数智引航
专注数据 智能引航

VexDB 文档

向量数据库技术文档

版本：V3.0.0.1

生成日期：2026年01月13日

目录

关于VexDB

从这里开始

向量检索指南

管理员指南

API参考

编排组件

关于 VexDB

VexDB 是北京数智引航科技有限公司开发的一款基于关系模型的向量数据库。VexDB 融合了关系数据能力、多路语义检索能力，支持向量数据的统一存储、管理与高效查询，实现了标量数据与向量数据的协同管理，专注 GenAI 应用程序的构建。

VexDB 提供开发版和商业版两个版本。开发版适用于开发商、个人开发者和开源项目，提供一年免费试用；商业版面向企业级 AI 应用场景，提供更完整的关系型数据库特性及功能，更专业的技术支持服务。

产品介绍

VexDB是一款融合关系数据能力、多路语义检索能力的向量数据库，具备高性能、大容量、高精度、强一致、高可用、高安全、易用性的特点，能够支持百亿千维向量数据毫秒级结果查询，召回准确度达 99% 以上。

向量数据处理

VexDB 提供全面的向量计算能力，包括 KNN 精确查询、ANN 近似查询、向量与标量混合查询以及向量实时更新等功能，可高效支撑 **搜索业务** 与 **大模型业务** 场景：

- 搜索业务：图像、文本和视频的相似性检索、跨模态搜索等；
- 大模型应用：智能问答、AI 智能体、GPT 缓存、推理加速等。

在向量索引方面，VexDB 集成了 IVFFlat、IVFPQ、DiskANN、HNSW、HybridANN 等先进算法，结合图与倒排索引技术，适配高并发、低延迟、大规模数据等多样化场景需求。

关系型数据库核心能力

VexDB 兼容标准 SQL 语法，继承了关系型数据库的核心能力：

- 事务管理
- 数据一致性
- 分区表管理
- 查询优化
- 高可用架构
- 企业级安全
- 数据库运维监控

推荐阅读

阅读认识向量数据库掌握向量检索的相关术语；阅读向量检索指南了解 VexDB 向量功能的使用。

产品优势

高性能

- 支持 IVFFlat、IVFPQ 索引，实现千万级别高召回率，毫秒级响应。
- 支持 DiskANN 索引，改良了传统的 HNSW 索引。通过对于图索引算法的极致优化，重新设计构建算法、内存布局和磁盘数据结构，实现高并发下的高吞吐、低时延。
- 通过软硬件协同优化，进一步降低查询时延。
- 创新向标联合索引算法，能够自适应不同的标量选择率，同时发挥标量过滤索引和向量索引的加速效能，满足 AI 应用对于语义查询的复杂要求。

大容量

- 采用最新的量化压缩技术，降低75%存储成本，结合分层分片索引技术，VexDB单节点能够支持超过10亿条向量数据存储和高效查询。

高精度

- 通过一条 SQL 语句就能支持多路召回功能，将端到端的检索召回率提高 30%。
- 通过动态剪枝、位图加速和位置跳跃技术优化了复杂全文查询效率，借助 BM25 相关度评分方法，有效支撑了AI应用对于高精度语义查询的要求。

强一致

- 通过多版本并发控制算法进行事务管理，支持读已提交和可重复度两种隔离级别，都能够在高性能的前提下确保数据的强一致，实现 0 延迟的数据新鲜度。
- 支持数据原地更新（商业特性）以及动态实时更新。

  保证数据新鲜度的同时，提高高频更新场景下的性能稳定性，以及存储空间利用率，有效满足缓存场景和实时数据分析场景对于数据实时更新的需求。

高可用

- 支持一主多备的高可用部署架构，保障数据不丢失，服务不中断。
- 支持两地三中心跨 Region 容灾部署方案，保证极端灾难情况下数据的安全和可用性。

- 支持极致 RTO，高效并行回放，单点故障快速恢复。
- 支持备份即时恢复（PITR），避免误操作带来的数据丢失。

高安全

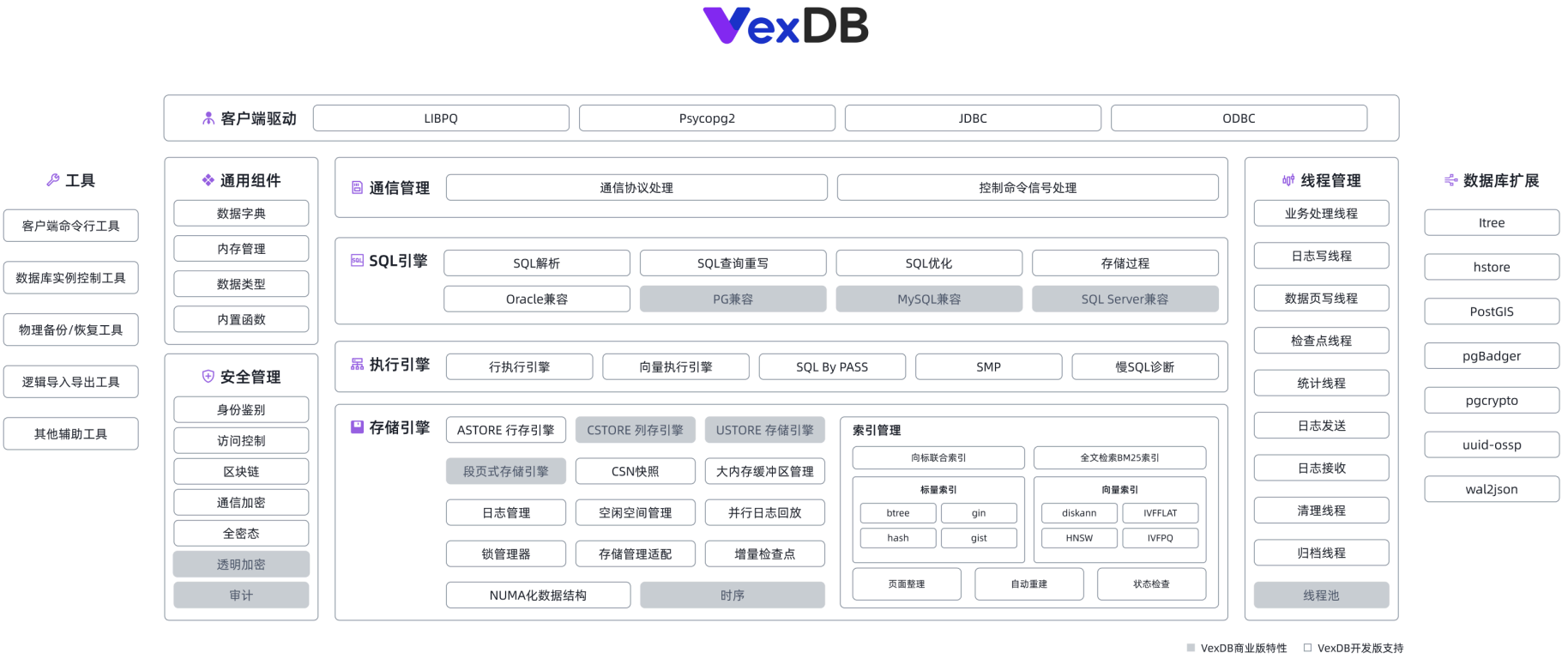
- 支持数据库加密存储，通过 AES128 加解密数据。
- 支持基于角色的访问控制模型和基于三权分立的访问控制模型。
- 支持统一审计机制（商业特性），可有效解决攻击者抵赖。
- 支持数据透明加密（商业特性），解决静态数据泄露问题。

易用性

- VexDB 支持多种硬件和操作系统平台。
- 提供了数据库使用全流程工具，实现易部署、易运维、易迁移。
- 通过原生的 SQL 语法，支持不同 AI 应用的特异化场景需求，统一关系查询、向量查询、和全文检索能力。
- 支持 Python、Java 等多语言开发者工具，高度兼容现有 AI 生态组件，降低开发门槛，促进大模型的价值发挥。

软件架构

VexDB 系统采用分层架构设计，充分满足了解耦、复用、灵活扩展和高效运维的现代数据库设计原则。



VexDB 主要模块包括：

- SQL引擎：负责 SQL 语句的解析、优化，并生成高性能的执行计划。

  负责接收 SQL 语句，进行词法、语法解析及语义分析，生成抽象的查询树。随后，基于代价的优化器（CBO）结合数据统计信息、索引情况与系统资源状态，从多种执行策略中选择最优路径，生成高性能的执行计划。

- 执行引擎：高效执行 SQL 引擎生成的计划，调度计算资源。

  调度 CPU、内存等计算资源，高效地执行由SQL引擎生成的计划。它实现多种数据操作算法（如连接、排序、聚合），以流水线或批处理方式处理数据，并将结果返回给客户端。

- 存储引擎： 保障数据安全，高效持久化存储和事务管理。

  负责数据在磁盘上的持久化存储、缓存管理（Buffer Pool）、事务管理（ACID）、redo恢复及空间分配。它确保数据的一致性、持久性和高效访问。

VexDB 不仅支持用于加速查询的 B-Tree、Hash 等标量索引，更集成了 IVFFlat、IVFPQ、HNSW、DiskANN 等常见的向量检索索引和向标混合索引 HybridANN，用于高效处理高维向量数据的相似性搜索，极大优化了诸如以图搜图、语义检索、推荐系统等AI场景的查询性能。

除了上述模块外，VexDB 还支持如下功能：

- 安全管理：提供如身份鉴别、访问控制、全密态等数据安全保障机制。
- 线程管理：区别于PostgreSQL系数据库的多进程结构，VexDB 采用线程模式。主要线程类型如：
  - 业务处理线程：负责接收客户端所有请求、解析SQL和生成执行计划等。
  - 日志写线程：将预写日志写入磁盘文件。
  - 统计线程：收集表和索引的空间信息和元组信息等，收集到的信息可以被优化器和 autovacuum 使用，还能作为数据库管理员的参考信息。
  - 通信管理：支持进行通信协议处理和控制命令信号处理。
- 数据库扩展管理：VexDB 支持扩展机制，允许数据库突破其初始设计的内置功能限制，从通用的关系型数据存储，变成了一个提供强大专用模块的计算平台，支持如 ltree、hstore、PostGIS 等扩展。
- 通用组件：数据库提供的通用组件包括数据字典、内存管理、数据类型、内置函数，提供统一性和可靠性。
- 工具：VexDB 提供客户端管理工具、物理备份/恢复工具、逻辑导入/导出工具等辅助用户使用的封装工具，降低运维难度的同时提升了软件易用性。
- 客户端驱动：支持通过驱动将各种语言编写的应用接入 VexDB 。如JDBC、ODBC、Psycopg2，有助于提高用户的开发效率和可移植性。

技术特性

VexDB 在兼容关系型数据库特性的同时，通过以下技术特性，为 AI 应用的开发与落地提供了可靠的向量数据管理解决方案。

多种索引算法支持

- VexDB 适配了通用索引扫描的 ANN（近似最近邻）查询。
- VexDB 实现了向量数据查询计划生成与索引选择能力的增强。
- VexDB 支持多种基于磁盘的向量索引结构，在保证 ACID 特性的同时，实现向量数据的高性能检索。

索引算法优化

VexDB 在处理大规模向量数据的相似性搜索时，创新性地实现了一种高效的基于磁盘的向量图索引 Graph Index。该索引结合了 HNSW 的层次化管理特点和 DiskANN 的磁盘存储优势，确保了查询的高效性，并提升了数据和查询的可靠性。

单个索引通过最新的量化和分层分片索引技术，能够支持高达十亿级数据的高效存储与查询。

高效向量检索

- VexDB 支持并行索引构建和扫描，通过并行化技术将查询时延降低了两倍以上；它还适配了多种 CPU 处理器架构的 SIMD 指令集，加速了距离计算性能，提升超过三倍，并支持自适应代码选择技术，实现了一次编译，多平台运行。
- VexDB 使用自适应算法来决定是否可以提前终止检索，从而将检索速度提升了 40%。与业界流行产品的相比，VexDB 在相同的环境和数据集上展现了超过 50% 的吞吐性能优势。

动态索引维护框架

- 引入页面整理机制，有效解决频繁更新场景下索引质量降低的问题。
- 智能控制索引膨胀，从而节省存储资源。

简单高效的管理方式

- 具备完备的日志体系，为每种基于磁盘的向量索引结构设计了专用的日志类型。
- 通过标准数据库驱动支持多语言应用接入，提供完善的数据库服务能力。

版本发布说明

通过阅读本节内容，您可以了解 VexDB 各版本的新功能与特性优化，以及错误修复情况等。建议您定期访问此页面以了解更新信息。

VexDB V3.0.0.1

发布日期

2025 年 11 月 11 日

新特性

Graph_Index索引增强

1. Graph_Index索引支持PQ量化和RaBitQ量化，通过新增的索引构建参数quantizer指定量化方法。通过向量量化技术能够将高维向量压缩成紧凑的编码，从而实现更快速的近似最近邻搜索，并减少索引的磁盘和内存占用。

2. Graph_Index索引支持并行vacuum，单个索引vacuum并行度受索引构建参数parallel_workers控制。

全文检索索引增强

1. 全文检索索引fulltext支持GLOBAL全局分区索引。创建fulltext索引语句不指定 LOCAL/GLOBAL 关键字时，默认创建为 GLOBAL 索引。
2. 全文检索索引fulltext的@~@查询条件新增查询参数BOOST和MINIMUM_SHOULD_MATCH，分别表示查询评分权重系数和分词匹配个数。
3. 新增支持分词函数bm25_query_tokenize，函数返回包括扩展分词关键字在内的字符串分词结果。

支持索引状态信息查询

1. 新增用于检查索引状态的函数index_inspect。调用函数将输出指定索引的内部结构信息。支持的索引类型包括Graph_Index（HNSW）、fulltext、DiskANN。

配套工具/驱动信息

工具/驱动	版本
HAS	V3.5.2
JDBC	V2.13
vexdb-psycopg2驱动	V1.1.0
pyvector-vexdb扩展包	V1.0.2

问题修复

1. 修复了并发业务场景下，重启数据库集群后主备数据文件不一致的问题。
2. 修复了VexDB商业版在PG兼容模式下，创建带WHERE条件的IVFPQ局部索引时出现数据库宕机的问题。

从这里开始

通过阅读本章内容，可以帮助您快速了解向量检索与向量数据库的运用，掌握 VexDB 的安装、连接与简单使用。

认识向量数据库

向量数据库与 AI 应用

传统数据库中，以行和列的形式存储数字、字符串等标量数据。相较于传统数据库，向量数据库则更擅长处理非结构化数据，比如：文本、图像和音频。而在机器学习中，数据通常是以向量形式存储的。因此，向量数据库和 AI 大模型之间的紧密结合，共同推动了 AI 应用的开发与落地。

向量类型和向量嵌入

向量是同时具有方向和大小的量，其在数学上表示为多维空间中的坐标，比如 N 维空间中的向量就是一个具有 N 个维度的坐标。向量的大小（也称为长度或模）通过公式 $\sqrt{a_1^2 + a_2^2 + a_3^2 + \cdots}$ 计算获得。向量的方向可以通过与其他向量的夹角来描述。两个向量 *A* 和 *B* 的夹角余弦值（即余弦相似度）计算公式为：$\cos\theta = \frac{A \cdot B}{|A| \cdot |B|}$，其中 $A \cdot B$ 表示向量的点积，$|A|$ 和 $|B|$ 分别表示向量的模。

向量嵌入（Vector Embeddings）是一种数据向量化的手段，指的是使用机器学习技术将各类非结构化数据（文本、图像和音频等）转化为固定长度的数字向量的过程，被广泛应用于多模态非结构化数据检索任务中。

通过将原始数据嵌入编码到同一个空间中，使之具有等长同类型的向量标签，并且数据之间的语义相似度可以通过向量之间的相似度计算进行度量。比如：通过表征训练可以让具有相同含义的单词，句子，片段，甚至是文档，图片等实体具有更高相似度的嵌入向量；反之，具有不同语义的实体的嵌入向量相似度较低。

相似性搜索

在传统数据库中，数据的查询操作是从数据库中找到与查询条件精确匹配的行。而在向量数据库中，向量检索通过对比向量之间特征的相似性，查找最相似的匹配，也就是向量的相似性搜索（Similarity Search）。

在介绍向量的相似性搜索之前，我们需要先了解特征和向量的关系。

通常情况下，我们通过区分各物体的不同的特征来判断物体的种类。为了量化判断的过程，假设我们将每种特征指标看作一个数轴，那么由两种特征维度构成的平面直角坐标系中，每个参与比较的物体都根据其在两个维度上的符合情况而位于平面内的一点。这个点的横纵坐标值分别代表了其对两个特征的吻合程度。对于越复杂的物体，在区分它们时所需要的特征维度也越多。

根据前面的例子我们不难推断出，在坐标系中越接近的两个点，其具有的特征越相似，代表两个物品的相似度越高。把数据的特征用向量来表示，向量数组的每个元素代表一个特征维度，这样就能够通过计算向量之间的距离来判断它们的相似度，这就是相似性搜索。

KNN 和 ANN

K 最近邻（K-Nearest Neighbor, KNN）算法和近似最近邻（Approximate Nearest Neighbor, ANN）算法是向量检索中常用的两种技术。

KNN 优先考虑准确性，细致地识别“K”个最近邻居。ANN 注重速度和效率，查找近似查询点的最近邻居，无法保证得到一组精确的最佳匹配；但 ANN 能够在高准确性和更快性能之间取得平衡。

检索增强生成

大语言模型（Large Language Model, LLM）是经过大量文本数据训练得到的 AI 模型。

RAG（Retrieval Augmented Generation）即检索增强生成。作为一种 AI 框架，它能够将向量检索的优势与 LLM 的功能结合在一起——将从知识库（比如向量数据库）中检索到的信息作为提示输入到大语言模型，以此来优化大语言模型的生成结果，使得大语言模型在生成更精确、更贴合上下文答案的同时，有效减少产生误导性信息的可能。

安装 VexDB

在进行正式安装之前，为了确保安装程序的顺利运行，并正常启动和使用数据库服务，必须先执行安装前准备中介绍的步骤，再进行交互安装。

安装前准备

本章节旨在指导用户检查当前环境是否符合 VexDB 数据库的安装部署要求，并详细介绍了用户应该如何检查、配置系统与环境，为 VexDB 数据库的安装部署做好准备工作。

以下操作均使用操作系统 root 用户执行。

服务器准备

以下介绍 VexDB 的硬件环境要求。硬件配置的规划需要考虑数据规模以及期望的数据库响应速度，请根据实际情况进行规划。

开发与测试环境

VexDB 数据库服务器在开发与测试环境中应具备的硬件配置要求如下：

表1 开发与测试环境的硬件配置要求

名称	CPU	内存	本地存储	网络
VexDB	2核+	8GB+	SAS，100G	千兆

生产环境

VexDB数据库服务器在生产环境中应具备的最低及建议硬件配置要求如下：

表2 生产环境的最低及建议硬件配置

项目	最低配置	建议配置
服务器	单机部署时要求 1 台物理机或虚拟机。集群部署时要求服务器与主备数量相关，如部署一主两备环境要求3台物理机或虚拟机。	/
内存	功能调试建议内存最低取值为 10 GB。具体的服务器内存配置应根据实际业务场景做调整。搭建一主两备集群，服务器内存配置最低取值为 12 GB，否则会出现共享内存（shared memory）不够的情况。	复杂的查询对内存的需求量比较高，在高并发场景下建议用户使用大内存的机器，或使用负载管理限制系统的并发量。性能测试和商业部署时，单实例部署时，服务器内存配置推荐为 128 GB 及以上。集群搭建一主两备时，服务器内存配置推荐为 128 GB 及以上。说明建议使用鲲鹏 920 标配服务器型号作为最佳配置，以获取更优性能结果。
CPU	功能调试时，CPU 配置最小为1×8核 2.0 GHz。	性能测试和商业部署时，单实例部署建议1×16核 2.0 GHz以上。支持超线程和非超线程两种 CPU 模式。VexDB 各节点的 CPU 模式要求一致。
硬盘	用于安装 VexDB 的硬盘需最少满足如下要求：至少 1 GB 用于安装 VexDB 的应用程序包。每个主机需大约 300 MB 用于元数据存储。预留 70% 以上的磁盘剩余空间用于数据存储。预留 30 GB 磁盘用于存放 WAL 数据文件。	建议系统盘配置为 Raid1，数据盘配置为 Raid5，且规划 4 组 Raid5数据盘用于安装 VexDB。注意 Raid 配置 Disk Cache Policy 一项需要设置为 Disabled，否则机器异常掉电后有数据丢失的风险。搭建双机单活共享存储需准备两个 lun 裸存储设备。其中作为仲裁存储设备最少为 64 MB，建议为1GB；共享存储设备建议为100GB。
网络	单机安装最低为千兆网络。集群安装最低为万兆网络。	至少满足最低要求。

操作系统及CPU满足度检查

以下介绍如何查看当前的系统环境信息，同时列举了安装部署 VexDB 的 CPU 及操作系统要求，为后续安装部署做准备。

1. 通过查看环境信息确认当前主机的系统环境。

   ```
   lscpu
   cat /etc/os-release
   ```

   VexDB针对不同 CPU 架构（不区分操作系统）进行了适配，根据 **表1** 判断当前环境是否满足安装部署的要求。

   表1 VexDB 安装包支持情况

操作系统	CPU 架构
Linux	X86（包括海光 C86）
Linux	鲲鹏920

2. 根据操作系统、操作系统版本、CPU架构、CPU型号选择对应的数据库软件安装程序。例如：

   查询环境信息：

   ```
   lscpu
   cat /etc/os-release
   ```

第三方依赖包安装

本文主要介绍 VexDB数据库在安装过程中所需要用到的第三方依赖包。

在安装数据库前应下载本小节所列的数据库所需依赖，若缺少本步骤，可能会因缺少部分依赖导致安装数据库过程中止。

- 通常，安装 VexDB 数据库的基础依赖包如下：

  ```
  zlib-devel、libaio、libuuid、readline-devel、krb5-libs、libicu、libxslt、tcl、perl、openldap、pam、openssl-devel、libxml2、bzip2
  ```

- 麒麟环境安装 VexDB 数据库的基础依赖包如下：

  ```
  readline、python2、libicu、cracklib、libxslt、tcl、perl、openldap、pam、systemd-libs、bzip2、gettext、libaio、ncurses-libs
  ```

- Ubuntu环境安装 VexDB 数据库的基础依赖包如下，使用 `apt install <name>` 命令来执行安装依赖：

  ```
  libreadline5 、zlib1g、libxml2、libaio1、libncurses5、gettext
  ```

安装示例语句如下（以通用环境所需的依赖为例）：

```
yum install -y zlib-devel libaio libuuid readline-devel krb5-libs libicu libxslt tcl perl openldap pam openssl-devel libxml2 bzip2
```

结果如下表示成功：

```
  Verifying      : libselinux-python-2.5-14.1.el7.x86_64                                                    37/43
  Verifying      : e2fsprogs-libs-1.42.9-13.el7.x86_64                                                      38/43
  Verifying      : krb5-libs-1.15.1-34.el7.x86_64                                                           39/43
  Verifying      : libselinux-utils-2.5-14.1.el7.x86_64                                                     40/43
  Verifying      : libcom_err-1.42.9-13.el7.x86_64                                                          41/43
  Verifying      : libselinux-2.5-14.1.el7.x86_64                                                           42/43
  Verifying      : e2fsprogs-1.42.9-13.el7.x86_64                                                           43/43

Installed:
  libicu.x86_64 0:50.2-4.el7_7                                  openssl-devel.x86_64 1:1.0.2k-25.el7_9

Dependency Installed:
  keyutils-libs-devel.x86_64 0:1.5.8-3.el7  krb5-devel.x86_64 0:1.15.1-51.el7_9       libcom_err-devel.x86_64 0:1.42.9-19.el7
  libkadm5.x86_64 0:1.15.1-51.el7_9         libselinux-devel.x86_64 0:2.5-15.el7      libsepol-devel.x86_64 0:2.5-10.el7
  libverto-devel.x86_64 0:0.2.5-4.el7

Updated:
  libuuid.x86_64 0:2.23.2-65.el7_9.1        libxml2.x86_64 0:2.9.1-6.el7_9.6            libxslt.x86_64 0:1.1.28-6.el7

Dependency Updated:
  e2fsprogs.x86_64 0:1.42.9-19.el7          e2fsprogs-libs.x86_64 0:1.42.9-19.el7       krb5-libs.x86_64 0:1.15.1-51.el7_9
  libblkid.x86_64 0:2.23.2-65.el7_9.1       libcom_err.x86_64 0:1.42.9-19.el7           libmount.x86_64 0:2.23.2-65.el7_9.1
  libselinux.x86_64 0:2.5-15.el7            libselinux-python.x86_64 0:2.5-15.el7       libselinux-utils.x86_64 0:2.5-15.el7
  libsmartcols.x86_64 0:2.23.2-65.el7_9.1   libss.x86_64 0:1.42.9-19.el7                openssl.x86_64 1:1.0.2k-25.el7_9
  openssl-libs.x86_64 1:1.0.2k-25.el7_9     util-linux.x86_64 0:2.23.2-65.el7_9.1

Complete!
```

防火墙配置

在普遍场景中，数据库的业务服务和数据库节点的网络联通都是在安全域内完成数据交互。

为了保证 VexDB 的正常使用，需要将节点间的访问端口打通才可以保证读写请求、数据心跳等信息的正常传输。如果没有特殊安全的要求，建议关闭目标节点的防火墙。

若无法关闭防火墙，可以参考设置防火墙白名单。

1. 检查防火墙状态（以 CentOS 7 操作系统为例，不同操作系统的命令可能不同）。

```
sudo firewall-cmd --state
sudo systemctl status firewalld.service
```

2. 关闭防火墙服务。

```
sudo systemctl stop firewalld.service
```

3. 关闭防火墙自动启动服务。

```
sudo systemctl disable firewalld.service
```

4. 检查防火墙状态。

```
sudo firewall-cmd --state
sudo systemctl status firewalld.service
```

关闭透明大页

开启透明大页可能会对数据库性能产生负面影响，所以本小节将指导用户如何关闭透明大页。

1. 编辑 service 文件，将如下内容放入文件中。

```
vi /etc/systemd/system/disable-thp.service
```

内容如下：

```
[Unit]
Description=Disable Transparent Huge Pages (THP)

[Service]
Type=simple
ExecStart=/bin/sh -c "echo 'never' > /sys/kernel/mm/transparent_hugepage/enabled && echo 'never' > /sys/kernel/mm/transparent_hugepage/defrag"

[Install]
WantedBy=multi-user.target
```

2. 修改完成后，执行如下命令加载系统服务，并设置开机自启动。

```
systemctl daemon-reload
systemctl start disable-thp
systemctl enable disable-thp
```

3. 查看THP状态，当返回结果均为 `always madvise [never]` 时表示成功设置透明大页永久关闭。

```
cat /sys/kernel/mm/transparent_hugepage/enabled
cat /sys/kernel/mm/transparent_hugepage/defrag
```

时区配置

数据库默认时区为中国时区，如果操作系统时区和数据库默认时区不一致，会导致数据库日志显示时间和实际时间不一致。所以本小节将指导用户如何查看和修改时区配置。

1. 将 `/usr/share/zoneinfo/` 目录下的时区文件拷贝为 `/etc/localtime` 文件，从而设置时区和时间。

```
cp /usr/share/zoneinfo/$主时区/$次时区 /etc/localtime
```

例如：

```
cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

2. 检查时区配置，可通过以下命令查看操作系统当前的时区。

```
timedatectl
```

返回结果如下表示时区正常。

```
      Local time: Thu 2024-09-05 17:34:43 CST
  Universal time: Thu 2024-09-05 09:34:43 UTC
        RTC time: Thu 2024-09-05 09:34:43
       Time zone: Asia/Shanghai (CST, +0800)
     NTP enabled: yes
```

IPC 参数配置

当 RemoveIPC=yes 时，操作系统会在用户退出时，删除该用户的 IPC 资源（共享内存段和信号量），从而使得 VexDB 服务器使用的 IPC 资源被清理，可能引发数据库宕机，所以需要设置 RemoveIPC 参数为 no。

1. 进入 /etc/systemd/logind.conf 文件，在配置文件末尾新增配置 RemoveIPC=no，若文件中已设置则跳过本步骤。

```
vi /etc/systemd/logind.conf
```

2. 进入 /usr/lib/systemd/system/systemd-logind.service 文件，新增或修改配置 RemoveIPC=no，若文件中已设置则跳过本步骤。

