Role": "user", "Content": "您
好"}]']);damoxing(> damoxing(>


call_model
```

```
-----------------------------------------------------------------------
-------------------------------------------------------------
-----------------------------------------------------------------------
-------------------------------------------------------------
-----------------------------------------------------------------------
-------------------------------------------------------------
-----------------------------------------------------------------------
--
 {"Response":{"RequestId":"*******-****-****-****-
***********","Note":"以上内容为AI生成，不代表开发者立场，请勿删除或修改本标
记","Ch
oices":[{"Index":0,"Message":{"Role":"assistant","Content":"您好！很高兴
与您交流。请问有什么我可以帮助您的吗？无论是关于生活、工作、学
习还是其他方面的问题，我都会尽力为您提供帮
助。"},"FinishReason":"stop"}],"Created":1739793838,"Id":"*******-
****-****-****-***********
","Usage":{"PromptTokens":3,"CompletionTokens":33,"TotalTokens":36}}}
(1 row)

damoxing=>
```

开启 syscache/relcache 自动缓存淘汰机制

最近更新时间：2025-06-17 09:58:42

功能背景

在大规模复杂业务场景下，进程长连接，且该连接打开了很多的表、索引等，导致 syscache 和 relcache 缓存压力巨大，进而导致内存占用过高或缓存命中率下降，影响整体性能。

功能介绍

云数据库 PostgreSQL 支持基于 LRU （ Least Recently Used ）的 syscache 和 relcache 缓存淘汰机制，用户开启功能后，您可以设定一个进程最大允许的 syscache/relcache 数量。系统能够动态识别和淘汰长时间未被访问的关系缓存，优先保留热点元数据缓存，降低内存占用。

支持版本

内核小版本为 v13.20_r1.21、v14.17_r1.29、v15.12_r1.15、v16.8_r1.9、v17.4_r1.3 及以上的数据库实例支持该功能。您也可以通过 内核小版本升级 获取该功能。

适用场景

长连接环境下的 syscache / relcache 缓存管理。

使用说明

对于 syscache ，系统提供两个参数，用于管理 syscache 淘汰机制。

参数名称	说明	取值建议
tencentdb_syscache_max_num	• 一个进程最多能拥有多少个 syscache ，超过此数量开始进行缓存淘汰。 • 默认值为-1 ，表示不开启该功能。	• 如果发现进程内存占用过高，或者 syscache 数量异常增长，建议取值范围10000 – 50000。 • 如需进行修改，可选取20000等中间值，根据监控和实际效果逐步调整。
tencentdb_syscache_evict_num	• 一次淘汰会清理多少个 syscache。 • 默认值为1，表示一次只淘汰一个。	• 一般情况下，保持默认值1即可，保证淘汰过程平滑、对性能影响较小。 • 如果 syscache 增长较快，或者需要更快释放内存，可以适当提高

		该值为5 – 10 ，但不建议设置过大，以免一次性淘汰过多缓存影响查询性能。

对于 relcache ，系统提供两个参数，用于管理 relcache 淘汰机制：

参数名称	说明	取值建议
tencentdb_relcache_max_num	• 一个进程最多能拥有多少个 relcache ，超过此数量开始进行缓存淘汰。 • 默认值为-1 ，表示不开启该功能。	• 如果系统中存在大量表或频繁访问不同表，可能导致 relcache 数量激增，建议取值范围为1000 – 5000，具体数值请根据实例的内存大小和负载情况调整。 • 如果实例内存充足且访问表数量不多，可以保持默认值-1，关闭该功能，避免频繁淘汰带来的性能开销。
tencentdb_relcache_evict_num	• 一次淘汰会清理多少个 relcache。 • 默认值为1，表示一次只淘汰一个。	• 如果希望淘汰更快释放缓存，可以适当调大该值，比如5 – 10，减少淘汰次数，但可能带来瞬时性能波动。 • 如果希望淘汰过程更平滑，避免对性能产生较大影响，保持默认值1较为稳妥。

操作步骤

进入 云数据库 PostgreSQL 控制台 ，在需要管理 relcache 缓存的实例详情页，选择**参数设置**，单击参数运行值后的✎，根据需要修改 tencentdb_syscache_max_num 、tencentdb_syscache_evict_num 、tencentdb_relcache_max_num 和 tencentdb_relcache_evict_num 的参数值。