```
vi /usr/lib/systemd/system/systemd-logind.service
```

3. 重新加载配置参数。

```
systemctl daemon-reload
systemctl restart systemd-logind
```

4. 检查修改是否生效。

由于 CentOS 操作系统环境的 RemoveIPC 默认为关闭，则执行如下语句是无返回结果的。用户在确保步骤 1 至步骤 3 已执行的前提下，可正常安装数据库。

```
loginctl show-session | grep RemoveIPC
systemctl show systemd-logind | grep RemoveIPC
```

内核参数配置

为了优化系统性能、确保稳定性，并满足数据库对资源管理的特殊需求。用户应该对操作系统的内核参数配置进行合理配置，这关系到数据库的启动和使用时的资源配置。

设置内核参数前需参考内核参数介绍，结合实际情况调节参数大小，否则会影响数据库的安装部署，以下示例仅供参考。

1. 编译内核参数配置文件 /etc/sysctl.conf，将内核信息写入文件末尾并保存。

   其中关键配置项含义参考后文的介绍。

```
fs.aio-max-nr=1048576
fs.file-max= 76724600
kernel.sem = 4096 2097152000 4096 512000
kernel.shmall = 26843545         # pages,  80% MEM or higher
kernel.shmmax = 68719476736     # bytes,  50% MEM or higher
kernel.shmmin = 819200
net.core.netdev_max_backlog = 10000
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 4194304
net.core.somaxconn = 4096
net.ipv4.tcp_fin_timeout = 5
vm.dirty_background_bytes = 409600000
vm.dirty_expire_centisecs = 3000
vm.dirty_ratio = 80
vm.dirty_writeback_centisecs = 50
vm.overcommit_memory = 0
vm.swappiness = 0
net.ipv4.ip_local_port_range = 40000 65535
fs.nr_open = 20480000
```

2. 重载配置，使其在不关机的情况下生效。

```
sysctl -p
```

   主要参数介绍如下：

   - vm.dirty_background_bytes：该参数表示触发回刷的脏页数据量，超过该参数，脏页刷到磁盘，单位：Btye。
   - kernel.shmmin 参数用于设置系统可以创建的共享内存段的数量。通常取默认值：4096，单位：个。

- kernel.shmall 参数用于设置一个共享内存段内可以分配的共享内存页的最大数量。共享内存页是共享内存段中的最小可管理单元。建议取值为：系统内存*0.8/PAGE_SIZE，单位为 Byte。0.8 为推荐配置，

  注意：该参数设置太小有可能导致数据库启动报错。若 kernel.shmall 设置的过小，可能会导致操作系统无法为 VexDB 数据库分配其所需的共享内存，导致数据库启动失败。

- kernel.shmmax 参数用于设置单个共享内存段可拥有的最大尺寸。建议取值为：系统内存*0.5，单位：Byte。

  注意：

  - 数据库参数 shared_buffers 设置 VexDB 数据库需要使用的共享内存大小，建议 kernel.shmmax 设置的单个共享内存段大小可覆盖 VexDB 数据库需要使用的共享内存大小。
  - 数据库参数 max_process_memory 设置 VexDB 数据库可用的最大物理内存，此值与 VexDB 所需的共享内存共同决定 VexDB 数据库在运行时可以使用的内存总量，若设置不合理可能导致操作系统 OOM (Out of Memory) 问题。

SELinux配置

SELinux（Security-Enhanced Linux）是一个 Linux 内核模块，为 Linux 内核提供了一个强制访问控制机制，以保护系统免受未授权篡改或破坏的威胁。

1. 查看是否开启SELinux。如果是未开启则是 disabled，已开启则是enforcing，宽容模式为 permissive。

   ```
   getenforce
   ```

   若为开启状态则临时关闭SELinux。

   ```
   setenforce 0
   ```

2. 为避免临时关闭的SELinux在重启后又被开启，建议用户执行此步骤通过修改配置文件永久关闭SELinux。

   涉及重启系统，用户需谨慎操作。

   1）编辑配置文件。

   ```
   vi /etc/selinux/config
   ```

   2）将 `SELINUX=enforcing` 修改为 `SELINUX=disabled` 。

   3）重启系统。 非必选，若当前不执行重启操作，则在下次重启时上述修改会自动生效。

   ```
   reboot
   ```

关闭 SWAP 交换内存（可选）

关闭 swap 交换内存是为了保障数据库的访问性能，避免把数据库的缓冲区内存淘汰到磁盘上。 如果服务器内存比较小，内存过载时，可打开 swap 交换内存保障正常运行。

在各数据库节点上，使用 `swapoff -a` 命令将交换内存关闭。

```
swapoff -a
```

设置网卡 MTU 值（可选）

将各数据库节点和交换机的网卡 MTU 值（最大传输单元）设置为相同大小（MTU 值≥1500），推荐值：8192（MTU 值可根据需要自行修改）。需要连同交换机一起修改（修改方法请咨询交换机厂商）。

执行如下命令，设置网卡MTU值。网卡编号可通过 `ip a` 命令查看。

- 方法一：

```
#ifconfig 网卡编号 mtu 值
ifconfig eth0 mtu 8192
```

- 方法二：

```
cat /sys/class/net/网卡编号/mtu
echo "8192" > /sys/class/net/网卡编号/mtu
```

添加排序规则（可选）

在凝思操作系统中安装数据库前需添加排序规则，其他系统不需要，即可以选择跳过本小节。若其他操作系统下需要配置，也可以参考添加排序规则。

在初始化实例时，数据库会将当前操作系统所有 collation 语言排序，并且将规则信息（通过命令 `locale -a` 查看）加入到 PG_COLLATION 表中。

如果语言排序规则是在初始化实例之后才加入系统的，则不能自动加入到 PG_COLLATION 系统表中，在创建表等对象时如果指定了该排序规则，则会报错该排序规则不存在。

如果语言排序规则是在初始化实例之后才加入系统的，则不能自动加入到 PG_COLLATION 系统表中，在创建表等对象时如果指定了该排序规则，则会报错该排序规则不存在。

例如：Linux 操作系统默认安装时只有 C、POSIX、zh_CN.utf8 等语言排序规则，如果需要数据库支持 en_US.utf8 等其他语言排序规则，需要在初始化实例之前安装其语言排序规则。如果在初始化实例后再安装语言排序规则，则需要通过创建一个新的数据库，指定 lc_collate、lc_ctype 来使数据库支持新增的语言排序规则。

凝思操作系统中，添加语言排序规则方法如下：

1. 执行如下命令，勾选（按空格）en_US.UTF-8 编码。

```
sudo dpkg-reconfigure locales
```

2. 执行如下命令，添加排序规则。

```
sudo apt-get install locales
```

3. 执行如下命令，显示包含 en_US.UTF-8。

```
locale -a
```

重新启动 VexDB，即可创建包含 en_US.UTF-8 数据库。

交互安装（单机）

交互安装以脚本实现数据库安装过程中的各种配置选择，如输入密码、选择路径等，简洁直观的辅助用户完成数据库的安装。

交互安装支持 **实例化** 与 **非实例化** 两种模式：

- 实例化：安装数据库，并进行初始化。
- 非实例化：仅安装数据库所需的二进制文件，不进行数据库初始化。

本节以 **实例化** 为主要流程进行介绍，若选择 **非实例化** 安装，也可以根据本文介绍的内容单独初始化数据库。

> **说明**
>
> 在进行数据库实例化安装时，数据库字符集编码会根据环境变量自动设置，参考顺序为：LC_ALL>LC_CTYPES>LANG，如以上环境变量都不存在，则默认为 en_US.utf8。

安装过程

步骤1：创建数据库安装用户和目录

1. 以 root 用户登录操作系统。

   创建数据库安装用户（可自定义），设定初始密码（需要重复输入 2 次且完全一致）。自定义的操作系统用户名可用于安装数据库，文件属组和属主需要进行相应替换。

```
useradd -m vexdb
passwd vexdb
```

建议安装数据库的操作系统用户名中包含的字母均使用小写。否则，在执行 SQL 时，指定含有大写字母的操作系统同名数据库初始化用户时，需要被双引号包裹才能被识别。

2. （可选）创建数据库 coredump 目录。

```
mkdir -p /home/vexdb/data/db_coredump
chmod 770 /home/vexdb/data
chown vexdb:vexdb /home/vexdb/data
```

3. （可选）创建数据库数据目录（可自定义）。如果不执行，则安装数据库时将采用默认安装目录。

```
mkdir -p /home/vexdb/data/vexdb
chmod 700 /home/vexdb/data/vexdb
chown -R vexdb:vexdb /home/vexdb/data/vexdb
```

4. （可选）创建数据库软件目录（可自定义）。如果不执行，则安装数据库时将采用默认安装目录。

```
mkdir -p /home/vexdb/local/vexdb
chown -R vexdb:vexdb /home/vexdb
```

步骤2：修改资源限制

以 root 用户登录操作系统。

执行 `vi /etc/security/limits.conf`，在文件末尾添加如下内容，保存退出。

```
vexdb soft nproc unlimited
vexdb hard nproc unlimited
vexdb soft stack unlimited
vexdb hard stack unlimited
vexdb soft core unlimited
vexdb hard core unlimited
vexdb soft memlock unlimited
vexdb hard memlock unlimited
vexdb soft nofile 1024000
vexdb hard nofile 1024000
```

步骤3：解压安装包

以 root 用户登录操作系统。

创建目录（/soft/vb），解压安装包，以root用户赋予数据库安装用户 vexdb 操作的权限。

安装包以（本安装以 `VexDB-Developer-Edition-3.0_Build0_commitid-Linux-x86_64-no_mot-datetime.tar.gz` 为例，实际安装以获取安装包名称为准）和 license 文件上传到/soft/vb（路径可自定义，本安装步骤以 /soft/vb 为例）。

```
mkdir -p /soft/vb
cd /soft/vb
tar -xvf VexDB-Developer-Edition-3.0_Build0_commitid-Linux-x86_64-no_mot-datetime.tar.gz
chown -R vexdb:vexdb /soft/vb/
chmod -R 775 /soft/vb/
```

> **说明**
>
> - root 用户对 soft 目录的权限至少设置为 755，否则切换到其他用户后，无法访问 soft 目录。
> - 二进制安装程序与安装包需放于同一路径。

步骤4：运行安装程序

执行安装脚本

1. 切换到数据库安装用户 vexdb。

   安装程序不能由操作系统超级用户执行。

   ```
   su - vexdb
   ```

2. 运行安装程序。

   ```
   cd /soft/vb/vexdb-installer/
   ./vexdb_installer
   ```

交互步骤

根据回显信息进行交互，直至安装程序完成。

1. 安装环境检查。

```
================================================================================

欢迎使用安装工具(V1.0)，下面开始安装VexDB.

================================================================================
检查安装包是否完整
--------------

ok
================================================================================

输入<Enter>继续：
```

2. 系统配置信息。

```
================================================================================
系统配置信息
--------------

      操作系统 : CentOS Linux 7 (Core)
       CPU核数 : 8
      内存大小 : 11852 MB
    当前用户名 : vexdb

输入<Enter>继续：
```

3. 依赖检查（检查服务器是否已经安装需要的依赖包）。

```
================================================================================
依赖检查
--------------

  readline : 6.2
    python : 2.7.5
     libicu : 50.2
  cracklib : 2.9.0
   libxslt : 1.1.28
       tcl : 8.5.13
      perl : 5.16.3
  openldap : 2.4.44
       pam : 1.1.8
systemd-libs : 219
     bzip2 : 1.0.6
   gettext : 0.19.8.1
    libaio : 0.3.109
ncurses-libs : 5.9

输入<Enter>继续:

-----------------
准备安装环境...

准备安装环境结束
```

4. IPC 参数检查。

   若检查通过，自动跳转下一步，否则根据提示进行设置即可。

```
================================================================================
IPC参数检查
--------------

IPC参数检查完成
```

5. 选择是否进行实例化安装。

   若进行实例化安装则选 `Y`，若进行非实例化安装则选 `N`。

```
================================================================================
安装数据库
--------------

是否需要实例化数据库(Y/N): y
```

6. 选择安装类型（选 `2` 或按 `Enter`）。

   - 典型安装：使用默认参数配置初始化数据库。

- 自定义安装：手动配置安装参数和功能。

```
选择安装类型

典型安装     : 使用默认参数配置初始化数据库
自定义安装   : 手动配置安装参数和功能

  -> 1- 典型安装
     2- 自定义安装

选择安装类型，或者输入<Enter>选择默认值(1):
```

7. （非实例化安装无此步骤）设置超级管理员密码，需要输入密码，并再次输入密码确认（设置的密码最少包含 8 个字符，最多包含 16 个字符。密码由大小写字母加数字组成，例如：`aA123***`）。

```
================================================================================
数据库初始化用户密码(按下 退格 键进行回退)
---------------

输入数据库初始化用户(vexdb)密码: *********

请再次输入密码: *********
```

8. （非实例化安装无此步骤）设置密钥（选 `1` 或按 `Enter`）。

```
================================================================================
数据库加密密钥(PGENCRYPTIONKEY)
---------------

数据库加密密钥设置:

  ->  1-    使用数据库初始化用户密码(默认)
      2-    手动输入加密密钥

请选择数据库加密密钥设置方式,或者输入<Enter>选择默认值(1):
```

9. 设置数据库安装路径（输入步骤2创建的数据库安装路径（/soft/vb），或者输入<回车>使用默认路径（默认路径：/home/vexdb/local/vexdb））。

   此处软件安装目录不能与数据库目录相同，目录相同时会自动创建子目录用于分开存放软件和数据。

```
================================================================================
VexDB软件安装目录
--------------

VexDB软件安装目录
  默认位置: /home/vexdb/local/vexdb

输入绝对路径(按下 ctrl+退格 进行回退)，或者输入<Enter>使用默认路径:


================================================================================
数据库初始化目录
--------------

选择数据库目录  默认位置: /home/vexdb/data/vexdb

输入绝对路径(按下 ctrl+退格 进行回退)，或者输入<Enter>使用默认路径:
```

10. （非实例化安装无此步骤）磁盘 IO 调度算法检查。

    为避免影响数据库性能，生产环境建议采用 deadline 策略，若检查出非 deadline 策略，可按提示进行修改，再重新安装，也可以直接跳过继续安装。

```
================================================================================
磁盘IO调度算法检查
--------------

开始检查目录/home/vexdb/data/vexdb所属的磁盘IO调度算法

磁盘IO调度算法检查完成
```

11. 初始化阶段。

```
================================================================================
安装概要
---------------

VexDB软件安装目录:
    /home/vexdb/local/vexdb

数据库目录:
    /home/vexdb/data/vexdb

数据库初始化用户:
    vexdb

数据库初始化参数:
   listen_addresses='*'
   max_connections=500
   shared_buffers=2963MB
   max_process_memory=7881MB
   work_mem=4MB


输入<Enter>继续:


正在安装，请稍后...
初始化数据库成功，数据目录: /home/vexdb/data/vexdb
```

12. 安装数据库过程会生成随机口令作为管理员初始口令，显示信息如下（非实例化安装无此步骤）：

```
数据库三个默认管理员vbaudit、vbsso、vbadmin的默认口令为:
系统管理员[vbadmin] 初始密码: Z71a7b5#
安全管理员[vbsso] 初始密码: Xeb06!7b
审计管理员[vbaudit] 初始密码: X1bfa!65
```

13. 选择是否添加许可。

```
================================================================================
加载正式License
---------------

是否加载正式License(Y/N): n
```

> <img src="/Users/Four/JobProjects/vexdb/vexdb-doc/img/note.png" > 说明
>
> VexDB 开发版提供了时长为 1 年的临时许可。即使不添加正式许可，也可以在安装完成后免费试用 VexDB。
临时许可即将过期前，在数据库管理员用户和数据库初始化用户使用 vsql 连接数据库时，VexDB 会提示许可期限的有关信息。提醒天数的阈值是通过参数 vb

14. 安装完成。

```
================================================================================
安装完成
--------------

请先指定license路径(license路径错误会导致启动数据库失败):
    请将license路径写入文件 /home/vexdb/data/vexdb/postgresql.conf, 形式为 license_path='license路径'

初始化数据库运行环境:
    source ~/.bashrc

启用、停止、重启数据库:
    vb_ctl <start/stop/restart>


已安装完成,输入<Enter>退出:
```

步骤5：初始化环境变量

1. 以 vexdb 用户登录操作系统。
2. 执行如下命令初始化数据库环境变量。

```
source ~/.bashrc
```

步骤6：初始化数据库实例（可选）

运行安装程序后，在交互时若选择了**非实例化安装**，对于用户需要创建数据库实例的情况，应通过 vb_initdb 初始化数据库。

1. 以 vexdb 用户登录操作系统。
2. 使用 vb_initdb 工具初始化数据库。

```
vb_initdb  -D /home/vexdb/data/vexdb --nodename vexdb -w Vbase@123
```

上述语句的参数说明如下：

参数名	说明
-D	指定数据目录，后跟的 /home/vexdb/data/vexdb 为数据库目录，可自定义。注意：需存在且在 vexdb 用户下，需要 0700 权限。
–nodename	指定初始化的节点名称。
-w	指定管理员用户的密码。

步骤7：设置 Core_Pattern

为记录数据库异常停机信息，执行如下命令设定 Core_pattern 路径为 vexdb 组用户可写的路径:

1. 以 root 用户登录操作系统。
2. 执行如下操作完成设置。

```
echo "/omTmp/corefile/core-%e-%p-%t" > /proc/sys/kernel/core_pattern
```

- 本语句中/omTmp/corefile/为系统自动创建的默认coredump目录。
- 用户可手动创建的 coredump 目录（此目录应提前创建，且 vexdb 组用户可写），此时需对应修改 echo 语句中路径：echo “自定义目录/core-%e-%p-%t” > /proc/sys/kernel/core_pattern。

卸载程序

卸载 VexDB 的过程包含卸载 VexDB 和对 VexDB 服务器的环境清理。

前提条件： 关闭数据库。

```
vb_ctl stop
```

卸载VexDB

步骤1 以数据库安装用户vexdb登录数据库节点。

```
su - vexdb
```

步骤2 进入安装程序所在目录。

```
cd /soft/vb/vexdb-installer/
```

步骤3 执行卸载命令。

```
./vexdb_installer --uninstall
```

卸载完成后，已安装的 VexDB 程序（即数据库安装目录 $GAUSSHOME 对应目录）和 VexDB 相关环境变量将被删除，仅保留数据库实例。

参考链接

VexDB 集群安装和使用的文档请查阅 高可用集群。

镜像安装（单机）

平台适配

当前，支持镜像安装的平台包括：

操作系统	CPU 架构
所有 Linux 操作系统	Amd64
	Arm64

下载镜像

使用 VexDB 提供的官方镜像。使用命令 `docker pull` 来下载镜像。

其中，`tagname` 用于标识同一个镜像的不同版本，请查阅 DockerHub 获取当前支持的 Tag 版本。

```
docker pull shuzhiyinhang/vexdb:tagname
```

启动容器

启动 VexDB 实例：

```
docker run --name vexdb --privileged=true -d -e GS_PASSWORD=Vexdb@123 -e DBCOMPATIBILITY=A -e GS_USERNAME=vexdb shuzhiyinhang/vexdb:tagname -v ./data:/home/postgres/data -v ./etc:/home/postgres/vexdb/etc sh
```

`docker run` 参数说明：

- –name：给容器指定一个名称。
- –privileged=true：开启特权模式，赋予容器几乎所有的内核权限和能力，让容器几乎可以像宿主机上的进程一样访问所有硬件和系统资源。
- -d: 后台运行容器并返回容器 ID。
- -e: 设置环境变量。

- GS_PASSWORD：表示数据库初始化用户的密码。长度至少为8位，且必须包含大写字母、小写字母、数字、符号中的三种字符。
- GS_USERNAME：自动创建的用户名称，和初始用户共享密码。
- DBCOMPATIBILITY：表示数据库兼容模式，VexDB 默认的数据库兼容模式是 A，即 Oracle 兼容模式。

- -v：数据库文件默认存储在容器内，一旦更新镜像或删除容器，会导致数据丢失。建议通过 `-v` 参数将数据持久化到本地存储。

连接数据库

使用 VexDB 提供的命令行工具 vsql，打开一个数据库的客户端连接。

```
# 外部连接命令
vsql -d postgres -U vexdb -W 'Vexdb@123' -h host_ip -p 15432
# 容器内部访问
docker exec -it -u postgres vexdb bash
vsql
```

vsql 参数说明请参考 vsql 工具。

安装过程 FAQ

设置防火墙白名单

（root 用户执行）

1. 新增端口至防火墙白名单。

   ```
   firewall-cmd --zone=public --permanent --add-port=5432/tcp
   ```

2. 重新加载防火墙白名单。

   ```
   firewall-cmd --reload
   ```

3. 查看端口列表。

   ```
   firewall-cmd --list-port
   ```

   下列各 IP 端口均为各服务的默认端口，用户应根据实际的的规划做对应替换。

服务名称	端口号	说明
VexDB	5432	数据库服务端口
	5433（数据库服务端口+1）	数据库内部工具使用端口
ntp	123	ntp 默认端口
HAS	55434	集群本地监听端口
	55435	集群心跳端口
	55436	集群服务端口
	8008	集群通讯端口

以数据库服务端口 5432 为例，开放端口的命令如下：

```
firewall-cmd --zone=public --permanent --add-port=5432/tcp
firewall-cmd -reload
firewall-cmd -list-ports
```

内核参数介绍

fs.aio-max-nr

该参数表示可以拥有的异步 IO 请求数目。

推荐值：1048576。

在 Linux 6 和 7 上使用 aio 时需要设置 fs.aio-max-nr 来适应异步 IO。

设置语句：

```
sysctl fs.aio-max-nr='1048576'
```

file-max

该参数表示每次登录会话可以打开的文件数。

在 Linux 6 和 7 上，file-max 的值与内核资源使用参数 max_files_per_process 对应。

设置语句：

```
sysctl fs.file-max='76724600'
```

kernel.sem

该参数包含 4 个参数（依次为 SEMMSL、SEMMNS、SEMOPM、SEMMNI）。

- SEMMSL：每个信号量 set 中信号量最大个数，最小取值 250。
- SEMMNS：Linux 系统中信号量最大个数，至少取值 32000，等于 SEMMSL*SEMMNI。
- SEMOPM：SEMOP 系统调用允许的信号量最大个数，至少取值 100；或者等于 SEMMSL。
- SEMMNI：Linux 系统信号量 set 最大个数，最少 128。

> **说明**
>
> - 使用 `ipcs -l` 或 `ipcs -u` 查看信号量。
> - 每 16 个进程一组，每组信号量需要 17 个信号量。可根据需要修改 kernel.sem 值。
> - SEMMSL 代表信号量，SEMMNI 代表组。

设置语句：

```
sysctl kernel.sem='4096 2097152000 4096 512000'
```

kernel.shmall

该参数用于控制共享内存页数，等于 `系统内存（建议设置为80%）/PAGE_SIZE`，该参数设置太小有可能导致数据库启动报错。

单位：byte

示例：物理内存为 128GB，PAGE 大小为 4096，则 `kernel.shmall=128*1024*1024*1024*0.8/4096` = `26843545`。

> **说明**
>
> - 使用 `getconf PAGE_SIZE` 命令查看 page_size 大小。
> - 内存大小需换算成 bytes。

设置语句：

```
sysctl kernel.shmall='26843545'
```

kernel.shmmax

该参数表示最大单个共享内存段大小（建议为大于 shared_buffer 值）， 等于 `系统内存*0.5` 。

单位：bytes

示例：物理内存为128GB，则： `kernel.shmmax=128*1024*1024*1024*0.5 = 68719476736` 。

> **说明**
>
> 内存大小需换算成 bytes。

设置语句：

```
sysctl kernel.shmmax='68719476736'
```

kernel.shmmni

该参数系统范围内共享内存段的最大数量。

默认值：4096。

> **说明**
>
> 每个 VexDB 数据库集群至少 2 个共享内存段。

设置语句：

```
sysctl kernel.shmmni='819200'
```

net.core.netdev_max_backlog

该参数表示允许送到队列的数据包的最大数目。

> **说明**
>
> iptables 防火墙链表相关。

设置语句：

```
sysctl net.core.netdev_max_backlog='10000'
```

net.core.rmem_default

该参数表示预留用于接收缓冲的内存默认值。

单位：bytes。

设置语句：

```
sysctl net.core.rmem_default='262144'
```

net.core.rmem_max

该参数表示预留用于接收缓冲的内存最大值。

单位：bytes。

设置语句：

```
sysctl net.core.rmem_max='4194304'
```

net.core.wmem_default

该参数表示预留用于发送缓冲的内存默认值。

单位：bytes。

设置语句：

```
sysctl net.core.wmem_default='262144'
```

net.core.wmem_max

该参数表示预留用于发送缓冲的内存最大值。

单位：bytes。

设置语句：

```
sysctl net.core.wmem_max='4194304'
```

net.core.somaxconn

该参数表示 socket 监听的 backlog（队列）上限。

默认值：128

设置语句：

```
sysctl net.core.somaxconn='4096'
```

net.ipv4.tcp_fin_timeout

该参数表示 FIN_WAIT_2 状态的超时时长。

单位：秒。

> **说明**
>
> 用于加快僵死进程回收速度。

设置语句：

```
sysctl net.ipv4.tcp_fin_timeout='5'
```

vm.dirty_background_bytes

该参数表示触发回刷的脏页数据量，超过该参数后脏页刷到磁盘。

单位：btyes。

设置语句：

```
sysctl vm.dirty_background_bytes='409600000'
```

vm.dirty_expire_centisecs

该参数表示脏数据的过期时间，超过该时间系统会将脏数据回写到磁盘上。

单位：百分之一秒。

> **说明**
>
> 比 vm.dirty_expire_centisecs 值旧的脏页，将被刷到磁盘。

以下命令中 3000 代表 30 秒。

设置语句：

```
sysctl vm.dirty_expire_centisecs='3000'
```

vm.dirty_ratio

该参数表示脏数据百分比，超过这个百分比，新的 IO 请求将会被阻挡，直到脏数据被写进磁盘。

设置语句：

```
sysctl vm.dirty_ratio='80'
```

vm.dirty_writeback_centisecs

该参数表示多长时间，后台刷脏页进程会被唤醒一次，检查是否有缓存需要清理。

单位：百分之一秒。参数设置为 100 代表 1 秒，如下命令中 50 代表 0.5 秒。

设置语句：

```
sysctl vm.dirty_writeback_centisecs='50'
```

vm.overcommit_memory

该参数表示内存分配策略，可选值：0、1、2。

- 0：表示内核将检查是否有足够的可用内存供应用进程使用，如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。
- 1：表示内核允许分配所有的物理内存，而不管当前的内存状态如何。
- 2：表示内核允许分配超过所有物理内存和交换空间总和的内存。

> **说明**
>
> 在分配内存时 vm.overcommit_memory 设置为 0，vm.overcommit_ratio 参数可以不设置。

设置语句：

```
sysctl vm.overcommit_memory='0'
```

vm.swappiness

该参数表示激活交换之前可用内存的百分比。数值越低，使用的交换越少，并且物理内存中保留的内存页越多。

默认值：60

> **说明**
>
> 设置为 0 时表示关闭交换分区，数据库服务器不建议使用 swap，0 值仅在极限测试时使用。

设置语句：

```
sysctl vm.swappiness='60'
```

net.ipv4.ip_local_port_range

该参数可以设置本地动态端口分配范围，防止占用监听端口。

设置语句：

```
sysctl net.ipv4.ip_local_port_range='40000 65535'
```

fs.nr_open

该参数表示单个进程可分配的最大文件数。对于有很多对象（表、视图、索引、序列和物化视图等）的数据库，建议设置为 2000 万。

单位：个

> **说明**
>
> 建议设置为 2000 万。

设置语句：

```
sysctl fs.nr_open='20000000'
```

添加排序规则

在除凝思之外的其他 Linux 操作系统中，添加语言排序规则方法如下：

（root 用户执行）

1. 执行如下命令安装 locales 包。

   ```
   apt-get install locales
   ```

2. 执行如下命令，安装新的语言排序规则。

```
dpkg-reconfigure locales
```

也可使用如下命令安装 en_US.utf8。

```
localedef -v -c -i en_US -f UTF-8 en_US.UTF-8
```

VexDB 启停与状态查看

vb_ctl 是 VexDB 提供的数据库服务控制工具，可以用来启停数据库服务和查询数据库状态。详细说明请查阅管理员指南 vb_ctl 工具。

成功安装数据库之后，需要启动数据库服务：

```
vb_ctl start
```

其他命令

根据需要，用户可以使用以下命令停止或重启数据库服务，并查询数据库状态。

- 停止数据库服务：

```
vb_ctl stop
```

- 重启数据库服务：

```
vb_ctl restart
```

- 显示数据库状态：

```
vb_ctl status
```

连接 VexDB

- 配置远程客户端连接
- 使用 vsql 连接
- 使用应用程序接口连接

配置远程连接数据库（可选）

数据库安装并启动后，用户可以在数据库服务器本地通过客户端工具直接建立本地连接。如果需要通过主机的 IP 地址或者域名连接到远程数据库服务器，需要配置客户端接入认证策略。

配置文件 pg_hba.conf 位于数据库的数据目录 $PGDATA 中。

操作步骤

1. 以安装 VexDB 的操作系统用户身份，登录数据库实例所在主机。
2. 打开配置文件 pg_hba.conf。

```
vi $PGDATA/pg_hba.conf
```

3. 配置认证策略。

   pg_hba.conf 文件格式中一行对应一个认证规则，一条记录不能跨行存在。空白行和以 # 开头（即被注释）的行被忽略。

   pg_hba.conf 文件中的每条记录可以是下面四种格式之一：

```
local     DATABASE USER METHOD [OPTIONS]
host      DATABASE USER ADDRESS METHOD [OPTIONS]
hostssl   DATABASE USER ADDRESS METHOD [OPTIONS]
hostnossl DATABASE USER ADDRESS METHOD [OPTIONS]
```

   例如以下配置示例，表示允许来自 IP 地址 xx.xx.xx.xx 的客户端通过 mytest 用户连接到所有数据库，32 表示子网掩码为 1 的位数，即 255.255.255.255。连接方式为 trust，即不需要密码验证。

```
host all mytest xx.xx.xx.xx/32 trust
```

4. 保存文件并使用以下命令重启数据库，使修改生效。

```
vb_ctl restart
```

配置文件参考

参数名称	描述	取值范围
local	表示这条记录只接受通过 Unix 域套接字进行的连接。没有这种类型的记录，就不允许 Unix 域套接字的连接。只有在从服务器本机使用 vsql 连接且在不指定-U 参数的情况下，才是通过 Unix 域套接字连接。	-
host	表示这条记录既接受一个普通的 TCP/IP 套接字连接，也接受一个经过 SSL 加密的 TCP/IP 套接字连接。	连接地址
hostssl	表示这条记录只接受一个经过 SSL 加密的 TCP/IP 套接字连接。	用 SSL 进行安全的连接。。
hostnossl	表示这条记录只接受一个普通的 TCP/IP 套接字连接。	-
DATABASE	声明记录所匹配且允许访问的数据库。	• all：表示该记录匹配所有数据库。 • sameuser：表示如果请求访问的数据库和请求的用户同名，则匹配。 • samerole：表示请求的用户必须是与数据库同名角色中的成员。 • samegroup：与 samerole 作用完全一致，表示请求的用户必须是与数据库同名角色中的成员。 • 一个包含数据库名的文件或者文件中的数据库列表：文件可以通过在文件名前面加前缀@来声明。文件中的数据库列表以逗号或者换行符分隔。 • 特定的数据库名称或者用逗号分隔的数据库列表。 说明：值 replication 表示如果请求一个复制链接，则匹配，但复制链接不表示任何特定的数据库。如需使用名为 replication 的数据库，需在 database 列使用记录“replication”作为数据库名。
USER	声明记录所匹配且允许访问的数据库用户。	• all：表明该记录匹配所有用户。 • +用户角色：表示匹配任何直接或者间接属于这个角色的成员。说明： + 表示前缀符号。 • 一个包含用户名的文件或者文件中的用户列表：文件可以通过在文件名前面加前缀@来声明。文件中的用户列表以逗号或者换行符分隔。 • 特定的数据库用户名或者用逗号分隔的用户列表。
ADDRESS	指定与记录匹配且允许访问的 IP 地址范围。	支持 IPv4和 IPv6，可以使用如下两种形式来表示： IP 地址/掩码长度。例如，10.10.0.0/24 IP 地址子网掩码。例如，10.10.0.0 255.255.255.0 说明以 IPv4格式给出的 IP 地址会匹配那些拥有对应地址的 IPv6连接，比如127.0.0.1将匹配 IPv6地址 ::ffff:127.0.0.1
METHOD	声明连接时使用的认证方法。	本产品支持如下几种认证方式，详细解释请参考表格认证方式：trustrejectmd5（不推荐使用，默认不支持，可通过password_encryption_type参数配置sha256certgss（仅用于 VexDB 内部节点间认证）sm3scram-sha256ldap

认证方式

认证方式	说明
trust	采用这种认证模式时，本产品只完全信任从服务器本机使用 vsql 且不指定-U 参数的连接，此时不需要口令。trust 认证对于单用户工作站的本地连接是非常合适和方便的，通常不适用于多用户环境。如果想使用这种认证方法，可利用文件系统权限限制对服务器的 Unix 域套接字文件的访问。要使用这种限制有两个方法： • 设置参数unix_socket_permissions和unix_socket_group。 • 设置参数unix_socket_directory，将 Unix 域套接字文件放在一个经过恰当限制的目录里。 须知： 设置文件系统权限只能 Unix 域套接字连接，它不会限制本地 TCP/IP 连接。为保证本地 TCP/IP 安全，VexDB 不允许远程连接使用 trust 认证方法。
reject	无条件地拒绝连接。常用于过滤某些主机。
md5	要求客户端提供一个 md5加密的口令进行认证。 不推荐使用 md5认证，因为 md5为不安全的加密算法，存在网络安全风险。VexDB 保留 md5认证和密码存储，是为了便于第三方工具的使用（比如 TPCC 评测工具）。
sha256	要求客户端提供一个 sha256算法加密的口令进行认证，该口令在传送过程中结合 salt（服务器发送给客户端的随机数）的单向 sha256加密，增强了安全性。
cert	客户端证书认证模式，此模式需进行 SSL 连接配置且需要客户端提供有效的 SSL 证书，不需要提供用户密码。 该认证方式只支持 hostssl 类型的规则。
gss	使用基于 gssapi 的 kerberos 认证。 该认证方式依赖 kerberos server 等组件，仅支持 VexDB 内部通信认证。当前版本暂不支持外部客户端通过 kerberos 认证连接。 开启 VexDB 内部kerberos 认证会使增加内部节点建连时间，即影响首次涉及内部建连的 SQL 操作性能，内部连接建立好后， 后续操作不受影响。
sm3	国密 SM3算法，目前只支持 vsql、JDBC、ODBC 三种连接方式
scram-sha256	支持 SCRAM 认证，SCRAM 是一种基于盐（salt）和挑战-响应（challenge-response）的密码认证协议。
ldap	客户端使用 LDAP 服务器进行登录认证。配置编译时加上–with-ldap选项。假设 ldap 服务器 IP 为192.168.1.10，在 pg_hba.conf 中配置 ldap 认证： local all all ldap ldapserver=192.168.1.10 ldapprefix=“uid=” ldapsuffix=“,ou=People,o=System Support,dc=my-domain,dc=com” host all all 127.0.0.1/32 ldap ldapserver=192.168.1.10 ldapprefix=“uid=” ldapsuffix=“,ou=People,o=System Support,dc=my-domain,dc=com”

使用 vsql 连接

命令行工具 vsql 是 VexDB 提供的客户端连接工具。vsql 除了具备操作数据库的基本功能，还提供了若干高级特性，便于用户使用。本节仅介绍如何使用 vsql 连接数据库，关于 vsql 的更多使用方法请查阅 vsql 工具。

前提条件

已确认连接信息，包括连接的IP、数据库名称与数据库端口。

具体步骤

1. 以安装 VexDB 的操作系统用户身份，登录数据库实例所在主机。
2. 运行以下命令连接数据库。数据库安装完成后，存在一个默认数据库 postgres。第一次连接数据库时可以连接到此数据库。

   ```
   vsql -d postgres -p 5432 [-h xx.xx.xx.xx]
   ```

   其中：

   - -d：指定需要连接的数据库名称。
   - -p ： 指定数据库的端口号。
   - -h ： 指定正在运行服务器的主机名或者 Unix 域套接字的路径。
     - 若进行本地连接，则指定本地主机或不指定-h。
     - 若进行远程连接，则指定远程客户端的IP地址。
   - 可以使用 -U 指定连接数据库的用户。未指定连接使用的用户身份时，默认使用数据库初始用户（与安装 VexDB 的操作系统用户同名）进行连接。

   连接成功后，系统显示类似如下信息：

   ```
   vsql((VexDB Developer Edition {version}) compiled at {datetime} commit {commit_id} last mr  )
   Non-SSL connection (SSL connection is recommended when requiring high-security)
   Type "help" for help.

   postgres=>
   ```

   这表示用户现在可以交互地键入 SQL 命令了，它们将发送给 VexDB 服务器，并能够在屏幕上显示 SQL 或命令的结果。
3. 执行断开客户端连接的 vsql 元命令。

   ```
   \q
   ```

使用应用程序接口连接数据库

如果您希望通过驱动程序访问并操作 VexDB，请查阅 API 参考 中的介绍进行配置。

开始使用 VexDB

本文介绍使用 VexDB 时的基本操作。

在开始使用数据库之前，确保您已经顺利安装、启动并连接到数据库。

通过阅读本节，您可以完成以下操作：

- 创建数据库
- 创建表
- 管理表
  - 向表中插入数据
  - 更新表中数据
  - 查看数据
  - 删除表中数据
- 查看数据库对象

创建数据库

数据库安装完成后，默认包含名称为 postgres 的数据库。

- 创建一个新的数据库 testdb1。

  ```
  CREATE DATABASE testdb1;
  ```

  新的数据库默认将通过复制标准系统数据库模板 template0 来创建。

- 创建数据库 testdb2，并指定所有者为 dbuser。

  步骤1：创建用户。

  ```
  CREATE USER dbuser PASSWORD Vbase@123;
  ```

  步骤2：创建数据库。

  ```
  CREATE DATABASE testdb2 OWNER dbuser TEMPLATE template0;
  ```

- 查看数据库列表（vsql 客户端可用）

```
\l
```

创建表

以下简单介绍如何在数据库中创建表。

- 创建普通表。

```
CREATE TABLE customer_t1
(
c_customer_sk integer not null,
c_customer_id char(6) not null,
c_first_name varchar(20) ,
c_last_name varchar(20) ,
c_country varchar(20)
);
```

- 创建表，并指定 c_state 字段的缺省值为 GA。

```
CREATE TABLE customer_t2
(
c_customer_sk integer not null,
c_customer_id char(6) not null,
c_first_name varchar(20) ,
c_last_name varchar(20) ,
c_state CHAR(2) DEFAULT 'GA',
c_country varchar(20)
);
```

- 执行元命令列举表。常见的查看对象元命令可参考查看对象。

```
\dt
```

- 执行元命令查看表结构。

```
\d customer_t2
```

- 创建一个有主键约束的表。

```
CREATE TABLE customer_t3
(
c_customer_sk integer PRIMARY KEY,
c_customer_id char(6) not null,
c_first_name varchar(20) ,
c_last_name varchar(20) ,
c_country varchar(20)
);
```

- 向表中增加一个 varchar 列。

```
ALTER TABLE customer_t1 ADD c_company varchar(30);
```

- 删除表

```
DROP TABLE customer_t1;
DROP TABLE customer_t2;
DROP TABLE customer_t3;
```

管理表

以下简单介绍如何向表中插入数据和更新表中数据等相关操作。

向表中插入数据

在创建一个表后，表中并没有数据，在使用这个表之前，需要向表中插入数据。

1. 创建表 customer36。

```
CREATE TABLE customer36
(
c_customer_sk integer,
c_customer_id char(5),
c_first_name char(6));
```

2. 向表 customer36 中插入一行数据。
   - 方法一：数据值是按照这些字段在表中出现的顺序列出的，并且用逗号分隔。通常数据值是文本（常量），但也允许使用标量表达式。

     ```
     INSERT INTO customer36(c_customer_sk, c_customer_id, c_first_name)
     VALUES (3769, 'hello', 'Grace');
     ```

   - 方法二：如果已经知道表中字段的顺序，也可无需列出表中的字段。

```
INSERT INTO customer36 VALUES (3769, 'hello', 'Grace');
```

- 方法三：如果不知道所有字段的数值，可以忽略其中的一些。没有数值的字段将被填充为字段的缺省值。

```
INSERT INTO customer36 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');
```

3. 向表中插入多行数据。

```
INSERT INTO customer36 (c_customer_sk, c_customer_id, c_first_name) VALUES(6885, 'maps', 'Joes'), (4321, 'tpcds', 'Lily'), (9527, 'world', 'James');
```

如果需要向表中插入多条数据，除此命令外，也可以多次执行插入一行数据命令实现，但是建议使用此命令可以提升效率。

4. 从指定表插入数据到当前表。例如，在数据库中创建了一个表 customer36 的备份表 customer37，现在需要将表 customer36 中的数据插入到表 customer37 中，则可以执行如下命令。

```
CREATE TABLE customer37
(
c_customer_sk integer,
c_customer_id char(5),
c_first_name char(6)
);

INSERT INTO customer37 SELECT * FROM customer36;
```

更新表中数据

根据需要，可以更新单独一行，所有行或者指定部分行的表中数据。还可以独立更新每个字段，其他字段则不受影响。

> **说明**
>
> SQL 通常不会为数据行提供唯一标识，因此无法直接声明需要更新哪一行，但可以声明一个被更新的行必须满足的条件。如果表中有一个主键，可以指定准确的行。

1. 更新表 customer36 中 c_customer_sk 为 9527 的地域为9876。

```
UPDATE customer36 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;
```

2. 更新表中 c_customer_sk 字段的所有值为增加 100。

```
UPDATE customer36 SET c_customer_sk = c_customer_sk + 100;
```

在这里省略了 WHERE 子句，表示表中的所有行都要被更新。如果出现了 WHERE 子句，那么只有匹配其条件的行才会被更新。在 SET 子句中的等号是一个赋值，而在 WHERE 子句中的等号是比较。WHERE 条件不一定是相等测试，许多其他的操作符也可以使用。

3. 更新表中的多个字段。

```
UPDATE customer36 SET c_customer_id = 'Admin', c_first_name = 'Local' WHERE c_customer_sk = 4421;
```

查看数据

在查看数据之前，确保已经完成本文示例中向表中插入数据和更新表中数据的操作。

1. 查询表 customer36 的数据行数。

```
SELECT count(*) FROM customer36;
```

返回结果如下：

```
count
-------
    6
(1 row)
```

2. 查询表 customer36 的所有数据。如果没有完成向表中插入数据中重复插入数据的操作，以下步骤的返回结果可能略有差异。

```
SELECT * FROM customer36;
```

返回结果如下：

```
c_customer_sk | c_customer_id | c_first_name
--------------+---------------+--------------
         3869 | hello         | Grace
         3869 | hello         | Grace
         3869 |               | Grace
         6985 | maps          | Joes
         9976 | world         | James
         4421 | Admin         | Local
(6 rows)
```

3. 查询字段 c_customer_sk 的数据。

```
SELECT c_customer_sk FROM customer36;
```

返回结果如下：

```
c_customer_sk
---------------
        3869
        3869
        3869
        6985
        9976
        4421
(6 rows)
```

4. 过滤字段 c_customer_sk 的重复数据。

```
SELECT DISTINCT( c_customer_sk ) FROM customer36;
```

返回结果如下：

```
c_customer_sk
---------------
        9976
        6985
        3869
        4421
(4 rows)
```

5. 查询字段 c_customer_sk 为 3869 的所有数据。

```
SELECT * FROM customer36 WHERE c_customer_sk = 3869;
```

返回结果如下：

```
c_customer_sk | c_customer_id | c_first_name
---------------+---------------+--------------
        3869 | hello         | Grace
        3869 | hello         | Grace
        3869 |               | Grace
(3 rows)
```

6. 按照字段 c_customer_sk 进行排序。

```
SELECT * FROM customer36 ORDER BY c_customer_sk;
```

返回结果如下：

```
 c_customer_sk | c_customer_id | c_first_name
---------------+---------------+--------------
          3869 | hello         | Grace
          3869 | hello         | Grace
          3869 |               | Grace
          4421 | Admin         | Local
          6985 | maps          | Joes
          9976 | world         | James
 (6 rows)
```

删除表中数据

在删除表中数据之前，确保已经完成本文示例中向表中插入数据和更新表中数据的操作。

根据需要可以删除匹配条件的一组行或者一次删除表中的所有行。

> **说明**
>
> SQL不能直接访问独立的行，只能通过声明被删除的行匹配的条件。如果表中有一个主键，可以指定准确的行。

1. 删除表customer36中所有c_customer_sk为3869的记录。

```
DELETE FROM customer36 WHERE c_customer_sk = 3869;
```

2. 删除表中所有的行。建议使用truncate，详情可参考TRUNCATE。

```
DELETE FROM customer37;
```

或者

```
TRUNCATE TABLE customer37;
```

3. 清理测试环境。

```
DROP TABLE customer36,customer37;
```

查看数据库对象

vsql 工具提供了元命令可帮助管理员查看数据库对象信息。有关元命令的详细信息，可查阅元命令参考。

常用的查看对象元命令如下：

查看帮助信息

```
\?
```

查看数据库

```
\l
```

列举表

```
\dt
```

查看表结构

```
\d tablename
```

列举 schema

```
\dn
```

查看索引

```
\di
```

切换数据库

```
\c dbname
```

向量检索指南

- VexDB 中，使用专门的 floatvector 数据类型来存储向量数据，提供了多种向量函数和操作符来执行向量之间的计算。
- VexDB 通过各种近似搜索算法实现了不同的向量检索索引；支持向标联合索引HybridANN 索引，允许在向量索引基础上额外增加对标量条件的过滤功能，实现了标量数据与向量数据的协同管理。
- 支持通过向量数据库参数来控制向量检索时的行为。

floatvector 类型

VexDB 支持 floatvector 类型，用于存储浮点类型的向量数据。每个向量是一个固定长度的数组，可以存储机器学习模型的嵌入向量、图像特征、文本的词嵌入等。

```
CREATE TABLE items (
id SERIAL PRIMARY KEY,
embedding floatvector(128) -- 存储一个 128 维的向量
);
```

- 一个表可以包含1个或者多个floatvector字段。
- 创建 floatvector 类型时，需要指定向量的维度，例如 floatvector(128) 表示每个向量有 128 个元素。目前支持的向量维度取值范围为[1,16384]。

类型转换

- array -> floatvector

```
SELECT ARRAY[1,1,1]::floatvector;
```

- string ->floatvector

```
SELECT '[1,1,1]'::floatvector;
```

- text /char(n) / nchar(n)/ varchar(n) / nvarchar(n) / clob / bpchar -> floatvector

```
SELECT '[1.0]'floatvector AS vector_example;
SELECT '[1.0]'::char(5)::floatvector AS vector_example;
SELECT '[1.0]'::nchar(5)::floatvector AS vector_example;
SELECT '[1.0]'::varchar(5)::floatvector AS vector_example;
SELECT '[1.0]'::nvarchar(5)::floatvector AS vector_example;
SELECT '[1.0]'floatvector AS vector_example;
SELECT '[1.0]'::bpchar(5)::floatvector AS vector_example;
```

示例

1. 创建一个含向量列的新表。

```
CREATE TABLE items (id bigserial PRIMARY KEY, embedding floatvector(3));
```

2. 插入向量数据。

```
INSERT INTO items (embedding) VALUES ('[1,2,3]'), ('[4,5,6]');
```

3. UPSERT 向量数据。

```
INSERT INTO items (id, embedding)
VALUES
(1, '[1, 2, 4]'),
(2, '[4, 5, 7]')
ON CONFLICT (id)
DO UPDATE SET embedding = EXCLUDED.embedding;
```

4. 更新向量数据。

```
UPDATE items SET embedding = '[1, 2, 8]' WHERE id =1;
```

5. 删除向量数据。

```
DELETE FROM items WHERE id =1;
```

6. 向己存在的表中添加一个向量列。

```
ALTER TABLE items ADD COLUMN embedding1 floatvector (3);
```

向量操作符

VexDB 提供了多种操作符来执行向量之间的计算。

<->

计算两个向量之间的欧几里德距离。欧几里德距离是多维空间中向量表示的点之间的直线距离。较小的欧几里德距离表示向量之间的相似性较大，因此该运算符在查找和排序相似项目时非常有用。

```
SELECT array[1,1,1,1]::floatvector <-> array[2,2,2,2]::floatvector AS value;
```

示例

- 获取与向量 [3,1,2] 距离最近的前5条记录（欧氏距离）：

```
SELECT * FROM items ORDER BY embedding <-> '[3, 1, 2]' LIMIT 5;
```

- 获取与 id=1 的向量最接近的Top5记录（欧氏距离）：

```
SELECT * FROM items WHERE id != 1 ORDER BY embedding <-> (SELECT embedding FROM items WHERE id = 1) LIMIT 5;
```

- 过滤：与向量 [3,1,2] 距离 < 5 的所有记录（欧氏距离）：

```
SELECT * FROM items WHERE embedding <-> '[3, 1, 2]' < 5;
```

- 计算欧式距离但不排序：

```
SELECT embedding <-> '[3, 1, 2]' AS distance FROM items;
```

<#>

计算两个向量的负内积。内积 = -1*负内积，内积表示两个向量在相同维度上的对应分量相乘后再求和的结果，内积越大表示方向越一致。

```
SELECT -1 * (array[1,1,1,1]::floatvector <#> array[2,2,2,2]::floatvector) AS value;
```

示例

- 计算向量与查询向量的内积（<#> 是 负内积操作符，乘 -1 得到正内积）：

```
SELECT (embedding <#> '[3, 1, 2]') * -1 AS inner_product FROM items;
```

<=>

计算两个向量之间的余弦距离。余弦相似度 = 1-余弦距离，比较两个向量的方向而不是它们的大小。余弦相似度的范围在 -1 到 1 之间，1 表示向量相同，0 表示无关，-1 表示向量指向相反方向。

```
SELECT 1 - (array[1,1,1,1]::floatvector <=>array[2,2,2,2]::floatvector) AS value;
```

示例

- 计算向量与查询向量的余弦相似度（余弦相似度 = 1 - 余弦距离）：

```
SELECT 1 - (embedding <=> '[3, 1, 2]') AS cosine_similarity FROM items;
```

+

两个向量逐元素相加。

```
SELECT array[1,1,1,1]::floatvector + array[2,2,2,2]::floatvector AS value;
```

-

两个向量逐元素相减。

```
SELECT array[1,1,1,1]::floatvector - array[2,2,2,2]::floatvector AS value;
```

@ ~ @

自然语言匹配操作符，用于匹配文本数据和自然语言文本，并支持返回相关性分数，仅支持在全文检索索引查询中使用。

类型支持：

- TEXT @ ~ @ TEXT
- VARCHAR @ ~ @ VARCHAR
- CHAR @ ~ @ CHAR
- TEXT[] @ ~ @ TEXT[]
- VARCHAR[] @ ~ @ VARCHAR[]
- CHAR[] @ ~ @ CHAR[]

@-@

关键词匹配操作符，用于匹配文本数据和关键词描述文本，仅支持在全文检索索引查询中使用。

关键词描述文本格式keywords为 `'<kw>[ AND/OR <keywords>]'` 或者 `'(<keywords>)'` ，支持关键词间AND/OR关系和括号。

类型支持：

- TEXT @-@ TEXT
- VARCHAR @-@ VARCHAR
- CHAR @-@ CHAR

- TEXT[] @-@ TEXT
- VARCHAR[] @-@ VARCHAR
- CHAR[] @-@ CHAR

向量运算函数

- **floatvector_combine(double precision[], double precision[])**

  这个函数用于将两个 double precision 类型的数组逐元素相加成一个新的向量，返回这个新向量。

```
SELECT floatvector_combine(ARRAY[1.0, 2.0, 3.0], array[4, 5,6]);
```

- **floatvector_accum(double precision[], floatvector)**

  这个函数用于将一个向量累加到数组中，返回一个新的数组。其中，被累加的数组及结果数组的第一个元素为累加次数，之后的元素为各维度的累积值。

```
SELECT floatvector_accum(ARRAY[1.0, 2.0, 3.0], '[4, 5]'::floatvector);
```

- **floatvector_cmp(floatvector, floatvector)**

  这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它判断大小关系。如果第一个向量小于第二个，返回 -1；如果相等，返回 0；如果第一个向量大于第二个，返回 1。

```
SELECT floatvector_cmp('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_gt(floatvector, floatvector)**

  这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量大于第二个向量，则返回 true，否则返回 false。

```
SELECT floatvector_gt ('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_ge(floatvector, floatvector)**

  这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量大于等于第二个向量，则返回 true，否则返回 false。

```
SELECT floatvector_ge ('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_ne(floatvector, floatvector)**

  这个函数用于比较两个向量，如果两个向量的任意元素不相等，则返回 true，否则返回 false。

```
SELECT floatvector_ne ('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_eq(floatvector, floatvector)**

  这个函数用于比较两个向量，如果两个向量的所有元素都相等，则返回 true，否则返回 false。

```
SELECT floatvector_eq ('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_le(floatvector, floatvector)**

  这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量小于等于第二个向量，则返回 true，否则返回 false。

```
SELECT floatvector_le('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_lt(floatvector, floatvector)**

  这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量小于于第二个向量，则返回 true，否则返回 false。

```
SELECT floatvector_lt ('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_spherical_distance(floatvector, floatvector)**

  这个函数用于计算两个向量之间的球面距离（spherical distance），即余弦相似度转换为归一化的角度比例。

```
SELECT floatvector_spherical_distance('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_negative_inner_product(floatvector, floatvector)**

  这个函数用于计算两个向量的负内积（negative inner product），即两个向量对应位置上的元素相乘后求和并取负值。

```
SELECT floatvector_negative_inner_product ('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_l2_squared_distance(floatvector, floatvector)**

  这个函数用于计算两个向量之间的 L2 范数的平方距离。L2 范数距离是向量元素差的平方和。

```
SELECT floatvector_l2_squared_distance('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- **floatvector_avg(double precision[])**

  这个函数用于计算一个 double precision 类型数组中所有元素的值进行 N 等分。N 是数组的第一个元素，并返回一个 N 等分后的向量。

```
SELECT floatvector_avg('{3, 4.56, 7.89}') AS value;
```

- floatvector_sub(floatvector, floatvector)

  这个函数用于计算两个向量的元素级相减，返回一个新的向量，其中每个元素是对应位置上两个向量元素的差。

```
SELECT floatvector_sub('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- floatvector_add(floatvector, floatvector)

  这个函数用于计算两个向量的元素级相加，返回一个新的向量，其中每个元素是对应位置上两个向量元素的和。

```
SELECT floatvector_add('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- floatvector_norm(floatvector)

  这个函数用于计算给定向量的范数，即向量元素的平方和的平方根。

```
SELECT floatvector_norm('[1,1,1,1]'::floatvector) AS value;
```

- floatvector_dims(floatvector)

  这个函数用于返回给定向量的维度（即向量中元素的数量）。

```
SELECT floatvector_dims ('[1,1,1,1]'::floatvector) AS value;
```

- l2_distance(floatvector, floatvector)

  这个函数用于计算两个向量之间的 L2 范数距离。L2 范数距离也称为欧氏距离，表示两个向量之间的直线距离。

```
SELECT l2_distance('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- inner_product(floatvector, floatvector)

  这个函数用于计算两个向量的内积。内积是两个向量对应元素乘积的和。

```
SELECT inner_product('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- cosine_distance(floatvector, floatvector)

  这个函数用于计算两个向量之间的余弦距离。余弦距离是通过计算两个向量之间的夹角余弦值来衡量它们之间的相似度。

```
SELECT cosine_distance('[1,1,1,1]'::floatvector ,'[2,2,2,2]'::floatvector) AS value;
```

- floatvector_to_float4(floatvector)

  这个函数用于将向量类型数据转为浮点数数组函数。

```
SELECT floatvector_to_float4 ('[2,2,2,2]'::floatvector) AS value;
```

状态查询函数

index_inspect(regclass[,partition])

描述： 查询索引内部数据构成信息。支持的索引类型包括：Graph_Index（HNSW）、fulltext、DiskANN。

> **说明**
>
> 该函数自V3.0.0.1版本开始支持。

参数说明： 输入索引名。

如果需要定位分区表中具体的索引信息，可以使用如下语句查询某一个具体分区索引的信息。

```
SELECT * FROM INDEX_INSPECT('<index_name>', '<index_partition_name>');
--或者
SELECT * FROM INDEX_INSPECT('<index_name>', '<index_partition_oid>');
```

其中，索引名可替换为，即index_inspect(oid[,partition])。索引分区名（或者OID）可以通过PG_PARTITION系统表获取。

返回值类型： SETOF record

输出的结果集包含两列，分别是attribute（text类型），表示信息属性；content（text类型），表示属性对应的数值。属性列表参考如下：

Graph_Index索引输出内容：

attribute 属性	说明
Used Space	索引占用磁盘空间，content 格式为xxx GB/MB/KB，精确到小数点后三位。
Required Space	索引所存储数据二进制格式下所需要的最小空间，格式同上。
Space Utilization Rate	Used Space 和 Required Space 比重，以百分号的格式表示。
Available Slot#	索引中可以插入新数据的空闲位置数量，实际是否可以复用需由 VACUUM 状态确定。
Total Points	图索引点数量。
Total Elements	包含向量数据数量。
Total Neighbors	图索引领边关系数量。
Number of elements reaching level [0,1,2,...]	多层图索引中第 N 层的点数量，如该层图为空则不打印。
Average Number of Bottom Neighbors	底层图中各点的邻边关系数量平均值。
Standard Deviation of Bottom Neighbors	底层图中各点的邻边关系数量标准差。
Percentile [1,5,10,25,50,75,90,95,99]% of Bottom Neighbors Count	底层图中各点的邻边关系数量百分位数。

fulltext索引输出内容：

attribute 属性	说明
Used Space	索引占用磁盘空间，content 格式为 xxx GB/MB/KB，精确到小数点后三位。
Required Space	索引所存储数据二进制格式下所需要的最小空间，格式同上。
Space Utilization Rate	Used Space 和 Required Space 比重，以百分号的格式表示。
Number of documents	文档数量。
Number of distinct tokens	单词数量（不包含重复单词）。
Attribute [attr name]: average doc length	属性列 [attr name] 的文档平均长度。
Attribute [attr name]: number of short-length tokens	属性列 [attr name] 的短词个数（长度小于7，中文单字长度为3）。
Attribute [attr name]: number of mid-length tokens	属性列 [attr name] 中等长度词个数（长度小于13）。
Attribute [attr name]: number of long-length tokens	属性列 [attr name] 的长词个数（长度小于49）。
Attribute [attr name]: number of full tokens	属性列 [attr name] 的超长词个数（长度大于等于49）。
Document Store Used Size	文档信息存储占用磁盘空间。
Document Store Required Size	文档信息所存储数据二进制格式下所需要的最小空间。
Token Store Used Size	单词信息存储占用磁盘空间。
Token Store Required Size	单词信息所存储数据二进制格式下所需要的最小空间。
Short Inverted List Type [0,1,2] Used Size	短类型 X 倒排索引存储占用磁盘空间。
Short Inverted List Type [0,1,2] Required Size	短类型 X 倒排索引所存储数据二进制格式下所需要的最小空间。
Short Inverted List Type [0,1,2] Number of Entries	短类型 X 倒排索引存储内容个数。
Short Inverted List Type [0,1,2] Reserved Number of Entries	短类型 X 倒排索引可以插入新数据的空闲位置数量。
Inverted List Used Size Total	倒排索引总存储占用磁盘空间。
Inverted List Required Size Total	倒排索引总存储数据在二进制格式下所需要的最小空间。 倒排索引短类型0: 单词词频小于等于4； 倒排索引短类型1: 单词词频小于等于32； 倒排索引短类型2: 单词词频小于等于162。

DiskANN索引输出内容：

attribute 属性	说明
Used Space	索引占用磁盘空间，content 格式为 xxx GB/MB/KB，精确到小数点后三位。
Required Space	索引所存储数据二进制格式下所需要的最小空间，格式同上。
Space Utilization Rate	Used Space 和 Required Space 比重，以百分号的格式表示。
Total Points	图索引点数量（等同于包含的向量数据数量）。
Reserved Slot	索引中可以插入新数据的空闲位置数量。
Total Neighbors	图索引领边关系数量。
Average Number of Neighbors	图中各点的邻边关系数量平均值。
Standard Deviation for Numer of Neighbors	图中各点的邻边关系数量标准差。
Percentile [1,5,10,25,50,75,90,95,99]% of Neighbors Count	图中各点的邻边关系数量百分位数。 索引为空时部分输出会省略。

index_inspect(oid[,partition])

描述： 作用同index_inspect(regclass[,partition])，只是将函数入参替换为索引对象的oid。

> **说明**
>
> 该函数自V3.0.0.1版本开始支持。

文本检索函数

BM25（Best Matching 25）是一种用于信息检索的经典算法，主要用于衡量一个查询与一组文档之间的相关性得分。VexDB支持基于BM25算法实现的全文检索索引fulltext，本文介绍的函数仅支持在全文检索索引查询中使用。

bm25_score()

描述： 用于计算文档与查询的相关性得分。

BM25算法基于词频（Term Frequency，TF）、逆文档频率（Inverse Document Frequency，IDF）和文档长度归一化（所有文档的平均长度）进行相关性计算。

全文检索功能依赖文本整体的统计信息计算每条文本和查询的相关度分数，常规表达式无法仅通过文本和查询内容计算出有效内容，必须通过全文检索索引fulltext查询对应算子返回分数数值。

参数： 无

返回类型： real

示例： 从comments表中搜索与查询给定的’ quick fox dog’ 相关性最高的前3个结果。

```
SELECT *, bm25_score() as score
FROM comments
WHERE document @~@ 'quick fox dog'
ORDER BY score DESC NULLS LAST LIMIT 3;
```

使用建议

以下提供了一个使用fulltext索引进行全文检索的示例场景，并介绍了使用bm25_score函数计算文本相关度分数的说明。

1. 运行以下语句产生定义与数据作为用例运行环境。

```
-- 创建表和全文索引
create table t1 (id int, title text, content text);
create index on t1 using fulltext(title, content);
create table t2 (like t1 including all);
create table t3 (like t1 including all);
-- 导入数据
insert into t1 values
    (1, '数据库文档', '全文检索文档说明'),
    (1, '数据库文档', '向量查询文档说明'),
    (2, '数据库功能', '支持全文检索功能'),
    (2, '数据库功能', '向量查询功能和全文检索功能结合'),
    (3, '其他信息', 'TBA');
insert into t2 select id + 3, title, content from t1;
insert into t3 select id + 3, title, content from t2;
```

2. 使用全文检索索引。

   - 使用分数进行排序

     全文检索查询可以使用@~@操作符对自然语言查询内容做匹配，并根据其相似度返回TOP-K数据，例如：

     ```
     select id, content from t1 where content @~@ '全文检索功能' order by bm25_score() desc nulls last limit 5;
     ```

     注意，全文检索查询仅持降序查询。即越相关的数据越先返回，在BM25算法相似计算中不存在最不相似的概念，所以对于使用升序查询的语句（明确指定或者使用默认升序），例如对于以下语句，则会报警“错误的排序顺序”并仍使用降序排序：

     ```
     select id, content from t1 where content @~@ '全文检索功能' order by bm25_score() asc limit 5;
     ```

     同理，排序中NULL数据处理将按照nulls last排序，对于nulls first排序（明确指定或者使用默认排序），例如对于以下语句，会报警“错误的排序顺序”并仍使用降序排序：

```
select id, content from t1 where content @~@ '全文检索功能' order by bm25_score() desc nulls first limit 5;
```

- 获取分数

  其中的bm25_score()函数表示查询返回的数据与@~@自然语言查询的总相似分数；可以通过SELECT获取其数值，此时可以使用别名机制进行简化，例如：

  ```
  select id, content, bm25_score() as score from t1 where content @~@ '全文检索功能' order by score desc nulls last limit 5;
  ```

- 多@~@操作符条件

  - 分数计算：对所有条件的评分求和，条件不匹配为0分。如上例title @~@ ‘宠物’ 中数据(‘ ’, ’ 狗’)的title为空完全不符合条件，则该对应分数为0，只计算其他条件分数。
  - 多条件组合使用方法：所有条件必须通过AND排列，不支持用OR联结条件。
  - 返回数据过滤逻辑：可选返回内容为所有评分为非0的数据，例如查询包括条件A AND 条件B AND 条件C，那么数据跟此三个条件任意一个相关则可以返回，如果要求三个条件均必须符合则可以通过设置MINIMUM_SHOULD_MATCH实现。
  - @@查询不参与分数计算，所有条件同样必须通过AND排列但返回数据必须符合全部条件。可以和@~@查询混合使用。

- 跨表多@~@操作符相关语句

  同理，对于多个表的多@@操作符查询，bm25_score()函数返回其相关分数总和，但需注意函数生效范围，分数计算只涉及生效范围内的所有@@操作。

  以下两个查询语句效果是等价的：

  ```
  select t1.id, t2.id, t1.content, t2.content, bm25_score() as score
  from t1, t2
  where t1.content @~@ '全文检索功能' and t2.content @~@ '向量' and t1.id + 3 = t2.id;

  --等价于
  select t1.id, t2.id, t1.content, t2.content, a + b as score
  from (select *, bm25_score() as a from t1 where t1.content @~@ '全文检索功能') t1,
      (select *, bm25_score() as b from t2 where t2.content @~@ '向量') t2
  where t1.id + 3 = t2.id;
  ```

  同样，bm25_score()函数在不同生效范围会有不同的数值，以下示例中函数返回分数会根据其生效范围决定数值。

  ```
  select t1.id, t2.id, t1.content, t2.content, bm25_score() as score, a + b as cmp_score
  from (select *, bm25_score() as a from t1 where t1.content @~@ '全文检索功能') t1,  (select *, bm25_score() as b from t2 where t2.content @~@ '向量') t2 where t1.id + 3 = t2.id;
  ```

  注意以上跨表用法会读取全部符合条件的数据，如果需要做TOP-K检索，建议将TOPK下推至每个表中。

  以下方案以损失召回的代价大幅度提升可能的性能。

```
select t1.id, t2.id, t1.content, t2.content, a + b as score
from
    (select *, bm25_score() as a from t1 where t1.content @~@ '全文检索功能' order by a desc nulls last limit 10) t1
full join
    (select *, bm25_score() as b from t2 where t2.content @~@ '向量' order by b desc nulls last limit 10) t2
on (t1.id + 3 = t2.id)
order by score desc limit 5;
```

3. 清理当前环境，删除数据。

```
DROP TABLE IF EXISTS t1, t2, t3;
```

SQL语句中bm25_score函数使用范围

- 聚集操作，例如：

```
select id, string_agg(content, ',') as all_relavant_content, sum(bm25_score()) as total_score from t1 where content @~@ '全文检索功能' group by id;
```

聚集后的别名可能由于函数性质限制在某些地方不能直接使用，但可以尝试使用原表达式替换，例如：

```
select id, string_agg(content, ',') as all_relavant_content, sum(bm25_score()) as total_score from t1 where content @~@ '全文检索功能' group by id having sum(bm25_score()) > 1;
```

注意，该场景中使用聚集操作后对相似度进行排序操作无法通过索引完成，此时全文检索会遍历所有与查询相关的数据，时延较之TOP-K查询可能大幅增加。

由于评分函数表现为不定条件的分数总和，在聚集算子返回校验中并未对分数做唯一限制，如果错误使用评分函数作为算子下集合返回值，则可能产生报错或者对应集合中的任意分数。

```
select sum(bm25_score()) as total_score, bm25_score() from t1 where content @~@ '全文检索功能';
```

- 链接操作，在跨表场景文档中已经说明了函数在SELECT和ORDER BY部分的用例，可以正常使用，其生效范围即对应语句范围。

但需注意在函数在链接部分的指代范围不明确，在链接部分中使用该函数属于未定义行为，不推荐使用。例如：

```
select t1.id, t2.id, t1.content, t2.content, bm25_score() as score
from
    (select *, bm25_score() as a from t1 where t1.content @~@ '全文检索功能' order by a desc nulls last limit 10) t1
full join
    (select *, bm25_score() as b from t2 where t2.content @~@ '向量' order by b desc nulls last limit 10) t2
on (t1.id + 3 = t2.id)
full join
    (select * from t3 where t3.content @~@ '全文向量') t3
on (t1.id + 6 = t3.id and bm25_score() > 0.3)
order by score desc limit 5;
```

其链接条件包含bm25_score() > 0.3，该函数分数所对应范围可能有(t1),(t1,t2,t3),(t3)甚至可能包括(t2,t3)，其实际生效范围由优化器推导的最优路径决定，而且路径可能会随着数据分布和统计信息变化而发生改变，在生产场景使用可能产生严重问题。

- CTE操作，bm25_score函数作用域不穿透CTE子扫描，如果需要CTE子域中的相关分数，可以使用别名机制选择。例如：

```
-- 正确用例
with cte as
(select *, bm25_score() as s from t1 where content @~@ '全文检索功能' order by s desc nulls last limit 10)
select id, content, s from cte;

-- 错误用例
with cte as
(select *, bm25_score() as s from t1 where content @~@ '全文检索功能' order by s desc nulls last limit 10)
select id, content, bm25_score() from cte;
```

同理，外界的评分函数计算总和不包括CTE子域分数，如有需要可以手动累加。

- 并集操作，对于并集内部语句，只要函数使用符合语法要求则应该正常执行；但函数评分不能向内穿透其作用域，同CTE查询需要别名机制选择返回评分：

```
-- 正确用例
select id, content, s as score
from
(select id, content, bm25_score() as s from t1 where content @~@ '全文检索功能' order by s desc nulls last limit 10)
union all
(select id, content, bm25_score() as s from t2 where content @~@ '全文检索' order by s desc nulls last limit 10)
union
(select id, content, bm25_score() as s from t3 where content @~@ '功能' order by s desc nulls last limit 10)
order by score desc limit 5;

-- 错误用例
select id, content, bm25_score() as score
from
(select id, content, bm25_score() as s from t1 where content @~@ '全文检索功能' order by s desc nulls last limit 10)
union all
(select id, content, bm25_score() as s from t2 where content @~@ '全文检索' order by s desc nulls last limit 10)
union
(select id, content, bm25_score() as s from t3 where content @~@ '功能' order by s desc nulls last limit 10)
order by score desc limit 5;
```

bm25_tokenize(text[,dictionary])

描述： 用于文本预处理和分词。

参数： 函数输入需要进行分词处理的文本类型字符串（必选），以及分词使用的分词词典名称或OID（可选），不指定词典时使用系统默认词典cn_tokenizer。

返回类型： text[]，字符串分词结果，用于模拟显示用该词典构建索引中文本数据处理结果。

示例：

```
SELECT bm25_tokenize('我司正在研发一款基于openGauss的不仅高性能又高可用而且高精度还易用的数据库');
```

返回结果如下：

```
                          bm25_tokenize
-----------------------------------------------------------------
 {我司,正在,研发,一款,opengauss,高性能,高,可用,高精度,易用,数据库}
(1 row)
```

bm25_query_tokenize(text[,dictionary])

描述： 对输入文本字符串进行扩展分词。

扩展分词 = 基础分词 + 语义/语法扩展，表示对查询字符串进行更全面和智能的分词处理，而不仅仅是简单的分词。

> **说明**
>
> 该函数自V3.0.0.1开始支持。

参数： 函数输入需要进行分词处理的文本类型字符串（必选），以及分词使用的分词词典名称或OID（可选），不指定词典时使用系统默认词典cn_tokenizer。

返回类型： text[]，函数返回包括扩展分词关键字在内的字符串分词结果，用于模拟显示用该词典进行查询时对查询文本的处理结果。

示例：

```
SELECT bm25_query_tokenize('向量数据库是AI时代的技术基石');
```

返回结果如下：

```
           bm25_query_tokenize
-----------------------------------------
 {向量,数据,据库,数据库,ai,时代,技术,基石}
(1 row)
```

数据库控制参数

除了在索引介绍章节提到的索引构建参数和 GUC （Grand Unified Configuration）查询参数以外，用户还可以通过以下 GUC 参数来控制 VexDB 向量数据库的行为。

vector_buffers

描述： 向量缓存大小，所有向量数据都会通过该缓存读取。该参数属于 POSTMASTER 类型参数。

取值范围： 65535 ~ 1073741823，单位KB

默认值： 3145728

设置建议： 尽可能超过表内向量数据大小，需要注意该参数独立于 shared_buffers（VexDB 使用的共享内存大小），即向量不再占用 shared_buffers，请合理分配内存。

vector_buffer_thread_num

描述： 向量缓存淘汰线程数，在 vector_buffers 小于所有业务正使用的索引大小总和时，需要进行缓存淘汰。该参数属于 POSTMASTER 类型参数。

取值范围： 1~8

默认值： 2

设置建议： 将该参数设置为高峰期下索引读写线程数（包括索引构建线程数）的 1/10。

max_vector_indexer_query_threads

描述： 同时进行向量检索的最大并行线程数。适用于IVFFlat 索引、IVFPQ索引和HybridANN索引的查询过程。

该参数属于POSTMASTER类型参数。

取值范围： 整型，[0,256]

默认值： 0，表示关闭向量并行检索。

向量检索索引

VexDB 通过不同近似搜索算法实现了以下向量检索索引：

- IVFFlat索引
- IVF-PQ索引
- Graph_Index索引
- DiskANN索引

Graph_Index 索引

Graph_Index索引通过构建一个多层次的图结构，利用图中节点的邻近关系，快速定位到离查询向量最近的向量节点。

Graph_Index是基于传统HNSW索引优化后得到的一种图索引，它解决了原生HNSW索引内存占用过高的问题。使用图索引的场景下，更推荐选择Graph_Index。

Graph_Index与原生HNSW

传统的HNSW索引为了获得更好的查询性能，用更多的内存来存储额外的结构和信息，以实现快速且准确的近似最近邻搜索。例如：对于1000w数据量1024维索引，最低存储空间占用大小甚至高达150GB到300GB，因此，过高的内存占用是原生HNSW索引最显著的缺点。

为了解决这一问题，Graph_Index从以下角度对原生HNSW进行了改良：

- 引入缓存机制，大幅度降低了对机器规格的内存要求。

  使用vector_buffers存储向量部分的缓存数据，使用shared_buffers存储邻边关系和元数据等信息。向量索引上的增删改查操作均直接与缓存交互，不产生任何额外的内存申请消耗。

  在同时使用多个向量索引的情况下，Graph_Index能够根据使用情况分配不同大小的缓存给各个索引，有助于进一步减少资源占用。

- 动态获取距离计算函数，根据当前机器架构选择最优计算指令集，不强制要求机器规格的指令集支持。

  并且，由于采取了向量的缓存措施，所有向量都有专门的内存对齐适配，进一步优化计算指令集使用，加速索引操作。

原理介绍

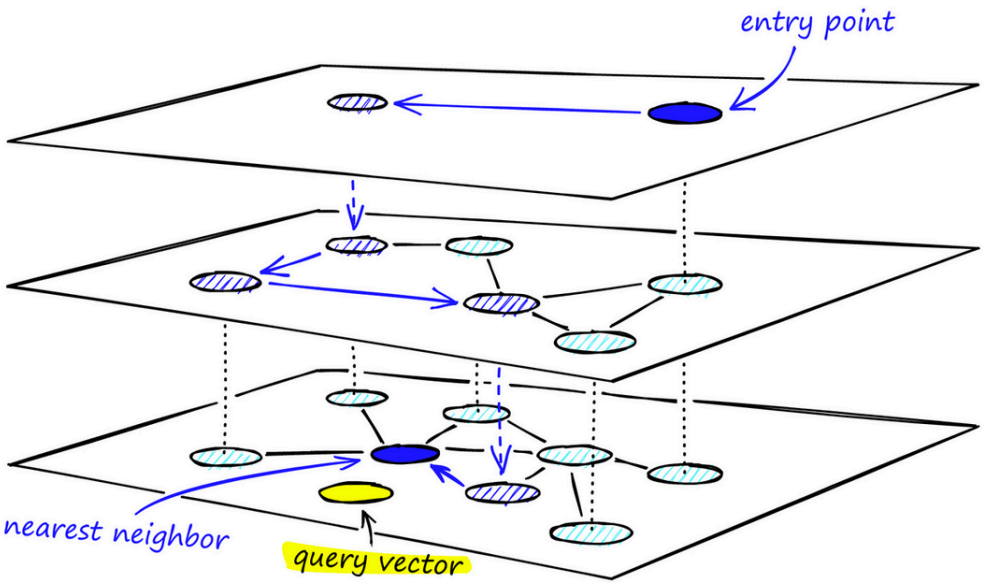
Graph_Index通过构造多层图结构来近似表示数据的分布。每个节点代表一个数据点，节点间连接的线表示数据点之间的相似性或距离。通过遍历这个图，我们可以快速找到与给定数据相似的邻居。在图结构中进行最近邻查找，可以实现比聚类算法更高的查询加速比。

搜索过程

在创建索引时构建一个初始的图，然后随着向量数据的插入不断完善优化这个多层图结构，使得相似的数据点在图中更加接近。

在搜索阶段，算法从顶层的起始点开始沿着图中的边逐步扩散，直到找到足够多的近似最近邻。

1. 顶层是查询的开始层，包含较少的节点，包含最长的链接。上层充当快速到达（接近）目标位置的导航，在深入更密集的更低层进行细致搜索之前，将搜索引导到更接近目标的位置。随着搜索向下层移动，链接长度变得越来越短，顶点数量越来越多。
2. 第一次搜索其邻居点，把它们按距离目标的远近顺序存储在定长的动态列表中。之后的每一次查找，依次取出动态列表中的点，搜索其邻居点，将这些新探索的邻居点插入动态列表，每次插入动态列表时需要重新排序，保留前 N 个。如果列表有变化则继续查找，不断迭代直至达到稳态，然后以动态列表中的第一个点作为下一层的入口点，进入下一层。
3. 重复上述过程，在每一层使用贪心算法遍历完候选列表中的顶点以及它们的邻居点后，达到最底层。我们将找不到比当前顶点更近的顶点，这是一个局部最小值，也是搜索的停止条件。



多层图的构建过程

1. 在图的构建过程中，向量逐个进行插入。向量在每一层的插入的概率由一个函数决定，对于每个层（除了第 0 层），概率函数都会重复一次。将向量添加到其插入层以及下面的每一层。
2. 图的构建从顶层开始。在进入图之后，通过贪心算法遍历图，找到与我们插入的向量最近的 ef 个最近邻顶点，此时 ef = 1。
3. 在找到局部最小值（距离插入向量最近的连接顶点）之后，它转移到下一层（与搜索过程中所做的一样）。重复这个过程，直到到达我们选择的插入层。这里开始了第二阶段的构建。
4. ef 值增加到 ef_Construction（索引构建参数），意味着将返回更多的最近邻顶点。在第二阶段，这些最近邻顶点成为与新插入的元素 q 的链接的候选顶点，也是下一层的入口点。从这些候选顶点中选择M个最近邻作为链接顶点。
5. 经过多次迭代之后，当添加链接时，还会考虑参数 m（定义一个顶点可以拥有的最大链接数）。需要设置合适的 m 值以平衡搜索的召回率和性能。
6. 插入的停止条件是在第 0 层中找到局部最小值。

支持向量量化

向量量化是一种将高精度向量映射为低精度向量的技术。其核心思想是用“代表性”的向量（称为“码字”）来代表一个区域内的所有向量，从而用码字的索引代替原始向量，实现数据压缩。通过将高维向量降维并压缩成紧凑的编码，实现快速、高效的近似最近邻搜索。

为了继续改善Graph_Index索引的搜索效率，Graph_Index从V3.0.0.1开始支持了两种向量量化方法，分别是PQ量化和RaBitQ量化。与非量化索引相比，量化通过压缩向量减少了磁盘和内存占用。在构建向量时，可以通过索引构建参数quantizer=pq或者quantizer=rabitq来指定索引的量化方法。

开启向量量化后： - 会小幅增加索引建立时间； - 可能因为计算精度的损失导致召回率下降。 - 在表初始数据量小于3840条时，不支持进行向量量化。即便创建索引时指定了quantizer参数，仍会退化为原始Graph_Index索引。且即使后续插入了足够数量的数据，也不会自动计算。 - PQ量化和RaBitQ量化对比：

维度	PQ量化	RaBitQ量化
空间占用	量化后占用的存储空间约为原始向量的1/4	量化后占用的存储空间约为原始向量的1/2
搜索速度	查询速度略降低，约为原始向量的0.8~1倍	查询速度提升约为原始向量的1.3倍
搜索精度	精度略微降低，受具体数据场景影响	精度有极低幅度降低，受具体场景影响小

PQ量化

PQ（Product Quantization）即乘积量化。在IVFPQ索引的原理介绍中，已经详细解释了利用PQ算法处理一个向量时的基本流程。经过PQ量化处理的索引仅存储了码本、码字以及码字索引，在利用改索引进行向量检索时：先依据查询向量和码本预计算好距离表，然后通过数据库中存储的码字索引快速查表、累加距离，从而找到最近邻。

使用示例：创建Graph_Index索引时指定PQ量化。

```
CREATE INDEX idx using graph_index(embedding) WITH (m=5, ef_construction=10, quantizer=pq, parallel_workers=10);
```

RaBitQ量化

RaBitQ，全称为Randomly transformed bi-valued vectors for quantizing data vectors，是一种新型的二值（Binary）量化方法。RabBitQ以bit为单位进行量化压缩和距离估算，通过将向量每一维压缩为1-bit二进制串（0或1），实现高达32倍的内存压缩。与传统的PQ量化相比，RaBitQ能在极高压缩比的前提下达到高精度和高性能的效果。

RabitQ的工作原理如下：

1. RabitQ量化

   向量图索引构建时，RaBitQ使用K-means算法对采样数据聚类，产生聚类质心centroids，以离原始向量最近的质心为基准对原始向量进行归一化，使归一化后的单位向量均匀分布于单位超球面上，则超球体内部的高维超立方体的顶点自然成为RaBitQ的Codebook码本，再将归一化的单位向量映射到超立方体的最近顶点，对于这样的顶点向量，即可用1-bit来表示每一个维度数据，也就得到了RaBitQ量化数据，此外可以通过扩展bits位数额外量化更多信息，以提供更高的召回率。在此期间，通过随机正交矩阵对向量、质心和码本进行旋转，注入随机性，消除一些特定分布上的偏移，使RaBitQ计算结果更稳定。此外，会通过一些预计算提前存储一些中间数据，为后续简化向量检索时的距离估算做准备。

   > 说明
   >
   > 为了尽可能提高查询效率，RaBitQ未使用全精度向量进行距离计算，因此不保证返回查询结果集的严格有序，但实际场景下出现向量返回错误顺序的概率极低。

2. 距离估算

   RaBitQ算法构造了一套经过理论验证的无偏估计方法，用于估算查询向量和量化索引向量的距离，在达到最小化误差的基础上，保障了高召回率。

①预计算

在向量图索引进行构建或数据插入阶段，利用RaBitQ算法对原始向量进行压缩，得到RaBitQ量化数据，并预计算出用于后续向量距离估算所需要的大部分数据和误差因子，将这些数据持久化（不包含原始向量）。

②向量检索

在向量图索引检索阶段，预先将量化的和预计算的数据缓存，并充分利用SIMD指令集进行计算，尽可能高效地估算出距离，达到提高检索效率的目的。

使用示例：创建Graph_Index索引时指定RaBitQ量化。

```
CREATE INDEX idx USING graph_index(embedding) WITH (parallel_workers=16,ef_construction=200,quantizer=rabitq);
```

索引参数

索引构建参数

参数名称	取值说明	参数描述
m	取值范围：2~100 默认值：16	每个顶点/节点与其最近邻居的最大连接数。
ef_construction	取值范围：4~1000 默认值：64	控制索引构建过程中使用的候选列表的大小。
parallel_workers	取值范围：0~64 默认值：0	并行构建参数，构建索引并行计算线程数。 从V3.0.0.1开始支持并行VACUUM。自动或者手动触发索引的VACUUM行为时，会额外启用当前参数值个数的的后端线程参与VACUUM过程。(单个索引并行构建/VACUUM的最大并行线程数为64，这个值不受参数max_background_workers影响。)
quantizer	取值范围： pq/rabitq	指定索引量化的方法。

GUC查询参数

参数名称	取值说明	参数描述
hnsw_ef_search	取值范围：1~32767 默认值：100	会话级参数，表示索引搜索过程中将考虑的最大候选邻居数。数值越高召回越精确，检索越慢。 实际生效值是top-k（查询的limit n）和hnsw_ef_search中更大的值。

索引构建操作符

索引操作符	操作符描述
floatvector_cosine_ops	计算向量的余弦距离。
floatvector_l2_ops	计算向量的欧几里得距离。
floatvector_ip_ops	计算向量的内积。

> 上述参数在索引构建阶段和向量查询阶段的具体使用方式，可参考使用示例。

使用建议

- 设置合适的 maintenance_work_mem 和 max_process_memory 值有利于索引构建速度的增快，当 maintenance_work_mem 不合适时会出现如下提示： `NOTICE: hnsw graph no longer fits into maintenance_work_mem after XXXXX tuples DETAIL: Building will take significantly more time. HINT: Increase maintenance_work_mem to speed up builds`
- 注意不要将 maintenance_work_mem 设置得太高，以免耗尽服务器的内存。
- max_process_memory 参数必须大于 maintenance_work_mem。
- 建立索引后如果数据变化（删除、更新）较多， 死元组数大于候选队列数，可能导致查询结果变少，建议每更新 40%~60% 数据重建一次索引。
- 当需要提升索引构建速度， 可以适当增大 parallel_workers，建议设置为机器CPU核数的 75% ， 最大值 64。
- 当需要提升查询召回率时， 可考虑以下措施：
  - 适当增大 m，m 越大越有利于提高召回率，构建索引越慢，查询越慢，插入索引速度越慢。
  - 增大 ef_construction，ef_construction 越大越有利于提高召回率，构建索引越慢，插入索引速度越慢。
  - 增大 hnsw_ef_search，hnsw_ef_search越大，召回率越高但相应的查询延时会变高。
- 如果向量检索时，查询未按照预期走索引，请参考向量查询不走索引中介绍的内容进行排查，确认查询是否满足走索引的条件。

使用示例

1. 数据准备：创建生成随机向量的函数。

```
CREATE OR REPLACE FUNCTION random_array(dim integer,min_value int, max_value int)
RETURNS text
AS $$
SELECT REGEXP_REPLACE(REGEXP_REPLACE(array_agg(round(random()* (max_value - min_value + 1) + min_value,3))::text,'{','['),'}',']')
FROM generate_series(1, dim);
$$
LANGUAGE SQL
VOLATILE
COST 1;
```

2. 创建含向量类型字段的测试表，并插入测试数据。

```
drop table if exists t_1194690;
CREATE TABLE t_1194690(id BIGINT, v floatVECTOR(15));
INSERT INTO t_1194690 SELECT i, random_array(15,1,3)::floatvector(15) FROM generate_series(1, 10000) AS i;
```

3. 创建graph_index索引。

   - 按余弦距离构建索引：

     ```
     CREATE INDEX idx_1194690a ON t_1194690 USING graph_index(v floatvector_cosine_ops)
     WITH(m=16,ef_construction=64, parallel_workers=1);
     ```

   - 按欧几里得距离构建索引：

     ```
     CREATE INDEX idx_1194690b ON t_1194690 USING graph_index(v floatvector_l2_ops)
     WITH(m=16,ef_construction=64, parallel_workers=1);
     ```

   - 按内积构建索引：

     ```
     CREATE INDEX idx_1194690c ON t_1194690 USING graph_index(v floatvector_ip_ops)
     WITH(m=16,ef_construction=64, parallel_workers=1);
     ```

     索引构建参数也可省略，此时参数设置为默认值：

     ```
     CREATE INDEX idx_1194690d ON t_1194690 USING graph_index(v floatvector_ip_ops);
     ```

4. 进行向量相似性搜索。

   设置查询参数：

```
set hnsw_ef_search = 100;
```

- 按余弦距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <=> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按欧几里得距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <-> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按内积排序：

```
SELECT * FROM t_1194690
ORDER BY v <#> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

IVFFlat 索引

VexDB 支持在向量数据上创建 IVFFlat 索引，采用 IVF 算法实现快速查询向量数据的目的。

> **说明**
>
> 该索引已弃用，仅为向前兼容保留。推荐使用 Graph_Index 索引。

原理介绍

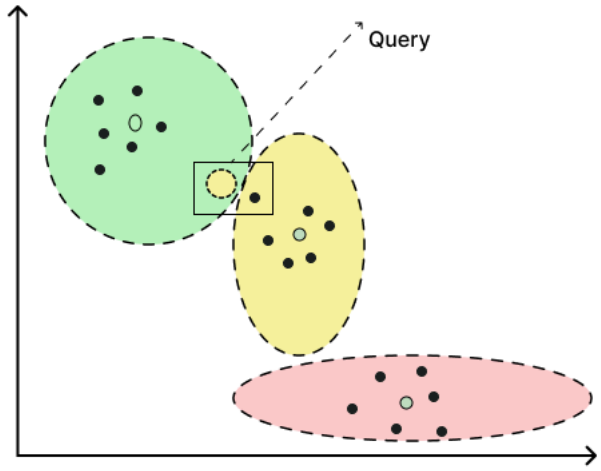
IVF算法

IVF（Inverted File System，倒排系统）算法的主要原理在于将所有向量划分成多个簇，查询时只搜索最相邻的簇，减少计算量，从而加速近似最近邻查询。

首先要对数据进行聚类。聚类是指按照某个特定标准（比如距离）把相似的数据划分到一起，使得同一个簇内的数据对象的相似性尽可能大，同时不在同一个簇中的数据对象的差异性也尽可能地大。在这个步骤中，按照 k-means 聚类算法将数据集分成多个簇，每个簇都代表一个具有类似特征或语义接近的向量集群，这种划分为检索过程的有效开展提供了便利。每个簇都有一个中心点，这个点的坐标正是倒排索引中的键。

IVFFlat 中索引的使用将搜索限制在与最近中心点相关的区域，减少了需要检查的向量数量，从而加速了搜索过程。

然而，明确了一个最近中心点表示的搜索范围后，可能存在其他簇中的向量比此范围中的向量更接近检索条件的情况（如下图所示），这也是搜索存在误差的主要原因。可以通过后文介绍的查询参数决定搜索过程要考虑的中心点的数量（簇的个数），通过适当扩大搜索范围来减小误差。



*上图中，距离查询向量最近的向量并非处于最近中心点所代表的绿色区域中，而是与黄色区域中的一个向量距离最近。

索引参数

索引构建参数

参数名称	取值说明	参数描述
ivf_nlist	取值范围：1~65535 默认值：100	索引中倒排列表的数量，即划分的“簇”的个数。
enable_toast	取值范围：true/false 默认值：true	向量索引 tuple 是否启用 toast 压缩。
parallel_workers	取值范围：0~64 默认值：0	并行构建参数，构建索引并行计算线程数。

GUC 构建参数

参数名称	取值说明	参数描述
ivf_extend_file_block_batch_count	取值范围：1~1024 默认值：64	该实例级参数指定了索引文件写满后，扩展时批量分配的 8K 数据页的个数。通过批量分配 ivf_extend_file_block_batch_count 个页，可以在读写时更高效地利用操作系统缓存。

GUC 查询参数

参数名称	取值说明	参数描述
ivf_probes	取值范围：1~65535 默认值：1	会话级参数，指定查询范围包含的倒排列表数，即查询的“簇”的个数。ivf_probes 的上限为ivf_nlist，如果设置为超过 ivf_nlist 的值， 系统会自动设置 ivf_probes = ivf_nlist。
max_vector_indexer_query_threads	取值范围：整型，[0,256] 默认值：0，表示关闭向量并行检索。	同时进行向量检索的最大并行线程数。

索引构建操作符

索引操作符	操作符描述
floatvector_cosine_ops	计算向量的余弦距离。
floatvector_l2_ops	计算向量的欧几里得距离。
floatvector_ip_ops	计算向量的内积。

> **说明**
>
> 上述参数在索引构建阶段和向量查询阶段的具体使用方式，可参考使用示例。

使用建议

- 建立 IVFFlat 索引时，表中数据不宜过少，建议不少于 10^4 条数据。
- 建议将所有需要查询的数据提前插入表中，最后再建立索引；如果决定先建立索引再导入数据，在数据导入完成后需要重建索引。
- 建立索引后如果数据变化（含插入、删除、更新）超过 20% 会产生数据分布漂移，导致查询精度下降与查询时间不稳定，建议每更新 40%~60% 数据重建一次索引。
- 当需要提升索引构建速度，可以适当增大 parallel_workers , 建议设置为机器 CPU 核数的 75%，最大值 64。
- 通常建议 ivf_nlist 的值至少为 10。较高的 ivf_nlist 值可通过缩小查询的搜索空间来加快查询速度。然而，缩小查询搜索空间可能会导致召回率降低。
- 对于少于一百万行的数据集，建议设置 ivf_lists = rows / 1000；对于超过一百万行的数据集，建议设置 ivf_lists= sqrt(rows)。
- 当需要提升召回率时，可适当增大 ivf_probes，ivf_probes 越大，召回率越高，查询越慢，当 ivf_probes = ivf_nlist 时，索引查询等同于全表扫描。
- 如果向量检索时，查询未按照预期走索引，请参考向量查询不走索引中介绍的内容进行排查，确认查询是否满足走索引的条件。

使用示例

1. 数据准备：创建生成随机向量的函数。

```
CREATE OR REPLACE FUNCTION random_array(dim integer,min_value int, max_value int)
RETURNS text
AS $$
SELECT REGEXP_REPLACE(REGEXP_REPLACE(array_agg(round(random()* (max_value - min_value + 1) + min_value,3))::text,'{','['),'}',']')
FROM generate_series(1, dim);
$$
LANGUAGE SQL
VOLATILE
COST 1;
```

2. 创建含向量字段的测试表，并调用上一步创建的函数插入向量数据。

```
drop table t_1194690;
CREATE TABLE t_1194690(id BIGINT, v floatVECTOR(15));
INSERT INTO t_1194690 SELECT i, random_array(15,1,3)::floatvector(15) FROM generate_series(1, 10000) AS i;
```

3. 创建IVFFlat 索引。

   - 按余弦距离构建索引：

     ```
     CREATE INDEX idx_1194690a ON t_1194690 USING ivfflat(v floatvector_cosine_ops)WITH(ivf_nlist= 100,enable_toast=true, parallel_workers=1);
     ```

   - 按欧几里得距离构建索引：

     ```
     CREATE INDEX idx_1194690b ON t_1194690 USING ivfflat(v floatvector_l2_ops)WITH(ivf_nlist= 100,enable_toast=true, parallel_workers=1);
     ```

   - 按内积构建索引：

     ```
     CREATE INDEX idx_1194690c ON
     ```

4. 进行向量相似性搜索。

   设置查询参数：

   ```
   set ivf_probes = 10;
   ```

   - 按余弦距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <=> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按欧几里得距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <-> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按内积排序：

```
SELECT * FROM t_1194690
ORDER BY v <#> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

IVFPQ 索引

IVFPQ（Product Quantization, 乘积量化）索引是对 IVFFlat 索引的变形， 在 IVFFlat 的基础上增加了乘积量化算法，牺牲了一定的查询精度来换取构建和查询性能的优化。

> **说明**
>
> 该索引已弃用，仅为向前兼容保留。推荐使用 Graph_Index 索引。

原理介绍

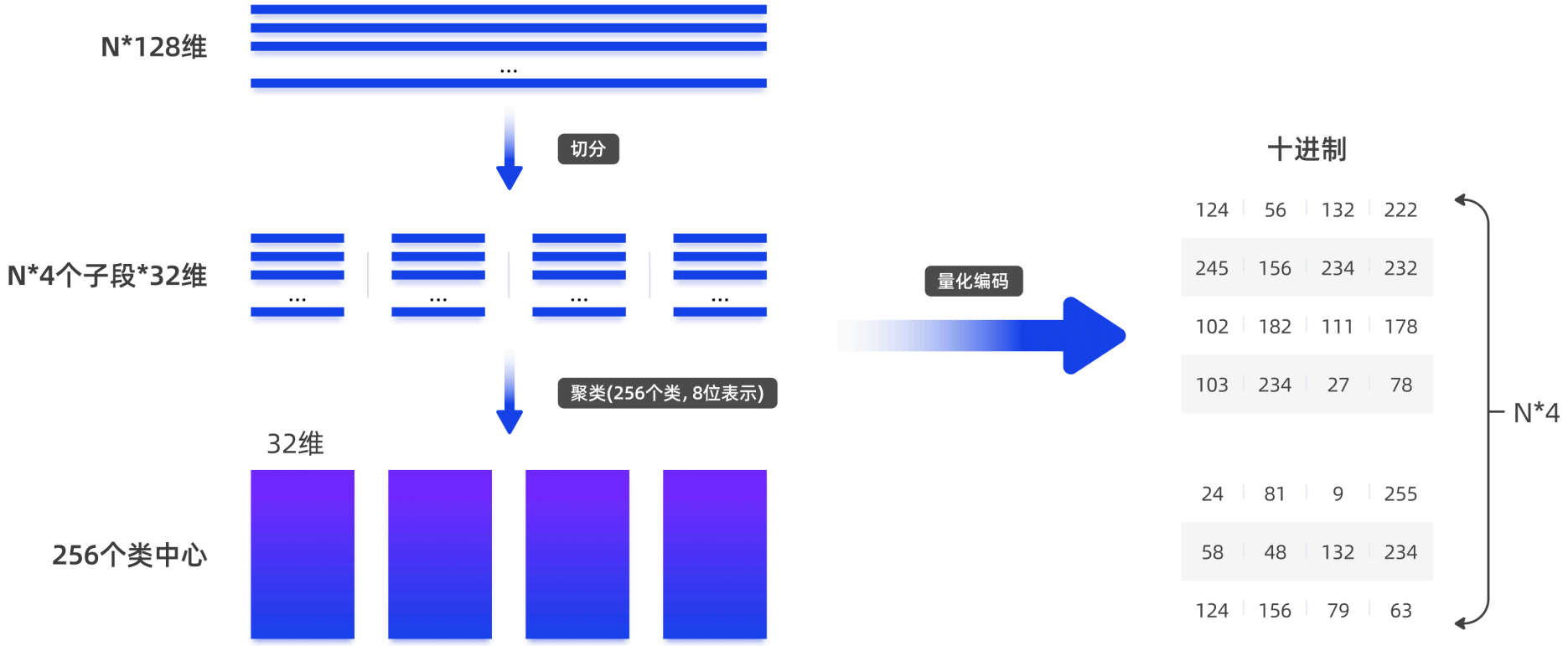
IVFPQ 算法

IVF算法的目的是减少需要计算距离的目标向量的个数，只计算部分向量与查询向量间的距离。为了继续提高查询效率，采用乘积量化（PQ）的算法，优化计算距离的过程。**即：先通过IVF算法将全部向量分成若干个簇，在每个簇中，使用PQ算法进行降维。**

当采用PQ算法处理一个向量时，高维向量经过降维被分成若干个子向量，这些子向量经过K-means聚类处理后被分配给距离它们最近的（子）簇，我们用这个（子）簇心的ID来代表这个子向量。占用大量内存的高维向量被转化为ID向量后，显著降低了向量数据占用的内存空间，这也是乘积量化的意义。

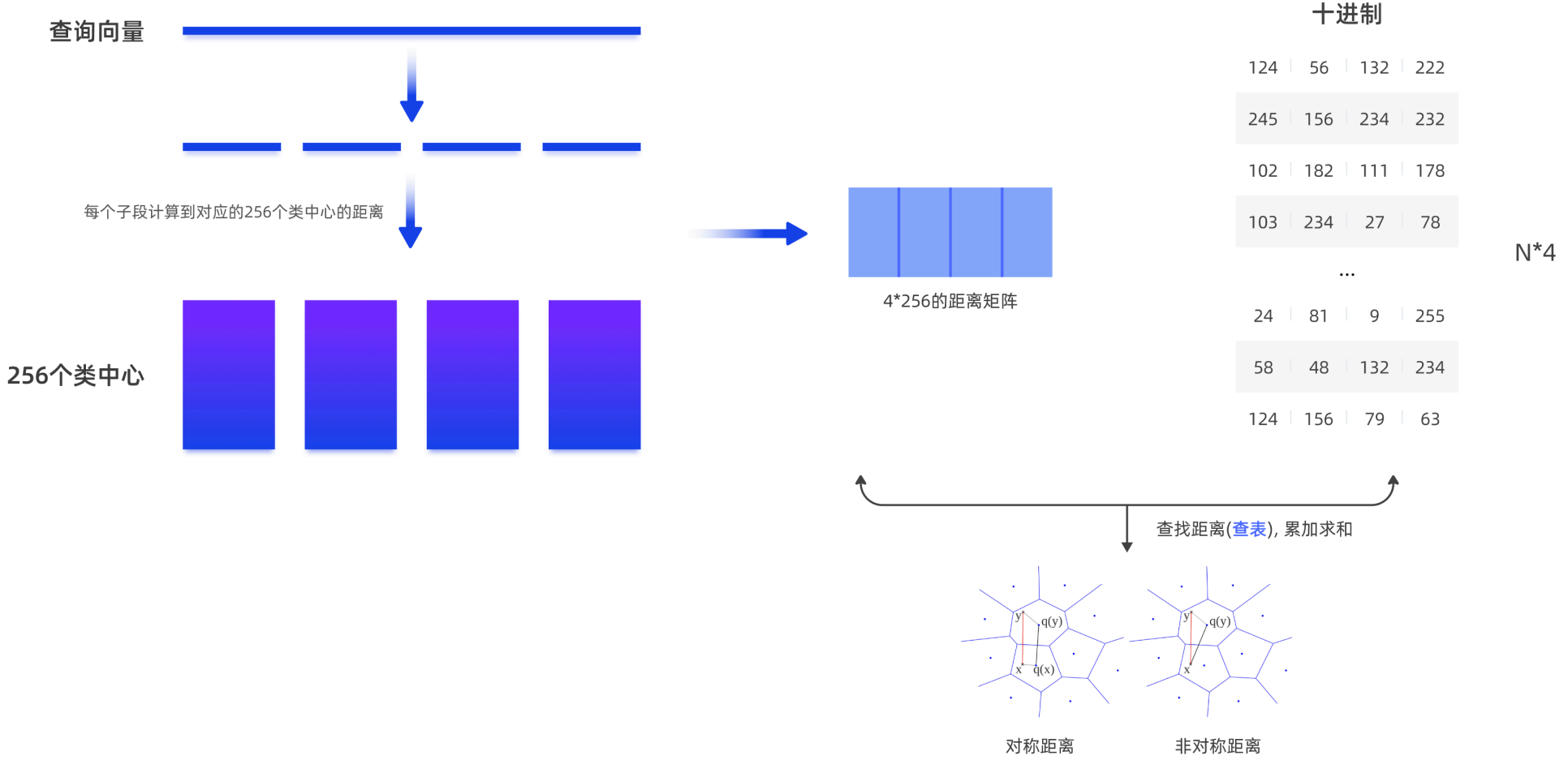
由这些子向量组成的ID集合被称为“码本”。假设一个原始向量被分为K个类，每个类中包含M段子向量，那么这个码本所能表示的样本空间容量为 K^M，即M个码本的笛卡尔积，这也是乘积量化中“乘积”的由来。

以下图为例，在索引构建过程中，先将一个128维的向量分成4段，每段32维，每段通过聚类量化为8位（256个聚类中心，2^8），即一个128维的向量可以量化成4维（每维8位）表示。经过K-means聚类处理后，得到了256个类，每个类都有一个码本。通过这样的方式，最终得以用较短的编码来表示样本，从而达到量化的目的。



那么乘积量化是如何优化距离计算过程的呢？由于PQ算法假定了处于同一个簇中的向量段离查询向量的距离相同。因此，对于一个被分为m段子向量的查询向量，计算此查询向量与每个向量间的距离时，只需要查m次表。

例如在检索时，以同样的方法把128维的查询向量分成4段的3 维向量，计算每一段向量与当前段聚类中心的距离得到一个4*256的表，将查询向量量化为[m1,m2,m3,m4]，然后从前面计算的表中查询计算查询向量与库里面向量的距离 `d=d1+d2+d3+d4` （d1为查询向量第一段子向量与其ID为m1 的簇心的距离，d2、d3和d4同理）。



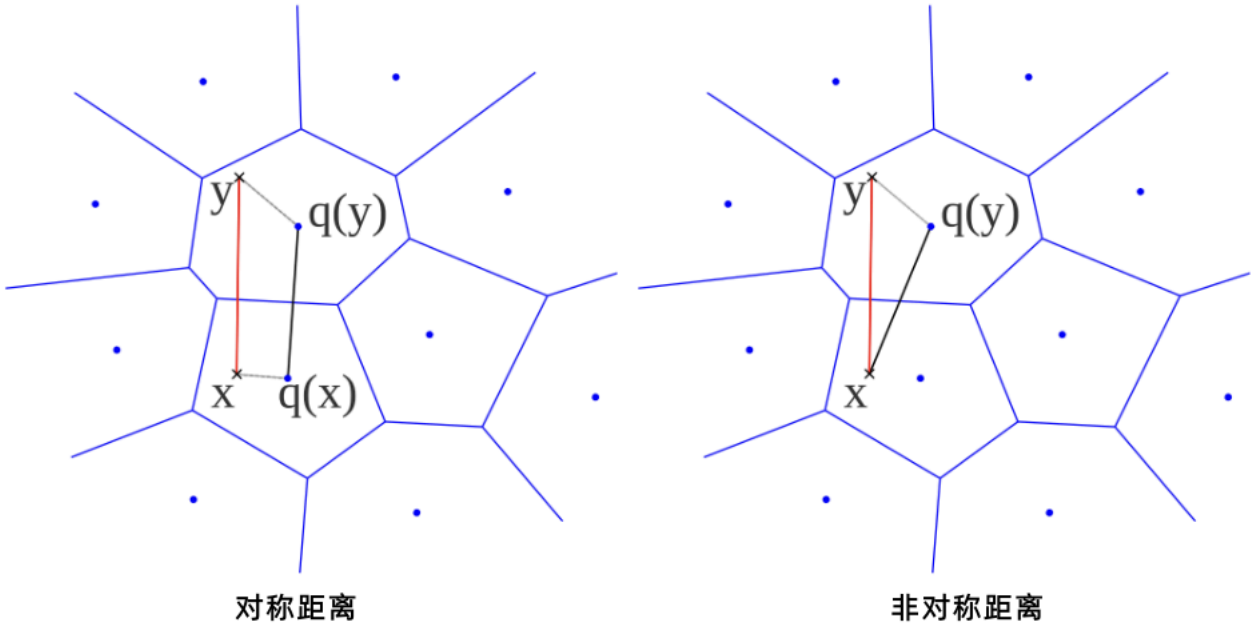
PQ乘积量化方法在计算距离的时候，有两种距离计算方式：对称距离和非对称距离。对称距离速度更快，而非对称距离的损失更小，也就是更接近真实的距离。

- 对称距离：查询向量所属的聚类中心与所有样本所属聚类中心的距离；
- 非对称距离：查询向量与所有样本所属聚类中心的距离。

如下图所示，假设有查询向量x和数据库中的向量y，那么：

对称距离：向量x和向量y都由其聚类中心 `q(x)` 、 `q(y)` 表示，距离 `d(x,y)` 可通过距离 `d(q(x),q(y))` 来估算；

非对称距离：查询向量x不经过量化，距离 `d(x,y)` 可通过距离 `d(x,q(y))` 来估算。



对称距离　　非对称距离

由于计算过程中采用的是近似距离，因而造成了一定的误差。由此可知PQ算法加快了速度但是会损失一定的精度。

索引参数

索引构建参数

参数名称	取值说明	参数描述
ivf_nlist	取值范围：1~65535 默认值：100	索引中倒排列表的数量，即划分的“簇”的个数。
num_subquantizers	取值范围：1~65535 默认值：8	指定将一个原始向量维度(d)量化成多少个段，num_subquantizers 一般小于原始向量的维数（d），且 d 必须是 num_subquantizers 的整数倍。
nbits	取值范围：1~16 默认值：8	指定原始向量分段后每段子向量编码的 bit 数。
by_residual	取值范围：true/false 默认值：true	是否用残差进行量化。
parallel_workers	取值范围：0~64 默认值：0	并行构建参数，构建索引并行计算线程数。

GUC 构建参数

参数名称	取值说明	参数描述
ivf_extend_file_block_batch_count	取值范围：1~1024 默认值：64	该实例级参数指定了索引文件写满后，扩展时批量分配的 8K 数据页的个数。通过批量分配 ivf_extend_file_block_batch_count 个页，可以在读写时更高效地利用操作系统缓存。

GUC查询参数

参数名称	取值说明	参数描述
ivf_probes	取值范围：1~65535 默认值：1	会话级参数，指定查询范围包含的倒排列表数，即查询的“簇”的个数。ivf_probes 的上限为 ivf_nlist，如果设置为超过 ivf_nlists 的值， 系统会自动设置ivf_probes = ivf_nlist。
ivfpq_refine_k_factor	取值范围：1.0~10000.0 默认值：2.0	会话级参数，控制初步筛选的候选向量的多寡。
max_vector_indexer_query_threads	取值范围：整型，[0,256] 默认值：0，表示关闭向量并行检索。	同时进行向量检索的最大并行线程数。

索引构建操作符

索引操作符	操作符描述
floatvector_cosine_ops	计算向量的余弦距离。
floatvector_l2_ops	计算向量的欧几里得距离。
floatvector_ip_ops	计算向量的内积。

> **说明**
>
> 上述参数在索引构建阶段和向量查询阶段的具体使用方式，可参考使用示例。

使用建议

- 建立 IVFPQ 索引时， 表中数据不宜过少，建议不少于 10^4 条数据。
- 建议将所有需要查询的数据提前插入表中，最后再建立索引；如果决定先建立索引再导入数据，在数据导入完成后需要重建索引。
- 建立索引后如果数据变化（含插入、删除、更新）超过 20% 会产生数据分布漂移，导致查询精度下降与查询时间不稳定，建议每更新 40%~60% 数据重建一次索引。
- 当需要提升索引构建速度， 可以适当增大 parallel_workers， 建议设置为机器 CPU 核数的 75%，最大值 64。
- 当需要提升查询召回率时， 可考虑以下措施：
- 适当增大 ivf_probes，ivf_probes 越大，召回率越高，查询越慢，当 ivf_probes = ivf_nlist 时，索引查询等同于全表扫描。
- 增大 ivfpq_refine_k_factor，一定范围内 ivfpq_refine_k_factor 越大越有利于提高召回率，查询越慢。
- 增大 nbits，nbits 越大，召回率越高但相应的查询延时会变高。
- 增大 num_subquantizers，num_subquantizers 越大，召回率越高但相应的查询延时会变高。
- nbits 推荐值一般为 8 的倍数。
- 通常建议 ivf_lists 的值至少为 10。较高的 ivf_nlist 值可通过缩小查询的搜索空间来加快查询速度。然而，缩小查询搜索空间可能会导致召回率降低。
- 对于少于一百万行的数据集，建议设置 ivf_nlist = rows / 1000；对于超过一百万行的数据集，建议设置 ivf_nlist = sqrt(rows)。
- 如果向量检索时，查询未按照预期走索引，请参考向量查询不走索引中介绍的内容进行排查，确认查询是否满足走索引的条件。

使用示例

1. 数据准备：创建生成随机向量的函数。

```
CREATE OR REPLACE FUNCTION random_array(dim integer,min_value int, max_value int)
RETURNS text
AS $$
SELECT REGEXP_REPLACE(REGEXP_REPLACE(array_agg(round(random()* (max_value - min_value + 1) + min_value,3))::text,'{','['),'}',']')
FROM generate_series(1, dim);
$$
LANGUAGE SQL
VOLATILE
COST 1;
```

2. 创建含向量类型字段的测试表，并调用上一步创建的函数插入向量数据。

```
drop table t_1194690;
CREATE TABLE t_1194690(id BIGINT, v floatVECTOR(15));
INSERT INTO t_1194690 SELECT i, random_array(15,1,3)::floatvector(15) FROM generate_series(1, 10000) AS i;
```

3. 创建IVFPQ索引。

   - 按余弦距离构建索引：

     ```
     CREATE INDEX idx_1194690a ON t_1194690 USING ivfpq(v floatvector_cosine_ops)WITH(ivf_nlist= 100, num_subquantizers=3,nbits=4,parallel_workers=1);
     ```

   - 按欧几里得距离构建索引：

     ```
     CREATE INDEX idx_1194690b ON t_1194690 USING ivfpq(v floatvector_l2_ops)WITH(ivf_nlist= 100, num_subquantizers=3,nbits=4,parallel_workers=1);
     ```

   - 按内积构建索引：

     ```
     CREATE INDEX idx_1194690c ON t_1194690 USING ivfpq(v floatvector_ip_ops)WITH(ivf_nlist= 100, num_subquantizers=3,nbits=4,parallel_workers=1);
     ```

     索引构建参数也可省略，此时参数设置为默认值：

     ```
     CREATE INDEX idx_1194690d ON t_1194690 USING ivfpq(v floatvector_ip_ops)WITH(num_subquantizers=3);
     ```

4. 进行向量相似性搜索：

   设置查询参数：

   ```
   set ivf_probes = 10;
   ```

   - 按余弦距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <=> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按欧几里得距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <-> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按内积排序：

```
SELECT * FROM t_1194690
ORDER BY v <#> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

DiskANN索引

> **说明**
>
> 该索引已弃用，仅为向前兼容保留。推荐使用 Graph_Index 索引。

原理介绍

随着数据规模的快速增长，向量检索的计算和存储成本成为了不可忽视的问题。DiskANN 索引是一种结合内存与磁盘存储的高效向量检索结构，通过图算法和量化技术优化大规模高维数据的近似最近邻搜索（ANNS，Approximate Nearest Neighbor Search），在查询速度、准确性与存储成本之间实现了兼顾。

DiskANN 支持将大规模的图索引以及数据集的全量向量存储在磁盘上，同时，为了节省存储空间，采用乘积量化（PQ）将每个数据点压缩，从而大幅降低存储空间的消耗。压缩向量存储在内存中，原始向量和图存储在 SSD 上。

搜索时，需要访问某个点的邻居，就从 SSD 中读取并载入主存，查询向量同样被压缩，通过计算压缩向量之间的距离得到近似距离。最终仍使用原始向量计算其精确距离，以此决定是否将该向量放入结果队列。

索引的构建

RobustPrune 是一种用于优化图结构的算法，它计算每条边的重要性，根据边的权重或重要性决定保留还是移除边。通过修剪图中不必要的边来降低图的密度，减少搜索时需要遍历的节点数量，同时保留足够的信息以保证查询的准确性。 其中 occlusion_factor 是一个大于 1的修剪参数，决定了在修建过程中保留多少边。occlusion_factor 越大，裁剪的条件越宽松，更多的边会被保留下来。合理的裁剪方式确保了图的持续可导航性和在多次修改后保持稳定召回率的能力。

DiskANN索引的构建过程如下：

1. 计算全局质心，以距离全局质心最近的点作为搜索算法的起始节点。
2. 从步骤1得到的起始点开始对每个点做贪婪搜索，将搜索路径上所有的点作为候选邻居集。执行 occlusion_factor = 1 的 RobustPrune，消除大部分不必要的边，在确保图的连通性的前提下初步构建图。
3. 指定索引构建参数 occlusion_factor（推荐使用默认值 1.2）并再次执行步骤2的过程。通过上述过程，我们能够获得直径比 HNSW 更小的图索引，从而使 DiskANN 能够最大限度地减少顺序磁盘读取的次数。

通过上述过程，我们能够获得直径比 HNSW 更小的图索引，从而使 DiskANN 能够最大限度地减少顺序磁盘读取的次数。

通过索引检索

1. 遍历当前节点的邻居节点，计算它们与查询向量的距离。

   同大多数基于图的ANN算法一样，DiskANN 在搜索时采用贪婪搜索 （GreedySearch）。通过不断地在候选集中选择距离查询点最近的点并探索其邻居，直到找不到更近的点为止。

2. 结果筛选。

   在贪婪搜索的过程中，始终维护一个大小为K的动态结果队列，用于存储当前找到的最近邻节点。每次找到更近的节点时，更新这个队列。 搜索结束后，返回队列中的 Top-K 作为最近邻结果。

DiskANN索引通过图结构和贪婪搜索策略，能够在庞大数据集的向量搜索中起到显著作用。

索引参数

索引构建参数

参数名称	取值说明	参数描述
occlusion_factor	取值范围：1.00000012~1.2 默认值：1.2	DISKANN 算法索引裁边参数。参数越小图索引连接度越小，部分场景适当下调该参数可以大幅缩短索引构建耗时并提高插入速度。但过小的参数设置会造成召回精度丢失。
enable_quantization	取值范围：true/false 默认值：false	使用压缩向量进行距离计算。目前无优势场景，不推荐开启。
enable_subgraph	取值范围：true/false 默认值：false	允许内存不足时尝试将数据分片构建，设置为 true 时会将数据分成多片，每片索引构建所需内存小于当前 maintenance_work_mem 参数指定数值，使用内存分别构建各片索引后汇聚成总索引，相较于使用磁盘和缓存速度更快，但会大幅度牺牲索引的召回精度，不推荐使用。
parallel_workers	取值范围：0~64 默认值：0	并行构建参数，构建索引并行计算线程数。 单个索引并行构建最大并行线程数为64，这个值不受参数max_background_workers影响。
m	取值范围：8~200 默认值：99	DiskANN 中 Vamana 图每个顶点/节点与其最近邻居的最大连接数。
ef_construction	取值范围：16~1000 默认值：100	控制索引构建过程中使用的候选列表的大小。

GUC查询参数

参数名称	取值说明	参数描述
diskann_search_list_size	取值范围： 1~32767 默认值：100	会话级参数。表示检索最近邻时的动态扫描队列大小。提高该参数可以降低检索陷入局部最优解的概率，提高召回率；相应的，也会消耗更多计算和内存资源，使查询性能降低。默认值100，可以根据召回率要求修改。 实际生效值是top-k（查询的limit n）和diskann_search_list_size中更大的值。
diskann_num_cached_nodes	取值范围： 32~1024 默认值：128	实例级参数，压缩向量信息缓存槽个数，只有使用 enable_quantization 构建的 DISKANN 索引占用该缓存。每个索引最多占用一个缓存槽。
diskann_query_with_pq	取值范围： true/false 默认值：false	会话级参数，使用压缩向量进行距离计算。只有当参数开启且进行检索的 DISKANN 索引使用 enable_quantization 构建才会生效，不推荐开启。

索引构建操作符

索引操作符	操作符描述
floatvector_cosine_ops	计算向量的余弦距离。
floatvector_l2_ops	计算向量的欧几里得距离。
floatvector_ip_ops	计算向量的内积。

使用建议

- 当需要提升索引构建速度， 可以适当增大 parallel_workers, 建议设置为 机器 CPU核数的 75%，最大值 64。
- enable_quantization 参数，目前无优势场景，不推荐开启。
- occlusion_factor 越小图索引连接度越小，部分场景适当下调该参数可以大 幅度缩短索引构建和插入速度，但过小的参数设置会造成召回精度丢失。
- enable_subgraph 设置为 true 时相较于使用磁盘和缓存速度更快，但会大 幅度牺牲索引的召回精度，不推荐使用。
- diskann_search_list_size，提高该参数可以提高召回率；相应的，也会消耗更多计算和内存资源，使查询性能降低。
- diskann_query_with_pq 参数，只有当参数开启且进行检索的 DiskANN索引使用 enable_quantization 构建才会生效，不推荐开启。
- 如果向量检索时，查询未按照预期走索引，请参考向量查询不走索引中介绍的内容进行排查，确认查询是否满足走索引的条件。

使用示例

1. 数据准备：创建用于生成随机向量的函数。

```
CREATE OR REPLACE FUNCTION random_array(dim integer,min_value int, max_value int)
RETURNS text
AS $$
SELECT REGEXP_REPLACE(REGEXP_REPLACE(array_agg(round(random()* (max_value - min_value + 1) + min_value,3))::text,'{','['),'}',']')
FROM generate_series(1, dim);
$$
LANGUAGE SQL
VOLATILE
COST 1;
```

2. 创建包含向量字段的表并插入测试数据。

```
drop table t_1194690;
CREATE TABLE t_1194690(id BIGINT, v floatVECTOR(15));
INSERT INTO t_1194690 SELECT i, random_array(15,1,3)::floatvector(15) FROM generate_series(1, 10000) AS i;
```

3. 创建 DiskANN 索引。

   - 按余弦距离构建索引：

     ```
     CREATE INDEX idx_1194690a ON t_1194690 USING diskann(v floatvector_cosine_ops)
     WITH(parallel_workers=1,enable_quantization=on,enable_subgraph=on);
     ```

   - 按欧几里得距离构建索引:

     ```
     CREATE INDEX idx_1194690b ON t_1194690 USING diskann(v floatvector_l2_ops)
     WITH(parallel_workers=1,enable_quantization=on,enable_subgraph=on);
     ```

   - 按内积构建索引:

     ```
     CREATE INDEX idx_1194690c ON t_1194690 USING diskann(v floatvector_ip_ops)
     WITH(parallel_workers=1,enable_quantization=on,enable_subgraph=on);
     ```

     参数也可省略，此时参数设置为默认值

     ```
     CREATE INDEX idx_1194690d ON t_1194690 USING diskann(v floatvector_ip_ops);
     ```

4. 进行向量相似性搜索。

设置查询参数：

```
set ef_search=10;
```

- 按余弦距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <=> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按欧几里得距离排序：

```
SELECT * FROM t_1194690
ORDER BY v <-> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

- 按内积排序：

```
SELECT * FROM t_1194690
ORDER BY v <#> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

向标联合索引

向标联合索引提供了带有标量条件的 ANN 向量检索能力。

在纯向量索引场景中，向量索引只能返回固定数量的 top-k 数据。用户在向量查询中加入其他查询条件可能会导致向量索引所查询到的 top-k 数据被其他条件过滤掉，进而造成了即使整个表中有足够的符合条件的数据，但实际查询返回数据条目依然不满足要求的问题。

合理使用向标联合索引能够有效避免类似问题。通过将标量条件融合到向量查询过程中，从而提升了查询的效率和准确性，并避免了后过滤导致找回数据量不够查询所要求的数量场景。

HybridANN索引

原理介绍

HybridAnn索引允许在向量索引基础上额外增加对标量条件的过滤功能。在构建索引时将需要支持的标量放入索引构建目标中。

HybridAnn采用了改良后的graph_index作为向量检索索引，令HybridANN也继承了graph_index的优势——更高的召回率，更低的内存占用和更短的索引构建耗时。

注意事项

- 构建HybridANN索引时，该索引第一列必须为向量类型floatvector，作为向量检索中的向量查询目标；其余列为非向量的其他标量数据，比如数值，时间，枚举类型等。具体支持的标量类型可以通过如下语句查看：

```
SELECT DISTINCT typname
FROM pg_am, pg_amop, pg_type
WHERE
    pg_type.oid = amoplefttype AND
    pg_am.oid = amopmethod AND
    amname = 'hybridann';
```

- 单个HybridANN索引最多能包含31个标量字段。
- 当增删业务较少时，可以适当放宽向标联合索引的数量限制。具体数量由同时被使用到的索引数量，增删业务量，机器规格等综合考虑。
- 向标联合索引的创建和查询资源占用以及对增删业务的影响高于其他索引类型。
- 与B+树索引要求相同，字符匹配、数组操作、表达式等B+树不支持的条件不能使用向标联合索引。
- 暂不支持在空表上创建HybridANN索引。
- 向标联合索引查询不支持针对标量字段执行 IS NULL 或 IS NOT NULL 查询， 也不支持order by标量字段走向标联合索引查询。
- 如果向标联合检索、纯向量/标量检索未按照预期走索引，请参考向量查询不走索引中介绍的内容进行排查，确认查询是否满足走索引的条件。

索引参数

索引构建参数

参数名称	取值说明	参数描述
m	取值范围：2~100 默认值：16	每个顶点/节点与其最近邻居的最大连接数。
ef_construction	取值范围：4~1000 默认值：64	控制索引构建过程中使用的候选列表的大小。
parallel_workers	取值范围：0~64 默认值：0	并行构建参数，构建索引并行计算线程数。
fillfactor	取值范围：10~100 默认值：90	构建索引时 B+树叶子节点的百分比负载度，参数逻辑同 B+树索引同名参数。不推荐更改。
vec_index_magnitudes_offset	取值范围：以冒号分隔的正整数列表。共5层。参数类型为字符串，格式为 N1[:N2[:N3[...]]。 默认值：20000:100000:500000:2500000:12500000	会话级参数，触发构建内部向量索引的数据量级，提供的量级越多，向量索引构建的越精细，查询时匹配度越高，但索引数量也越多，构建时间越长。 当增删业务较少时，可以适当放宽该参数的数量级限制。具体参数值由同时被使用到的索引数量，增删业务量，机器规格等综合考虑。
graph_magnitude_threshold	取值范围：1-10’000’000’000。 默认值20000	图索引构建阈值，大于等于该数值的向量索引使用图索引。

索引构建GUC参数

参数名称	取值说明	参数描述
max_vector_indexer_worker_threads	取值范围：2~64 默认值：16	实例级参数 max_vector_indexer_worker_threads 用于控制后台用于维护向量索引的线程数量上限。维护任务由系统自动调度分发，合理配置该参数可以有效加速大规模向量数据的索引维护过程。

GUC 查询参数

参数名称	取值说明	参数描述
hnsw_ef_search	取值范围：1~32767 默认值：100	会话级参数，表示索引搜索过程中将考虑的最大候选邻居数。数值越高召回越精确，检索越慢。 实际生效值是top-k（查询的limit n）和hnsw_ef_search中更大的值。
hybrid_query_ivf_probes_factor	取值范围：1~100 默认值：3	会话级参数。 该参数控制选择率小的查询搜索范围，调参逻辑等同于 ivf_probes ，根据召回率要求修改。
max_vector_indexer_query_threads	取值范围：整型，[0,256] 默认值：0，表示关闭向量并行检索。	同时进行向量检索的最大并行线程数。

使用建议

- 建议将所有需要查询的数据提前插入表中，最后再建立索引；如果决定先建立索引再导入数据，在数据导入完成后需要重建索引。
- 当需要提升索引构建速度， 可以适当增大 parallel_workers，建议设置为机器 CPU 核数的 75%，最大值 64。
- 当需要提升召回率时， 可适当增大 ef_search 和 hybrid_query_ivf_probes_factor，参数越大则召回率越高，查询变慢。
- 当前标量支持条件与系统默认索引 B+ 树一致。
- 使用 explain 语句确定语句是否使用向标混合索引以及预计的具体执行。

使用示例

1. 数据准备：创建用于生成随机向量的函数。

```
CREATE OR REPLACE FUNCTION random_array(dim integer,min_value int, max_value int)
RETURNS text
AS $$
SELECT REGEXP_REPLACE(REGEXP_REPLACE(array_agg(round(random()* (max_value - min_value + 1) + min_value,3))::text,'{','['),'}',']')
FROM generate_series(1, dim);
$$
LANGUAGE SQL
VOLATILE
COST 1;
```

2. 创建测试表并插入数据。

```
DROP TABLE t_1194690;
CREATE TABLE t_1194690(id BIGINT, v floatVECTOR(15));
INSERT INTO t_1194690 SELECT i, random_array(15,1,3)::floatvector(15) FROM generate_series(1, 10000) AS i;
```

3. 创建索引。注意把需要用到的标量放入索引创建内容中，将向量排在第一位。
   - 按余弦距离构建索引，使用B+树挂载向量索引实现：

```
CREATE INDEX idx_1194690a ON t_1194690 USING hybridann(v floatvector_cosine_ops, id) WITH (parallel_workers=5);
```

   - 按欧几里得距离构建索引，使用混合图索引实现（WITH设置参数也可省略，此时为默认值）：

```
CREATE INDEX idx_1194690b ON t_1194690 USING hybridann(v floatvector_l2_ops, id) WITH (parallel_workers=5);
```

4. 设置查询参数（可选）。

```
set ef_search=10;
set hybrid_query_ivf_probes_factor = 3;
```

5. 向量标量混合检索：带有标量条件的余弦距离排序。

```
SELECT * FROM t_1194690 WHERE id BETWEEN 5000 AND 6000
ORDER BY v <=> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

6. 向量标量混合检索：带有标量条件的欧几里得距离排序。

```
SELECT * FROM t_1194690 WHERE id BETWEEN 2000 AND 4000
ORDER BY v <-> '[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]'::floatvector
LIMIT 10;
```

全文检索索引

BM25（Best Matching 25）是一种广泛应用于信息检索领域的概率相关性模型，用于衡量查询与文档之间的匹配程度。通常使用场景为根据用户输入的查询内容，检索出与查询内容最相关的 top-k 文档，BM25 为查询和文档的相关性分数计算方式。其核心思想是结合以下几个因素来计算文档的相关性得分：

- 词频（Term Frequency，TF）：词语在文档中出现的次数，反映了词的重要性，但单纯依赖词频可能会使长文档得分偏高。
- 逆文档频率（Inverse Document Frequency，IDF）：反映了词语在整个文档集合中的稀有程度，常见词(如“的”、“是”)的 IDF 较低，而罕见词的 IDF 较高。
- 文档长度归一化：通过考虑文档长度及平均文档长度，平衡长文档与短文档之间的影响。

VexDB 支持基于 BM25 算法的全文检索索引 fulltext。

在进行全文检索时，您可以选择使用系统默认词典，或者通过下面介绍的方式创建自定义分词词典。

词典语法介绍

分词词典创建

```
CREATE TEXT SEARCH DICTIONARY <dict_name> (
TEMPLATE = vex_jieba
[, option = value [, ... ]]
);
```

创建分词词典时，只能指定模板为 `vex_jieba` 分词模板。该分词模板支持中英文混合分词与处理。创建语句如下：

```
CREATE TEXT SEARCH DICTIONARY test_dict (TEMPLATE = vex_jieba);
```

> **说明**
>
> - PG_TS_DICT 系统表包含定义文本检索词典的项；PG_TS_CONTENT 系统表用于记录每个自定义分词词典中包含的具体词条内容，分为停用词和用户词。
> - 内置的字典模版vex_jieba不支持tsvector或其他内核功能，仅限fulltext索引内部使用。

当前，创建分词词典的语句支持以下两个自选参数：

- stopwords：停用词词典，指定哪些词或符号在分词时会被自动过滤掉（常用于排除“的”、“是”、“and”等无实际含义的词）。

配置值	含义说明
未指定（默认）	使用系统内置的默认停用词词典。
empty	创建一个空停用词词典，用户可后续通过 SQL 添加。
	使用已有分词词典 中的停用词配置作为当前词典的停用词。

- userdict：用户自定义词典，用于指定自定义关键词，这些词在分词结果中将原样保留，不会被拆分。

配置值	含义说明
未指定（默认）	使用系统默认用户词典。
empty	创建一个空白词典，支持后续添加关键词。
	继承已有分词词典 的用户词典配置。

词典修改

- 插入停用词

  使用函数 `vexjieba_add_stopwords(dict_name, words[])` 向词典添加停用词：

  ```
  select vexjieba_add_stopwords(
      <dict_name_or_oid>,
      array[stopword1, stopword2, ...]);
  ```

  系统会自动去重，重复插入不会影响效果。

- 清空停用词

  使用函数 `vexjieba_delete_stopwords(dict_name)` 将该词典中所有停用词一次性清除：

  ```
  SELECT vexjieba_delete_stopwords('my_dict');
  ```

- 插入自定义关键词

  使用函数 `vexjieba_add_userdict(dict_name, words[])` 插入自定义关键词，支持指定词频：

  ```
  select vexjieba_add_userdict(
      <dict_name_or_oid>,
      array[keyword1, keyword2, ...]);
  ```

keyword格式	说明
关键词	如不指定词频，仅插入词本身，如 ‘苹果’
关键词+词频	‘苹果,10000’

  默认词频为 10000，高于大多数系统词条，保证可切分。

- 清空用户自定义词典

  使用函数 `vexjieba_delete_userdict(dict_name)` 清除该词典中所有自定义关键词：

  ```
  select vexjieba_delete_userdict(<dict_name_or_oid>);
  ```

词典重加载

修改词典内容后，必须执行重加载操作以使新词生效。

```
select vexjieba_reload(<dict_name_or_oid>);
```

词典删除

删除全文检索词典。

```
DROP TEXT SEARCH DICTIONARY [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

词典使用示例

1. 使用默认分词词典 cn_tokenizer。

   ```
   select bm25_tokenize('我司正在研发一款基于 openGauss 的不仅高性能又高可用而且高精度还易用的数据库');
   ```

   分词结果：{我司,正在,研发,一款,opengauss,高性能,高,可用,高精度,易用,数据库}

2. 创建一个分词词典 cn_dict，停用词词典为空，用户词典为空。

   ```
   CREATE TEXT SEARCH DICTIONARY cn_dict(
       template = pg_catalog.vex_jieba,
       stopwords = empty,
       userdict = empty);
   ```

3. 插入自定义停用词。

   ```
   select vexjieba_add_stopwords(
       'cn_dict',
       array['不仅', '又', '还', '的']);
   ```

4. 插入自定义关键词。

   ```
   SELECT vexjieba_add_userdict(
       'cn_dict',
       ARRAY['基于 openGauss', '高性能','高可用,10000','高精度']);
   ```

5. 重新加载词典 cn_dict 并使用。

   ```
   select vexjieba_reload('cn_dict');
   select bm25_tokenize('我司正在研发一款基于 openGauss 的不仅高性能又高可用而且高精度还易用的数据库', 'cn_dict');
   ```

分词结果：{我司,正,在,研发,一款,基于openGauss,高性能,高可用,而且,高精度,还易用,数据库}

6. 清空停用词。

```
SELECT vexjieba_delete_stopwords('cn_dict');
```

7. 重新加载词典cn_dict并使用。

```
select vexjieba_reload('cn_dict');
select bm25_tokenize('我司正在研发一款基于 openGauss 的不仅高性能又高可用而且高精度还易用的数据库', 'cn_dict');
```

分词结果为：{我司,正,在,研发,一款,基于openGauss,的,不仅,高性能,又,高可用,而且,高精度,还易用,的,数据库}

8. 清空用户自定义词典。

```
select vexjieba_delete_userdict('cn_dict');
```

9. 重新加载词典 cn_dict 并使用。

```
select vexjieba_reload('cn_dict');
select bm25_tokenize('我司正在研发一款基于 openGauss 的不仅高性能又高可用而且高精度还易用的数据库', 'cn_dict');
```

分词结果为：{我司,正,在,研发,一款,基于,openGauss,的,不仅,高性,能,又,高,可用,而,且,高,精度,还易用,的,数据库}

10. 删除词典。

```
DROP TEXT SEARCH DICTIONARY public.cn_dict CASCADE;
```

fulltext索引使用

创建fulltext索引

- 在普通表上创建fulltext索引

```
CREATE INDEX index_name ON table_name USING fulltext( column_name [, ...] )
WITH ( {storage_parameter = value} [, ... ];
```

- 在分区表上创建fulltext索引 storage_parameter是指定索引方法的存储参数，支持的参数请参考后文“索引构建参数”。

```
CREATE INDEX index_name ON table_name USING fulltext( column_name [, ...] ) [ LOCAL [...] | GLOBAL ]
WITH ( {storage_parameter = value} [, ... ];
```

分区表索引分为 LOCAL 索引与 GLOBAL 索引，LOCAL 索引与某个具体分区绑定，而 GLOBAL 索引则对应整个分区表。GLOBAL 索引无法执行任何指定分区的查询，例如通过分区名或者WHERE表达式条件确定查询的分区范围，只能对所有分区进行查询。

- 在V3.0.0中，分区表上创建的fulltext索引仅支持 LOCAL 索引。
- 从V3.0.0.1开始，fulltext支持 GLOBAL 全局分区索引，创建fulltext索引语句不指定 LOCAL、GLOBAL 关键字时，默认创建为 GLOBAL 索引。

索引构建参数

参数名称	取值说明	参数描述
DICTS	取值范围： cn_tokenizer 默认值：cn_tokenizer	用于指定文本类型数据解析词典，将查询内容或者文档文本转化为分隔开的单词。在多列属性场景下通过 # 对各列单独指定词典，顺序同索引列声明顺序，使用 cn_tokenizer 表示系统默认词典。 如果字段是文本数组， 就不会进行分词。
ALGORITHMS	取值范围： BM25/TF_IDF/ LOG_TF_IDF 默认值：BM25	用于指定文档-查询相似分数计算方式。 在多列属性场景下通过 `#` 对各列单独指定算法。
parallel_workers	取值范围：0~64 默认值：0	并行构建参数，构建索引并行计算线程数。 单个索引并行构建最大并行线程数为64，这个值不受参数max_background_workers影响。
COEFFICIENTS	参数 b 取值范围：[0,1] 默认值：0.75 参数 k 取值范围：[1,2] 默认值：1.2	用于指定文档-查询相似分数算法所使用的参数，设置格式“<参数>=<数值>”，不同参数通过 `:` 隔开，不允许包含空格，未指定的参数自动使用默认值，在多列属性场景下通过 `#` 对各列单独指定。目前参数包括b（默认值0.75），k（默认值1.2），只有 BM25算法会使用到这些参数。

索引查询参数

全文检索支持使用 `@~@` 和 `@-@` 操作符以及文本检索函数。

从V3.0.0.1开始，支持在 `@~@` 查询语法中，通过追加 `@<PARAM:...>@` 格式配置查询参数：

参数名称	取值说明	参数描述
boost	取值范围：大于 0 的浮点数。 默认值：1.0	BM25评分系数，当前查询内容 BM25 评分按bm25_score * BOOST 加权。
minimum_should_match	取值范围： • 正整数（固定匹配数） • 负整数（动态匹配数 = 词项总数 + 负数） • 正百分比（按百分比计算匹配数） • 负百分比（允许缺失比例）	指定当前查询字符串中需要匹配的词项数。

minimum_should_match取值说明：

格式类型	示例	说明
正整数	3	固定匹配数量，无论词项总数多少，必须精确匹配 `3` 个词项。若大于词项数量则自动缩小至词项数。
负整数	-2	动态匹配数量： `总词项数 + 负数值 = 最小匹配数` 。 例如总词项 `5` 个时， `-2` 表示需匹配 `5-2=3` 个。
正百分比	75%	按词项总数的百分比计算（向下取整）。 例如总词项 `4` 个时， `75%` 需匹配 `4×0.75=3` 个。
负百分比	-25%	按允许缺失的最大比例计算（向下取整）。 例如总词项 `4` 个时， `-25%` 允许缺失 `4×0.25=1` 个，即需匹配 `4-1=3` 个。

使用建议

- 注意关键词可以包含空格，但左右边的空格会被忽略，比如 `' a b c '` 等效于 `'a b c'` 。另外关键词搜索使用完全匹配，比如包含关键词’数据库’的文档只能通过’数据库’关键词关联，而’数’和’数据’等词不能匹配到’数据库’，如关键词 `@-@` 查询未匹配到预期结果可以使用 bm25_tokenize(text[, dictionary]) 函数校验文档分词内容。
- 多列匹配查询，只能用 AND 连接。对于需要表示 OR 逻辑的查询可以将算子嵌入到查询关键词或者使用常规 union 等优化实现。

全文检索使用示例

1. 创建测试表并插入测试数据。

```
DROP TABLE IF EXISTS comments;
CREATE TABLE comments (
id serial PRIMARY KEY,
document text,
title text
);

INSERT INTO comments (document, title) VALUES
('The quick brown fox jumps over the lazy dog', 'Animal story'),
('An agile fox was seen near the river bank', 'Wildlife report'),
('Dogs are loyal animals and good companions', 'Pet life'),
('Foxes are known for their cunning behavior', 'Wild instincts'),
('The quick blue hare outpaced the lazy dog', 'Animal race'),
('Dogs and foxes can both be found in Europe', 'Ecosystem overview');
```

2. 默认使用系统分词器 cn_tokenizer，document 使用 cn_tokenizer 和 BM25 算法，参数为 b=0.75 和 k=1.2；title 使用 cn_tokenizer 和 TF_IDF 算法，无需参数。

```
CREATE INDEX bm25_idx_comments
ON comments USING fulltext(document, title)
WITH (
DICTS='cn_tokenizer#cn_tokenizer',
ALGORITHMS='BM25#TF_IDF',
COEFFICIENTS='b=0.75:k=1.2'
);
```

3. 关键词匹配查询。

```
-- 以下查询只能通过BM25索引执行，关键词包含使用@-@操作符，BM25评分使用@~@操作符。
-- 1. 单关键词命中，查询document包含“fox”关键词的记录
SELECT * FROM comments WHERE document @-@ 'fox';
-- 2. 多关键词AND查询，查询document包含“fox”和“dog”关键词的记录
SELECT * FROM comments WHERE document @-@ 'fox AND dog';
-- 3. 多关键词OR查询，查询document包含“fox”或者“dog”关键词的记录
SELECT * FROM comments WHERE document @-@ 'fox OR dog';
-- 4. 复杂布尔逻辑查询，查询document包含“fox” 或者 “dog”和“quick”同时存在的关键词的记录
SELECT * FROM comments WHERE document @-@ 'fox OR (dog AND quick)';
-- 5. 多列匹配查询（必须用AND连接），查询document包含“fox” 和 title包含“Animal”的记录
SELECT * FROM comments WHERE document @-@ 'fox' AND title @-@ 'Animal';
```

4. BM25 Top-k 排序查询。

   - 文本格式查询，查找与搜索词最相关文档，并按 BM25 算法打分排序，返回前3。

     ```
     SELECT *, bm25_score() as score
     FROM comments
     WHERE document @~@ 'quick fox dog'
     ORDER BY score DESC NULLS LAST LIMIT 3;
     ```

   - 查询 document 包含“fox” 并设置 BOOST=2.0，提高该字段权重。

```
SELECT *, bm25_score() AS score
FROM comments
WHERE document @~@ 'fox @<PARAM:BOOST=2.0>@'
ORDER BY score DESC NULLS LAST;
```

5. 多列匹配时分别设置 BOOST。
   - document 权重 1.5，title 权重 3.0：

```
SELECT *, bm25_score() AS score
FROM comments
WHERE document @~@ 'fox @<PARAM:BOOST=1.5>@'
AND title @~@ 'Animal @<PARAM:BOOST=3.0>@'
ORDER BY score DESC NULLS LAST;
```

   - 查询“quick fox dog” 但至少需要匹配 2 个词项：

```
SELECT *, bm25_score() AS score
FROM comments
WHERE document @~@ 'quick fox dog @<PARAM:MINIMUM_SHOULD_MATCH=2>@'
ORDER BY score DESC NULLS LAST;
```

   - 至少匹配 75% 的词项（对于3个词项，至少匹配2个（3 × 75% = 2.25，向上取整）：

```
SELECT *, bm25_score() AS score
FROM comments
WHERE document @~@ 'quick fox dog @<PARAM:MINIMUM_SHOULD_MATCH=75%>@'
ORDER BY score DESC NULLS LAST;
```

   - 允许缺失 1 个词项：

```
SELECT *, bm25_score() AS score
FROM comments
WHERE document @~@ 'quick fox dog @<PARAM:MINIMUM_SHOULD_MATCH=-1>@'
ORDER BY score DESC NULLS LAST;
```

   - 严格匹配全部词项：

```
SELECT *, bm25_score() AS score
FROM comments
WHERE document @~@ 'quick fox dog @<PARAM:MINIMUM_SHOULD_MATCH=100%>@'
ORDER BY score DESC NULLS LAST;
```

   - 查询“quick fox dog”，至少匹配 2 个词项，且 BOOST=1.8：

```
SELECT *, bm25_score() AS score
FROM comments
WHERE document @~@ 'quick fox dog @<PARAM:MINIMUM_SHOULD_MATCH=2 PARAM:BOOST=1.8>@'
ORDER BY score DESC NULLS LAST;
```

6. 外部分词创建索引。

```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (
id serial PRIMARY KEY,
raw_text text,        -- 自动分词字段
tokens text[]         -- 外部分词字段
);
INSERT INTO docs (raw_text, tokens) VALUES
('The quick brown fox jumps over the lazy dog', ARRAY['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']),
('An agile fox was seen near the river bank', ARRAY['an', 'agile', 'fox', 'was', 'seen', 'near', 'the', 'river', 'bank']),
('Dogs are loyal animals and good companions', ARRAY['dogs', 'are', 'loyal', 'animals', 'and', 'good', 'companions']);

CREATE INDEX idx_bm25_tokens
ON docs USING fulltext(tokens)
WITH (
ALGORITHMS = BM25,
COEFFICIENTS = 'b=0.75:k=1.2'
);
```

使用限制

前提条件

不支持在 USTORE、CSTORE 存储引擎中使用向量检索。

向量字段和向量索引维度限制

- 不允许建立超过 16384 个维度的向量字段。
- Graph_Index、DiskANN、HybridANN 索引支持在不超过 16384 维度的字段上建立。
- 其他向量索引仅支持在不超过 2000 维度的字段上建立。

不支持的向量运算符

暂不支持的运算符	含义	影响
||	将两个向量连接成一个新的更长的向量。	尝试使用此运算符时将导致错误。
*	两个向量逐元素相乘。	尝试使用此运算符时将导致错误。
<+>	计算两个向量之间的曼哈顿距离，即各维度差的绝对值之和。	尝试使用此运算符时将导致错误。

不支持的向量比较符

暂不支持的比较符	含义	影响
=	用于判断两个向量的所有对应元素是否完全相等。	尝试使用此比较符将导致报错。
>	用于判断一个向量是否大于另一个向量。	尝试使用此比较符将导致报错。
<	用于判断一个向量是否小于另一个向量。	尝试使用此比较符将导致报错。
>=	用于判断一个向量是否大于等于另一个向量。	尝试使用此比较符将导致报错。
<=	用于判断一个向量是否小于等于另一个向量。	尝试使用此比较符将导致报错。
<>	用于判断两个向量是否不同。	尝试使用此比较符将导致报错。
!=	用于判断两个向量是否不同。	尝试使用此比较符将导致报错。
BETWEEN	判断向量是否在指定范围内。	尝试使用此比较符将导致报错。

不支持的聚合函数

除count()之外， 向量字段不支持其他聚合函数。

以下用法报错：

```
create table test1(c1 int,c2 floatvector(3));
select avg(c2) from test1;
select sum(c2) from test1;
```

向量字段不支持 union 和 col in(...) 语法

向量数据暂不支持 `=` 操作符，故暂不支持上述用法。

向量字段不能用在 GROUP BY/ORDER BY 子句中

```
create table test1(c1 int,c2 floatvector(3));
--以下 SELECT 语句报错:
select c2 from test group by c2 limit 3;
select c2 from test order by c2;
```

向量字段不能作为主键或唯一性约束

以下用法报错：

```
create table test1(c1 int unique,c2 floatvector primary key);
create table test1(c1 int,c2 floatvector unique);
```

建立索引删除/更新大量数据后应先 VACUUM 重建索引

原因

当删除或者更新数据后，改动之前的数据并不会立即从存储中移除，而是作为“死元组”存在于表中。索引仍然可能指向这些死元组，导致查询统计结果时时误将这些无效数据包含在内，虽然返回客户端之前会过滤掉，用户不会直接看到已经删除的数据，但是当候补元组数量不足时，可能造成返回数据少于预期。

HNSW 索引由参数 hnsw_ef_search 决定候补元组数，hnsw_ef_search 越大候补元组数越多；ivfflat 及 ivf-pq 索引由参数 ivf_nlist 和 ivf_probes 决定候补元组数，ivf_nlist 越小、ivf_probes 越大，候补元组数越多。

建议

当更新/删除数据占总元组数达到 60% 或更高，删除索引，再 VACUUM，重建索引。

常见问题

同一个向量类型字段上能否创建多个向量索引

是的，VexDB 支持这种用法。允许在表的同一个字段上同时创建多种索引，查询优化器会根据各索引估计代价大小选择最终使用的索引。

目前无优势场景，暂不建议这样使用。

向量查询不走索引

向量查询走向量索引的前提条件:

- 使用ORDER BY进行升序排序，且第一个排序条件必须是向量距离计算，距离越小表示越相似。查询使用的距离计算操作符必须与创建索引时指定的操作符一致，对应关系如下：
  - <-> floatvector_l2_ops
  - <=> floatvector_cosine_ops
  - <#> floatvector_ip_ops
- 使用LIMIT对查询结果做top-k限制，否则无法选择向量索引。使用Hint可以强制走索引，但只会返回部分数据。

向标联合索引：

- HybridANN 索引支持纯向量查询，使用限制条件同上。
- HybridANN索引支持纯标量查询，排除第一个向量列后，使用条件和B+树索引相同，但不支持IndexOnlyScan算子。
- HybridANN进行向标联合检索时，会将支持范围内的标量放入检索范围内，向量支持同纯向量查询，标量支持同纯标量查询。

除了上述条件外，是否选择索引以及选择哪些支持范围内的条件组合由代价估计决定。

影响 IVF-PQ 查询召回率的因素

IVF-PQ 子查询受外层 limit 影响，可能导致查询召回率下降：在包含子查询的情况下，如果子查询和外层查询的排序字段不一致，并且外层查询包含 LIMIT 子句，可能会导致查询结果的精确度降低。

```
WITH tmp AS (
SELECT v AS vec from test1
)
SELECT id
FROM items where id not in (select id from items where id>1000 order by id)
ORDER BY embedding <=> (SELECT vec FROM tmp)
LIMIT 5;
```

例如， 在上述语句中，内层子查询的排序字段为 id，外层排序字段是向量字段且携带了 limit 语句，此时可能会触发 ivf-pq 的 limit 下推逻辑而导致查询召回率下降。

索引返回元组数可能少于走全表扫描的数据或者 Limit 子句指定的数量

原因：由于 IVFFlat、IVF-PQ 和 HNSW 索引底层算法都是近似搜索算法，查询返回的结果数量会受到不同参数的影响。

对于 IVFFlat 和 IVF-PQ 索引，当 ivf_nlist 值较大而 ivf_probes 值较小时，索引会在较小的数据范围内进行搜索，这可能导致返回结果数量减少。

向量标量混合查询，走索引查询返回的结果可能少于走全表扫描的结果

原因：在混合查询（包含向量条件和标量条件）中，需要协调向量相似度检索与标量条件过滤的顺序。通常，查询会先基于向量条件进行相似度检索，得到的近似结果再经过标量条件过滤。这种处理顺序可能导致部分符合标量条件的记录因不在初步的向量相似度结果集中而被过滤掉，从而进一步减少最终的返回条数。因此，经过双重过滤后，最终返回的结果条数可能会少于预期的数量。

管理员指南

本章提供了 VexDB 常用的管理员操作，方便管理员用户更好地使用数据库。

设置字符集与排序规则

字符集 (Character Set) 是符号与编码的集合，排序规则 (Collation) 是在字符集中比较字符以及字符排序顺序的规则，字符分类（Character Type）是字符的处理方式。

字符集

基本概念

字符集（Character Set）定义了如何在数据库中表示和存储字符数据。

字符集是字符编码的集合，字符编码（Encoding）规则规定了一组字符到编码的映射，例如 UTF-8 的字符到 ASCII 码映射规则如下表所示。

表1 UTF-8 字符到 ASCII 码映射

字符	描述	ASCII 码
1	数字 1	49
2	数字 2	50
3	数字 3	51
一	汉字 1	19968
二	汉字 2	20108
三	汉字 3	19977

在 VexDB 数据库中，字符集与编码规则并无实质上的区分，为数据库对象指定字符集时，同时也指定了这些对象的编码规则。

字符集分类

字符集可分为 ASCII 字符集、非 Unicode 字符集与 Unicode 字符集。

- ASCII 字符集：使用 7 比特二进制字符串表示编码，前 3 位+后 4 位支持 128 个字符。例如上表中数字 1 的ASCII 码为 49，通过 hex(49) 换算后，即 0110001，其编码为 0x31。

  表2 ASCII 字符集编码

-	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	TAB	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

- 非 Unicode 字符集：使用一个或多个字节来表示一个字符。例如 GB18030-2022 中，会使用 1 ~ 4 个字节来表示一个字符，大多数中文字符会使用 2 个字节来表示，对于一些特殊的中文字符会使用 4 个字节。

  例如在 GBK 环境中，执行以下语句：

```
SELECT bit_length('数智引航');
```

返回结果为 64，即 16 Bytes * 4。

```
bit_length
------------
        64
(1 row)
```

若返回结果不为 64，请检查终端、客户端或服务端的字符集配置。

- Unicode 字符集：Unicode 是一个国际标准，支持多种语言的字符。Unicode 使用 1 ~ 4 字节来表示一个字符，例如 UTF-8 中，会使用 3 个字节来表示一个中文字符。

  例如在 UTF8 环境中，执行以下语句：

```
SELECT bit_length('数智引航');
```

返回结果为 96，即 24 Bytes * 4。

```
bit_length
------------
        96
(1 row)
```

若返回结果不为 96，请检查终端、客户端或服务端的字符集配置。

客户端与服务端字符集

客户端字符集是用户连接数据库的客户端使用的字符集，服务端字符集是数据库实例使用的字符集。此外，如果使用远程连接软件，还需要考虑终端字符集。

- 参数 server_encoding 表示数据库的服务端编码字符集。
- 参数 client_encoding 表示客户端的字符集，可以通过 SET NAMES 语句修改。
- 终端字符集

  远程连接软件使用的字符集，需要在连接软件中进行配置。

其中，client_encoding 由用户配置，server_encoding 仅支持在初始化数据库时配置。

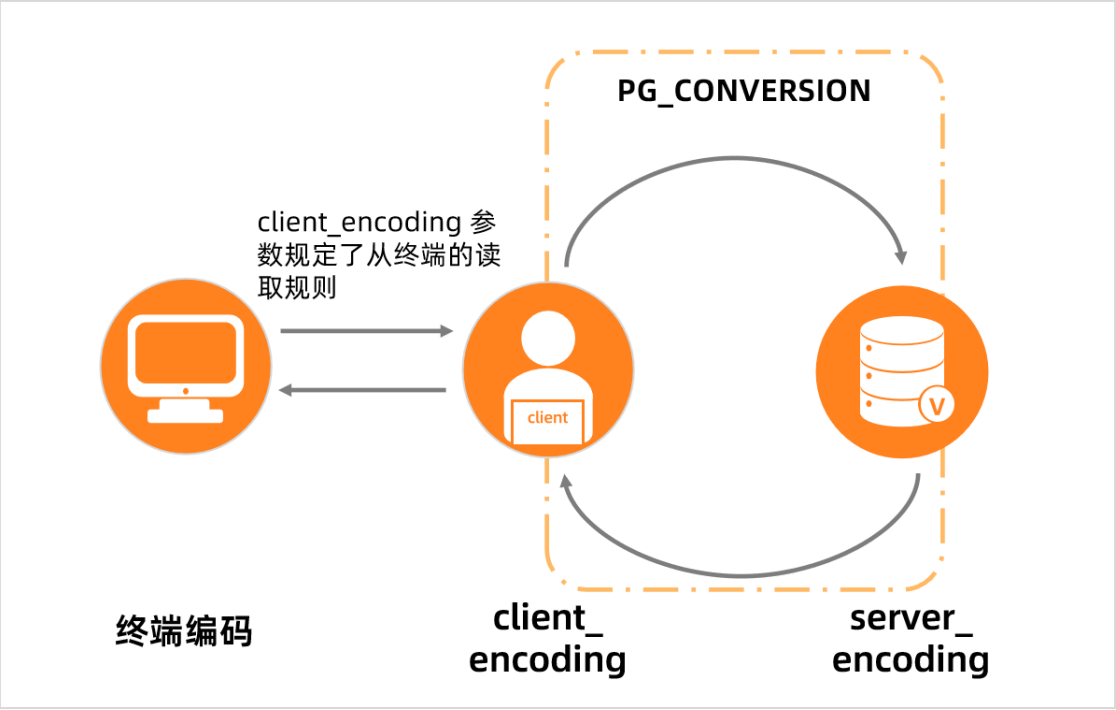
客户端与服务端编码转换

若客户端编码为 A，服务器端编码为 B，则需要满足数据库中存在编码格式 A 与 B 的转换。

系统表 PG_CONVERSION 记录了 VexDB 数据库支持的字符集转换规则。若无法转换，则建议客户端编码与服务器端编码保持一致，客户端编码可通过GUC 参数 client_encoding 修改。

- 服务端接收到客户端发送的 SQL 语句后，会将其由客户端字符集 client_encoding 向数据库字符集 server_encoding 转换编码。
- 查询结果数据发送到客户端之前也会将数据向客户端字符集 client_encoding 转换编码。

编码转换如下图所示。



> **说明**
>
> 使用终端连接数据库时，需要考虑到终端使用的字符集。例如使用 shell 工具连接到数据库服务器时，需要考虑 shell 工具输入的字符集；使用 JDBC 连接数据库时，需要考虑开发环境的字符集等。

例如，使用 JDBC 连接数据库时，VexDB 数据库的字符集为 UTF8，JDBC 连接串设置的字符集为 UTF8，而 java 文件的编码方式为 GBK。则返结果可能不符合预期参见 示例1：使用 JDBC 查询 GBK 编码。

对于以上情况，需要配置连接参数 characterEncoding=gbk 与 allowEncodingChanges=true。其中，characterEncoding 表示 JDBC 客户端的字符集，allowEncodingChanges 表示允许进行终端的字符集的转换。

字符集与字符类型

不同的 client_encoding 编码方式下，向字符类型（如 char 或 nchar）中插入数据，可能会超出长度限制。这是由于终端编码不变的情况下，client_encoding 还表示从终端读取编码的规则。

例如终端编码为 UTF8，client_encoding 为 GBK，则客户端会按 GBK 编码规则读取终端的输入，这可能导致编码读取错误。参见 示例3：不同编码字符集下 char 类型允许存储的字符数量。

> **说明**
>
> 形如 char(n) 中的 n 表示存储的字节数量。另外，VexDB 数据库支持形如 char(n character) 的定义方式，通过显式指定 n character，表示存储 n 个字符。

配置方式

VexDB 允许配置数据库级字符集，但不支持修改字符集。

VexDB 支持的字符集列表

表3 VexDB 支持的字符集列表

名称	描述	语言	服务端	字节/字符	别名
BIG5	Big Five	繁体中文	否	1-2	WIN950, Windows950
EUC_CN	扩展UNIX编码-中国	简体中文	是	1-3	
EUC_JP	扩展UNIX编码-日本	日文	是	1-3	
EUC_JIS_2004	扩展UNIX编码-日本, JIS X 0213	日文	是	1-3	
EUC_KR	扩展UNIX编码-韩国	韩文	是	1-3	
EUC_TW	扩展UNIX编码-台湾	繁体中文，中国台湾话	是	1-3	
GB18030	国家标准	中文	是	1-4	
GBK	扩展国家标准	简体中文	是	1-2	WIN936, Windows936
BIG5HKSCS	ISO 10646，HKSCS 2016 香港增补字符集	繁体中文	是	1-2	
ISO_8859_5	ISO 8859-5, ECMA 113	拉丁语/西里尔语	是	1	
ISO_8859_6	ISO 8859-6, ECMA 114	拉丁语/阿拉伯语	是	1	
ISO_8859_7	ISO 8859-7, ECMA 118	拉丁语/希腊语	是	1	
ISO_8859_8	ISO 8859-8, ECMA 121	拉丁语/希伯来语	是	1	
JOHAB	JOHAB	韩语	否	1-3	
KOI8R	KOI8-R	西里尔语（俄语）	是	1	KOI8
KOI8U	KOI8-U	西里尔语（乌克兰语）	是	1	
LATIN1	ISO 8859-1, ECMA 94	西欧	是	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	中欧	是	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	南欧	是	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	北欧	是	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	土耳其语	是	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	日耳曼语	是	1	ISO885910
LATIN7	ISO 8859-13	波罗的海	是	1	ISO885913
LATIN8	ISO 8859-14	凯尔特语	是	1	ISO885914

LATIN9	ISO 8859-15	带欧罗巴和口音的 LATIN1	是	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	罗马尼亚语	是	1	ISO885916
MULE_INTERNAL	Mule 内部编码	多语种编辑器	是	1-4	
SJIS	Shift JIS	日语	否	1-2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	日语	否	1-2	
SQL_ASCII	未指定（见文本）	任意	是	1	
UHC	统一韩语编码	韩语	否	1-2	WIN949, Windows949
UTF8	Unicode, 8-bit	所有	是	1-4	Unicode
WIN866	Windows CP866	西里尔语	是	1	ALT
WIN874	Windows CP874	泰语	是	1	
WIN1250	Windows CP1250	中欧	是	1	
WIN1251	Windows CP1251	西里尔语	是	1	WIN
WIN1252	Windows CP1252	西欧	是	1	
WIN1253	Windows CP1253	希腊语	是	1	
WIN1254	Windows CP1254	土耳其语	是	1	
WIN1255	Windows CP1255	希伯来语	是	1	
WIN1256	Windows CP1256	阿拉伯语	是	1	
WIN1257	Windows CP1257	波罗的海	是	1	
WIN1258	Windows CP1258	越南语	是	1	ABC, TCVN, TCVN5712, VSCII

需要注意并非所有的客户端 API 都支持上面列出的字符集。

SQL_ASCII 设置与其他设置不同：如果服务端字符集是 SQL_ASCII，VexDB 会把字节值 0~127 根据 ASCII 码进行翻译，而字节值 128~255 则会被当作无法解析的字符。如果字符集设置为 SQL_ASCII，就不会进行编码转换。因此，这个设置基本不是用来声明所使用的指定编码，因为该字符集会忽略编码。在大多数情况下，如果使用了任何非 ASCII 编码的数据，则不建议使用 SQL_ASCII 字符集，因为 VexDB 将无法转换或者校验非 ASCII 字符。

- 指定新的数据库字符集编码必须与所选择的本地环境中 LC_COLLATE 与 LC_CTYPE 的设置兼容。

- 当指定的字符编码集为 GBK 时，部分中文生僻字无法直接作为对象名。这是因为 GBK 第二个字节的编码范围在 0x40~0x7E 之间时，字节编码与 ASCII 字符 @A-Z []^ `a-z{|} 重叠。其中 @[]^{|} 是数据库中的操作符，直接作为对象名时，会语法报错。例如“俦”字，GBK16 进制编码为 0x8240，第二个字节为 0x40，与 ASCII “@” 符号编码相同，因此无法直接作为对象名使用。如果确实要使用，可以在创建和访问对象时，通过增加双引号来规避这个问题。

初始化字符集

在初始化数据库时，VexDB 会根据初始化参数 encoding 配置，默认会选择操作系统的字符集作为初始化字符集。由于 VexDB 安装包文件默认使用 UTF8 编码，因此建议在初始化时指定 locale=UTF8。

在创建数据库时配置字符集

用户可以在数据库创建时指定非缺省编码，但是指定的编码必须与所选的区域相兼容。VexDB 支持通过以下方式在创建数据库时指定字符集：

- 通过命令行指定：

```
createdb -E GB18030 -T template0 --lc-collate=zh_CN.gb18030 --lc-ctype=zh_CN.gb18030 testdb
```

- 通过 SQL 命令指定：

```
CREATE DATABASE testdb WITH ENCODING 'GB18030' LC_COLLATE='zh_CN.gb18030' LC_CTYPE='zh_CN.gb18030' TEMPLATE=template0;
```

  上述命令指定 TEMPLATE 为 template0 数据库，指定 TEMPLATE 为其他数据库时，编码和区域设置不能被改变，因为可能导致数据损坏。

数据库的编码存储在 PG_DATABASE 系统表中。 可以使用 vsql 的 `-l` 选项或 `\l` 命令列出这些编码。

```
vsql -l
```

返回结果如下：

```
                              List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 postgres  | vexdb    | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | vexdb    | UTF8     | en_US.utf8 | en_US.utf8 | =c/vexdb          +
           |          |          |            |            | vexdb=CTc/vexdb
 template1 | vexdb    | UTF8     | en_US.utf8 | en_US.utf8 | =c/vexdb          +
           |          |          |            |            | vexdb=CTc/vexdb
 vexdb     | vexdb    | UTF8     | en_US.utf8 | en_US.utf8 |
(5 rows)
```

示例

示例1：使用 JDBC 查询 GBK 编码

VexDB 数据库的字符集为 UTF8，JDBC 连接串设置的字符集为 UTF8，而 java 文件的编码方式为 GBK 时：

1. 执行以下 Java 代码。

```
import java.sql.*;

public class testConn{
    static Connection conn = null;
    static String cname = "com.vexdb.Driver";
    static String url = "jdbc:vexdb://xxx.xxx.xxx.xx:xxxx/postgres?loggerLevel=OFF&characterEncoding=gbk&allowEncodingChanges=true";
    static String username = "vbadmin";
    static String passwd = "VBase@123";
// coded in GBK
    public static void main(String[] args){
        try{
            Class.forName(cname);
            conn = DriverManager.getConnection(url,username,passwd);
                    System.out.println("[SUCCESS] conn database success.\n");
            PreparedStatement pstmt = conn.prepareStatement("SELECT bit_length('数智引航')");
            pstmt.setFetchSize(1);
            ResultSet rs = pstmt.executeQuery();
            while (rs.next())
            {
                System.out.println("bit_length\n----------\n" + rs.getInt(1) + "\n");
            }
        }catch (Exception e){
            System.out.println("[FAIL] conn database fail." + e.getMessage());
        }
    }

    public void disConn(Connection conn) throws SQLException{
        if(conn != null){
            conn.close();
        }
    }
}
```

返回结果为 192，不符合预期（4 * 16 Bytes）：

```
bit_length
----------
192
```

修改连接参数后，重新执行以上代码。

```
static String url = "jdbc:vexdb://xxx.xxx.xxx.xx:xxxx/postgres?loggerLevel=OFF&characterEncoding=gbk&allowEncodingChanges=true";
```

返回结果如下，符合预期：

```
bit_length
----------
64
```

示例2：使用 dump 函数查看编码

前提条件：在终端、服务端字符集为 utf8 环境执行以下操作。

1. 设置 client_encoding 为 UTF-8。

```
SET client_encoding = 'utf8';
```

2. 使用 dump 函数，查看 ‘数智引航’ 的编码。

```
SELECT dump('数智引航');
```

返回结果如下，共计 12 字节，因此占用 4 个字符：

```
                            dump
-------------------------------------------------------------
Typ=1 Len=12: 230 181 183 233 135 143 230 149 176 230 141 174
(1 row)
```

3. 设置 client_encoding 为 GBK。

```
SET client_encoding = 'gbk';
```

4. 再次使用 dump 函数，查看 ‘数智引航’ 的编码。

```
SELECT dump('数智引航');
```

返回结果如下，共计 18 字节，因此占 6 个字符：

```
                                    dump
--------------------------------------------------------------------------------
Typ=1 Len=18: 229 168 180 231 131 189 229 153 186 233 143 129 231 137 136 229 181 129
(1 row)
```

示例3：不同编码字符集下 char 类型允许存储的字符数量

前提条件：在终端、服务端字符集为 utf8 环境执行以下操作。

1. 创建测试表。

```
CREATE TABLE chrset(c1 char(4 character));
```

2. 设置当前客户端字符集为 utf8，向表中插入数据。

```
SET client_encoding = 'utf8';
INSERT INTO chrset values('数智引航');
```

插入成功：

```
INSERT 0 1
```

3. 再设置当前客户端字符集为 GBK，向表中插入数据。

```
SET client_encoding = 'gbk';
INSERT INTO chrset values('数智引航');
```

插入失败，报错插入值长度超过了 char(4) 的限制。

```
ERROR:  value too long for type character(4)
CONTEXT:  referenced column: c1
```

4. 修改表列定义，将 char(4 character) 列修改为 char(6 character)。

```
ALTER TABLE chrset ALTER c1 TYPE char(6 character);
```

5. 重新向表中插入数据。

```
INSERT INTO chrset values('数智引航');
```

返回结果如下，表示插入成功。

```
INSERT 0 1
```

排序规则

本节介绍排序规则的概念和配置方式。

基本概念

通常，每一种可排序的数据类型都拥有一个排序规则。本节介绍的排序规则（Collation）特指字符的排序规则。排序规则会影响：

- ORDER BY 子句的排序。
- 比较操作符（如 >、<、= 等）。

排序规则依赖于字符集，不同的字符集支持的排序规则不同。可以通过系统表 PG_COLLATION 查询 VexDB 支持的排序规则。

在初始化数据库时，LC_COLLATE 默认会根据操作系统的 locale 设置进行选择，也可以通过 `--LC-COLLATE` 参数指定。指定 LC_COLLATE 时，需要确保该设置与字符集能够匹配。

使用方式

VexDB 数据库允许在查询中通过 COLLATE 子句指定查询使用的排序规则。

```
SELECT expr COLLATE collation;
```

- 不指定 COLLATE 时，使用对象的排序规则。

  ```
  SELECT col < 'expr' FROM tbl;
  ```

- 在括号中指定 COLLATE 时，使用显式指定的排序规则。

  ```
  SELECT col < ('expr' COLLATE 'zh_CN.utf8') FROM tbl;
  ```

  > **说明**
  >
  > 通过 COLLATE 指定的排序规则必须为当前 server_encoding 字符集支持的排序规则。如上述语句在 server_encoding 为 zh_CN.gbk 时，则会报错： `ERROR: collation "zh_CN.utf8" for encoding "GBK" does not exist` 。

- 若解析器无法确定使用哪种排序规则，则会报错。

```
CREATE TABLE tbl(col1 text COLLATE 'zh_CN',col2 text COLLATE 'en_US');
INSERT INTO tbl VALUES ('vexdb','vexdb');
SELECT col1 < col2 from tbl;
```

报错为：

```
ERROR:  could not determine which collation to use for string comparison
```

对于上述情况，可以通过显式指定 COLLATE 关键字规避。

```
SELECT col1 < col2 COLLATE 'zh_CN' from tbl;
```

- 对于不涉及排序的函数或操作符，即使表达式前后的排序规则不同，也不会产生影响。

```
CREATE TABLE tbl(col1 text COLLATE 'zh_CN',col2 text COLLATE 'en_US');
INSERT INTO tbl VALUES ('vexdb','vexdb');
SELECT col1 || col2 from tbl;
```

配置方式

允许配置排序规则的对象

- VexDB 允许为数据库、列级别设置排序规则。其中优先级为：

```
显式指定 > 列 > 数据库
```

- 为索引指定排序规则时，仅显式指定排序规则时，查询计划才会使用该索引。有关该行为的详信息，参见示例1：为索引指定排序规则。

初始化排序规则

在初始化数据库时，LC_COLLATE 默认会使用操作系统 locale.conf 中的 LC_COLLATE 配置，用户可以在初始化数据库时通过 `--lc-collate` 参数指定。指定 `--lc-collate` 时，需要确保该设置与字符集能够匹配。

方式一：

1. 修改 locale.conf，将 LANG 设置为需要使用的语言设置。例如 “zh_CN.gbk” 。

```
LANG = zh_CN.GBK
```

2. 重启操作系统，使配置生效。

```
locale | grep 'LANG\|LC_COLLATE\|LC_CTYPE'
```

上述命令返回结果如下，表示操作系统支持 zh_CN.gbk 排序规则：

```
LANG=zh_CN.gbk
LC_CTYPE="zh_CN.gbk"
LC_COLLATE="zh_CN.gbk"
```

3. 使用 vb_initdb 命令行工具初始化数据库。

```
vb_initdb -D $GAUSSHOME --nodename=vb
```

方式二：

使用 vb_initdb 命令行工具显式指定排序规则。

```
vb_initdb -D $GAUSSHOME \
          --nodename=vb \
          --encoding=GBK \
          --lc-collate=C \
          --lc-ctype=C
```

数据库级排序规则

VexDB 数据库支持 CREATE DATABASE 语句在创建数据库时使用 LC_COLLATE 指定排序规则。LC_COLLATE 在数据库创建后不允许修改。

```
CREATE DATABASE database_name
             ENCODING [=] encoding
             LC_COLLATE [=] lc_collate
             LC_CTYPE [=] lc_ctype;
```

当新建数据库 Encoding、LC_Collate 或 LC_Ctype 与模板数据库（SQL_ASCII）不匹配（为 GBK、UTF8 或 LATIN1 等）时，必须指定 template [=] template0。

参数说明

- database_name

  要创建的数据库名称。

- encoding

  指定数据库使用的字符编码，可以是字符串（如’ SQL_ASCII’ ）、整数编号。

  不指定时，默认使用模版数据库的编码。模板数据库 template0 和 template1 的编码默认与操作系统环境相关。template1 不允许修改字符编码，因此若要变更编码，请使用 template0 创建数据库。

  指定新的数据库字符集编码必须与所选择的本地环境中（LC_COLLATE 和 LC_CTYPE）的设置兼容。

- lc_collate

  指定新数据库使用的字符集。例如，通过lc_collate = ’ zh_CN.gbk’ 设定该参数。

  该参数的使用会影响到对字符串的排序顺序（如使用 ORDER BY 执行，以及在文本列上使用索引的顺序）。默认是使用模板数据库的排序顺序。

  取值范围：操作系统支持的字符集。

- lc_ctype

  指定新数据库使用的字符分类。例如，通过 lc_ctype = ‘zh_CN.gbk’ 设定该参数。该参数的使用会影响到字符的分类，如大写、小写和数字。默认是使用模板数据库的字符分类。

  取值范围：操作系统支持的字符分类。

> **说明**
>
> 对于 lc_collate 和 lc_ctype 参数的取值范围，取决于本地环境支持的字符集。 例如：在 Linux 操作系统上，可通过 `locale -a` 命令获取操作系统支持的字符集列表，在应用 lc_collate 和 lc_ctype 参数时可从中选择用户需要的字符集和字符分类。

数据库的排序规则存储在 PG_DATABASE 系统表中。 可以使用 vsql 的 -l 选项或 命令列出这些排序规则。

```
vsql -l
```

列级排序规则

VexDB 数据库支持在 CREATE TABLE 语句为列指定排序规则。支持通过 ALTER TABLE 修改列的排序规则。

```
CREATE TABLE table_name ( column_name data_type COLLATE collation
                        [, ...] );
```

参数说明

- table_name

要创建的表名。

- column_name

  要创建的表名。

- data_type

  字段的数据类型。

- collation

  指定列的排序规则。指定排序规则的列必须是可排序的数据类型。

  如果没有指定，则使用默认的排序规则。排序规则可以使用 select * from pg_collation; 命令从 pg_collation 系统表中查询，默认的排序规则为查询结果中以 default 开始的行。

> **说明**
>
> 列的排序规则存储在PG_ATTRIBUTE系统表中。可以通过以下 SQL 查询 tbl 表上各列的排序规则。

```
WITH foo AS (SELECT co.oid,co.collname FROM pg_collation co
           JOIN pg_namespace nco ON co.collnamespace = nco.oid
           WHERE nco.nspname <> 'pg_catalog'::name
               OR co.collname <> 'default'::name)
   SELECT tbl.relname AS table_name,
       col.attname AS column_name,
       foo.collname::varchar(100) AS "collation"
   FROM pg_attribute col
   JOIN pg_class tbl ON col.attrelid = tbl.oid
   LEFT JOIN foo ON col.attcollation = foo.oid
   WHERE col.attnum > 0
   AND col.attisdropped = false
  AND relname = 'tbl';
```

示例

示例1：为索引指定排序规则

1. 创建测试表，其中 c2 列排序规则指定为 C。

```
CREATE TABLE coll_ind(c1 int,c2 text COLLATE 'C');
```

2. 在测试表上创建索引，指定排序规则为 zh_CN。

```
CREATE INDEX coll_idx ON coll_ind(c2 COLLATE 'zh_CN');
```

3. 不显示指定排序规则，查看执行计划。

```
EXPLAIN (costs off,verbose) SELECT * FROM coll_ind WHERE c2 = 'a';
```

返回结果如下：

```
           QUERY PLAN
-------------------------------------
Seq Scan on public.coll_ind
Output: c1, c2
Filter: (coll_ind.c2 = 'a'::text)
(3 rows)
```

4. 显式指定排序规则，查看执行计划。

```
EXPLAIN (costs off,verbose) SELECT * FROM coll_ind WHERE c2 = 'a' COLLATE 'zh_CN';
```

返回结果如下，表示执行计划使用了索引：

```
                       QUERY PLAN
-------------------------------------------------------------
Bitmap Heap Scan on public.coll_ind
Output: c1, c2
Recheck Cond: (coll_ind.c2 = 'a'::text COLLATE "zh_CN")
->  Bitmap Index Scan on coll_idx
        Index Cond: (coll_ind.c2 = 'a'::text COLLATE "zh_CN")
(5 rows)
```

5. 清理测试环境。

```
DROP TABLE coll_ind;
```

示例2：指定数据库的排序规则

1. 在默认字符集为 utf8 的数据库实例中，创建排序规则为 zh_CN.gbk 的数据库。

```
CREATE DATABASE gbk LC_COLLATE 'zh_CN.gbk';
```

返回结果如下，报错 utf8 字符集与 zh_CN.gbk 排序规则不匹配：

```
ERROR:  encoding "UTF8" does not match locale "zh_CN.gbk"
```

2. 在创建数据库命令中增加 ENCODING 选项。

```
CREATE DATABASE gbk LC_COLLATE 'zh_CN.gbk' ENCODING 'gbk';
```

返回结果如下，报错 GBK 字符集与区域 en_US.UTF8 不匹配：

```
ERROR:  encoding "GBK" does not match locale "en_US.UTF-8"
```

3. 在创建数据库命令中增加 LC_CTYPE 选项。

```
CREATE DATABASE gbk LC_COLLATE 'zh_CN.gbk' ENCODING 'gbk' LC_CTYPE 'zh_CN.gbk';
```

返回结果如下，创建数据库成功。

```
CREATE DATABASE
```

4. 清理测试环境。

```
DROP DATABASE gbk;
```

示例3：指定列的排序规则

1. 创建测试表，并插入数据。

```
CREATE TABLE coll_tbl(c1 text COLLATE 'en_US',c2 text COLLATE 'C');
INSERT INTO coll_tbl VALUES ('vexdb','vexdb');
```

2. 查询 c1 < c2。

```
SELECT c1 < c2 FROM coll_tbl;
```

返回结果如下，由于执行器无法选择使用哪一种排序规则，返回报错。

```
ERROR:  could not determine which collation to use for string comparison
```

3. 在查询中显式指定排序规则。

```
SELECT c1 < c2 collate 'en_US' from coll_tbl;
```

返回结果如下：

```
?column?
----------
f
(1 row)
```

4. 清理测试环境。

```
DROP TABLE coll_tbl;
```

字符分类

基本概念

字符分类（Character Type）用于系统判断字符的处理方式。例如一个字符是单字节还是多字节，一个字符是字母还是符号等。

在数据库中，LC_CTYPE 的配置会影响大小转换，例如不被视为字母的字符，就不会被 `LOWER()` 或 `UPPER()` 函数转换大小写。有关该行为的示例，参考 示例1：不同字符分类的行为。

配置方式

初始化字符分类

在初始化数据库时，LC_CTYPE 默认会根据操作系统的 locale 设置进行选择，也可以通过 –lc-ctype 参数指定。指定 –lc-ctype 时，需要确保该设置与字符集能够匹配。

数据库级字符分类

配置数据库级字符分类的方式与配置排序规则的方式相同，请查阅 数据库级排序规则。

- LC_CTYPE 在数据库创建后不能修改。
- 数据库的字符分类存储在 PG_DATABASE 系统表中。 可以使用 vsql 的 `-l` 选项或 `\l` 命令列出这些字符分类。

```
vsql -l
```

示例

示例1：不同字符分类的行为

1. 创建测试数据库，分别指定 LC_CTYPE 为 zh_CN.gb18030 与 C。

```
CREATE DATABASE chr1 ENCODING 'GB18030' LC_COLLATE 'zh_CN.gb18030' LC_CTYPE 'zh_CN.gb18030';
CREATE DATABASE chr2 ENCODING 'GB18030' LC_COLLATE 'C' LC_CTYPE 'C';
```

2. 切换到 chr1，执行以下语句：

```
\c chr1
SELECT lower('Ａ');
```

返回结果如下：

```
lower
-------
ａ
(1 row)
```

3. 切换到 chr2，执行以下语句：

```
\c chr2
SELECT lower('Ａ');
```

返回结果如下：

```
lower
-------
Ａ
(1 row)
```

这是由于 C 语言的字符分类不会将全角英文字符视为字母，因此不会进行大小写转换。

4. 清理测试环境。

```
DROP DATABASE chr1;
DROP DATABASE chr2;
```

设置参数

背景信息

VexDB 提供了许多运行参数，配置这些参数可以影响数据库系统的行为。用户可以通过 GUC（Grand Unified Configuration）参数来控制 VexDB 数据库的部署形态和运行行为。

VexDB 提供了多种修改 GUC 参数的方法，用户可以方便的针对数据库、用户、会话进行设置。

注意事项

- 参数名称不区分大小写。
- 参数取值有整型、浮点型、字符串、布尔型和枚举型五类。
  - 布尔值可以是（on，off）、（true，false）、（yes，no）或者（1，0），且不区分大小写。
  - 枚举类型的取值是在系统表pg_settings的enumvals字段取值定义的。
- 对于有单位的参数，在设置时请指定单位，否则将使用默认的单位。
  - 参数的默认单位在系统表PG_SETTINGS的unit字段定义的。
  - 内存单位有：KB（千字节）、MB（兆字节）和GB（吉字节）。
  - 时间单位：ms（毫秒）、s（秒）、min（分钟）、h（小时）和d（天）。

查看参数当前取值

VexDB 安装后，有一套默认的运行参数，为了使 VexDB 与业务的配合度更高，用户需要根据业务场景和数据量的大小进行 GUC 参数调整。

方法一：使用 SHOW 命令

`SHOW` 将显示当前运行时参数的数值。

- 使用如下命令查看单个参数，显示数据库版本信息的参数：

  ```
  SHOW configuration_parameter;
  ```

  configuration_parameter 是运行时参数的名称。

- 使用如下命令查看所有参数：

```
SHOW ALL;
```

示例：使用 `SHOW` 命令查看服务器的版本号。

```
SHOW server_version;
```

返回结果如下：

```
server_version
----------------
 9.2.4
(1 row)
```

方法二：使用 PG_SETTINGS 视图

PG_SETTINGS 视图显示数据库运行时参数的相关信息。

- 使用如下命令查看单个参数：

```
SELECT * FROM pg_settings WHERE NAME='server_version';
```

- 使用如下命令查看所有参数：

```
SELECT * FROM pg_settings;
```

GUC 参数分类

VexDB 提供了六类 GUC 参数，具体分类和设置方式如下：

表1 VeXDB 参数类型

参数类型	说明	设置方式
INTERNAL	固定参数，在创建数据库的时候确定，用户无法修改，只能通过show语法或者pg_settings视图进行查看。	无
POSTMASTER	数据库服务端参数，在数据库启动时确定，可以通过配置文件指定。	支持表2中的方式一、方式四。
SIGHUP	数据库全局参数，可在数据库启动时设置或者在数据库启动后，发送指令重新加载。	支持表2中的方式一、方式二、方式四。
BACKEND	会话连接参数。在创建会话连接时指定，连接建立后无法修改。连接断掉后参数失效。内部使用参数，不推荐用户设置。	支持表2中的方式一、方式二、方式四。说明：设置该参数后，下一次建立会话连接时生效。
SUSET	数据库管理员参数。可在数据库启动时、数据库启动后或者数据库管理员通过SQL进行设置。	支持表2中的方式一、方式二或由数据库管理员通过方式三设置。
USERSET	普通用户参数。可被任何用户在任何时刻设置。	支持表2中的方式一、方式二或方式三设置。

GUC 参数设置方式

表2 参数设置方式

> **说明**
>
> 使用方式一和方式二设置参数时，若所设参数不属于当前环境，数据库会提示参数不在支持范围内的相关信息。

序号	设置方法
方式一	使用如下命令修改参数，重启数据库后生效。 `gs_guc set -D datadir -c "paraname=value"` 说明：如果参数是一个字符串变量，则使用<code>-c parameter="'value'"</code>或者使用<code>-c "parameter = 'value'"</code>。 使用以下命令在数据库节点上同时设置某个参数。<pre>gs_guc set -N all -I all -c "paraname=value"</pre> 使用以下命令在数据库节点上设置cm_agent某个参数。<pre>gs_guc set -Z cmagent -c "paraname=value"</pre>或者<pre>gs_guc set -Z cmagent -N all -I all -c "paraname=value"</pre>使用以下命令在数据库节点上设置cm_serve

方式二	`gs_guc reload -D datadir -c "paraname=value"`
	说明：使用以下命令在数据库节点上同时设置某个参数。
	`gs_guc reload -N all -I all -c "paraname=value"`

方式三	修改指定数据库、用户、会话级别的参数。设置数据库级别的参数，在下次会话中生效：
	`ALTER DATABASE dbname SET paraname TO value;`
	设置用户级别的参数，在下次会话中生效：
	`ALTER USER username SET paraname TO value;`
	设置会话级别的参数，退出会话后，设置将失效：
	`SET paraname TO value;`
	说明：SET 设置的会话级参数优先级最高，其次是 ALTER 设置的，其中 ALTER USER 设置的参数值优先级高于 ALTER DATABASE 设置，这三种设置方式设置的优先级都高于 gs_guc 设置方式。

方式四	使用 ALTER SYSTEM SET 修改数据库参数。 设置POSTMASERT级别的参数，重启数据库后生效： `ALTER SYSTEM SET paraname TO value;` 设置SIGHUP级别的参数，立刻生效（实际等待线程重新加载参数略有延迟）： `ALTER SYSTEM SET paraname TO value;`

设置BACKEND级别的参数，在下次会话中生效：

```
ALTER SYSTEM SET paraname TO value;
```

查看系统表和系统视图

除了创建的表以外，数据库还包含很多系统表。这些系统表包含 VexDB 安装信息以及 VexDB 上运行的各种查询和进程的信息。可以通过查询系统表来收集有关数据库的信息。

系统表和系统视图中每个表的说明指出了表是对所有用户可见还是只对初始化用户可见。必须以初始化用户身份登录才能查询只对初始化用户可见的表。

VexDB 提供了以下类型的系统表和视图：

- 继承自 PG 的系统表和视图，这类系统表和视图具有 PG 前缀。
- 继承自 OG 的系统表和视图，这类系统表和视图具有 GS 前缀。

查看数据库中包含的表

例如，在 PG_TABLES 系统表中查看public schema中包含的所有表。

```
SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public';
```

返回结果如下：

```
   tablename
-------------------
 err_hr_staffs
 test
 err_hr_staffs_ft3
 web_returns_p1
 mig_seq_table
 films4
(6 rows)
```

查看数据库对象

通过 PG_CLASS 系统表存储数据库对象信息及其之间的关系。

```
SELECT * FROM PG_CLASS WHERE relname = 'object_name';
```

其中 object_name 表示待查询的对象名称。

查看数据库用户

通过系统表 PG_USER 可以查看数据库中所有用户的列表，还可以查看用户ID（USESYSID）和用户权限。

```
SELECT * FROM PG_USER;
```

结果类似这样：

```
usename             | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |   respool
    | parent | spacelimit | useconfig
--------------------------------+----------+-------------+----------+-----------+---------+----------+----------+----------+----------
----+--------+------------+-----------
dfc22b86afbd9a745668c3ecd0f15ec18 |    17107 | f           | f        | f         | f       | ** |          |          | default_p
ool |      0 |            |
guest                          |    17103 | f           | f        | f         | f       | ** |          |          | default_p
ool |      0 |            |
vexdb                          |       10 | t           | t        | t         | t       | ** |          |          | default_p
ool |      0 |            |
vexdb                          |    16404 | f           | f        | f         | f       | ** |          |          | default_p
ool |      0 |            |
lily                           |    16482 | f           | f        | f         | f       | ** |          |          | default_p
ool |      0 |            |
jack                           |    16478 | f           | f        | f         | f       | ** |          |          | default_p
ool |      0 |            |
(6 rows)
```

查看和停止正在运行的查询语句

通过系统视图 PG_STAT_ACTIVITY 可以查看正在运行的查询语句。方法如下：

1. 设置参数track_activities为on。

   ```
   SET track_activities = on;
   ```

   当此参数为on时，数据库系统才会收集当前活动查询的运行信息。

2. 查看正在运行的查询语句。以查看正在运行的查询语句所连接的数据库名、执行查询的用户、查询状态及查询对应的PID为例：

   ```
   SELECT datname, usename, state,pid FROM pg_stat_activity;
   ```

   结果类似如下这样：

```
datname  | usename | state  |       pid
----------+---------+--------+-----------------
postgres | Ruby    | active | 140298793514752
postgres | Ruby    | active | 140298718004992
postgres | Ruby    | idle   | 140298650908416
postgres | Ruby    | idle   | 140298625742592
postgres | vexdb   | active | 140298575406848
(5 rows)
```

如果state字段显示为idle，则表明此连接处于空闲，等待用户输入命令。

如果仅需要查看非空闲的查询语句，则使用如下命令查看：

```
SELECT datname, usename, state FROM pg_stat_activity WHERE state != 'idle';
```

3. 若需要取消运行时间过长的查询，通过 PG_TERMINATE_BACKEND 函数，根据线程 ID 结束会话。

```
SELECT PG_TERMINATE_BACKEND(139834759993104);
```

显示类似如下信息，表示结束会话成功。

```
PG_TERMINATE_BACKEND
----------------------
    t
(1 row)
```

显示类似如下信息，表示用户执行了结束当前会话的操作。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
```

> **说明**
>
> vsql客户端使用 PG_TERMINATE_BACKEND 函数结束当前会话后台线程时，客户端不会退出而是自动重连。即还会返回 “The connection to the server was lost. Attempting reset: Succeeded.”

查看数据库参数配置

通过 PG_SETTINGS 视图显示数据库运行时参数的相关信息。

```
SELECT * FROM PG_SETTINGS WHERE name = 'parameter_name';
```

其中 parameter_name 是待查询的参数名称。

创建和管理数据库

前提条件

用户必须拥有数据库创建的权限或者是数据库的系统管理员权限才能创建数据库，赋予创建数据库的权限参见访问控制。

背景信息

- 初始时，VexDB 包含两个模板数据库 template0、template1，以及默认的系统数据库 postgres。
- CREATE DATABASE 语法实际上通过拷贝模板数据库来创建新数据库。默认情况下，拷贝 template0。请避免使用客户端或其他手段连接及操作两个模板数据库。

> **说明**
>
> - 模板数据库中没有用户表，可通过系统表 PG_DATABASE 查看模板数据库属性。
> - 模板 template0 不允许用户连接；模板 template1 只允许数据库初始用户和系统管理员连接，普通用户无法连接。

- 数据库系统中会有多个数据库，但是客户端程序一次只能连接一个数据库。也不能在不同的数据库之间相互查询。一个 VexDB 实例中存在多个数据库时，需要通过 `-d` 参数指定相应的数据库进行连接。

注意事项

- 数据库名称遵循SQL标识符的一般规则。当前角色自动成为此新数据库的所有者。
- 如果一个数据库系统用于承载相互独立的用户和项目，建议把它们放在不同的数据库里。
- 如果项目或者用户是相互关联的，并且可以相互使用对方的资源，则应该把它们放在同一个数据库里，但可以规划在不同的模式中。模式只是一个纯粹的逻辑结构，某个模式的访问权限由权限系统模块控制。
- 创建数据库时，若数据库名称长度超过63字节，server端会对数据库名称进行截断，保留前63个字节，因此建议数据库名称长度不要超过63个字节。
- 如果数据库的编码为SQL_ASCII（可以通过show server_encoding;命令查看当前数据库存储编码），则在创建数据库对象时，如果对象名中含有多字节字符（例如中文），超过数据库对象名长度限制（63字节）的时候，数据库将会将最后一个字节（而不是字符）截断，可能造成出现半个字符的情况。

  针对这种情况，请遵循以下条件：

- 保证数据对象的名称不超过限定长度。
- 修改数据库的默认存储编码集（server_encoding）为utf-8编码集。
- 不要使用多字节字符做为对象名。
- 创建的数据库总数目不得超过128个。
- 如果出现因为误操作导致在多字节字符的中间截断而无法删除数据库对象的现象，请使用截断前的数据库对象名进行删除操作，或将该对象从各个数据库节点的相应系统表中依次删掉。

示例

创建数据库

- 使用如下命令创建一个新的表空间tpcds_local。

```
CREATE TABLESPACE tpcds_local RELATIVE LOCATION 'tablespace/tablespace' ;
```

- 使用如下命令创建一个新的数据库db_tpcc。

```
CREATE DATABASE db_tpcc WITH TABLESPACE = tpcds_local;
```

查看数据库

- 使用 `\l` 元命令查看数据库系统的数据库列表。

```
\l
```

- 使用如下命令通过系统表 pg_database 查询数据库列表。

```
SELECT datname FROM pg_database;
```

修改数据库

用户可以使用如下命令修改数据库属性（比如：owner、名称和默认的配置属性）。

- 使用以下命令为数据库设置默认的模式搜索路径。

```
ALTER DATABASE db_tpcc SET search_path TO pa_catalog,public;
```

- 使用如下命令修改数据库表空间。

```
ALTER DATABASE db_tpcc SET TABLESPACE tpcds_local;
```

- 使用如下命令为数据库重新命名。

```
ALTER DATABASE db_tpcc RENAME TO human_tpcds;
```

删除数据库

用户可以使用DROP DATABASE命令删除数据库。这个命令删除了数据库中的系统目录，并且删除了磁盘上带有数据的数据库目录。用户必须是数据库的owner或者系统管理员才能删除数据库。当有人连接数据库时，删除操作会失败。删除数据库时请先连接到其他的数据库。

使用如下命令删除数据库和表空间：

```
DROP DATABASE human_tpcds;
DROP TABLESPACE tpcds_local;
```

创建和管理表空间

功能描述

通过使用表空间，管理员可以控制一个数据库安装的磁盘布局。这样有以下优点：

- 如果初始化数据库所在的分区或者卷空间已满，又不能逻辑上扩展更多空间，可以在不同的分区上创建和使用表空间，直到系统重新配置空间。
- 表空间允许管理员根据数据库对象的使用模式安排数据位置，从而提高性能。
- 一个频繁使用的索引可以放在性能稳定且运算速度较快的磁盘上，比如一种固态设备。
- 一个存储归档的数据，很少使用的或者对性能要求不高的表可以存储在一个运算速度较慢的磁盘上。
- 管理员通过表空间可以设置占用的磁盘空间。用于在和其他数据共用分区的时候，防止表空间占用相同分区上的其他空间。
- 表空间对应于一个文件系统目录，假定数据库节点数据目录/pg_location/mount1/path1是用户拥有读写权限的空目录。

  使用表空间配额管理会使性能有30%左右的影响，MAXSIZE指定每个数据库节点的配额大小，误差范围在500MB以内。请根据实际的情况确认是否需要设置表空间的最大值。

VexDB 自带了两个表空间：pg_default 和 pg_global。

- 默认表空间pg_default：用来存储非共享系统表、用户表、用户表index、临时表、临时表index、内部临时表的默认表空间。对应存储目录为实例数据目录下的base目录。

- 共享表空间pg_global：用来存放共享系统表的表空间。对应存储目录为实例数据目录下的global目录。

注意事项

- 启用单用户表空间最低配额后，可以给用户设置并绑定表空间的最低配额。在最低配额值内，不允许其他用户使用。举例说明，例如用户A和用户B均有使用表空间T的权限，为用户A设置最低配额10MB，且A已经使用了5MB。当表空间容量为6MB时，需要保证A用户至少能够申请总和10MB，此时A用户已经有了5MB，所以B用户最多只能申请1MB。当表空间容量为5MB时，B用户没有可利用的资源。
- 给用户A配置了表空间T最低配额，如果A还使用其他表空间，将导致最低配额限制值统计不准确。

操作步骤

创建表空间

1. 执行如下命令创建用户jack。

```
CREATE USER jack IDENTIFIED BY 'Vbase@123';
```

2. 执行如下命令创建表空间，其中“fastspace”为新创建的表空间，“tablespace/tablespace_1”是用户拥有读写权限的空目录。

```
CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace_1';
```

3. 数据库系统管理员执行如下命令将“fastspace”表空间的访问权限赋予数据用户jack。

```
GRANT CREATE ON TABLESPACE fastspace TO jack;
```

在表空间中创建对象

如果用户拥有表空间的 CREATE 权限，就可以在表空间上创建数据库对象，比如：表和索引等。

以创建表为例。

- **方式1：** 执行如下命令在指定表空间创建表。

```
CREATE TABLE foo(i int) TABLESPACE fastspace;
```

- **方式2：** 先使用set default_tablespace设置默认表空间，再创建表。

假设设置“fastspace”为默认表空间，然后创建表foo2。

```
SET default_tablespace = 'fastspace';
CREATE TABLE foo2(i int);
```

查询表空间

- **方式1：** 检查pg_tablespace系统表。如下命令可查到系统和用户定义的全部表空间。

```
SELECT spcname FROM pg_tablespace;
```

- **方式2：** 使用vsql程序的元命令查询表空间。

```
\db
```

查询表空间使用率

1. 查询表空间的当前使用情况。

```
SELECT PG_TABLESPACE_SIZE('fastspace');
```

返回结果如下，其中2146304表示表空间的大小，单位为字节：

```
pg_tablespace_size
--------------------
            2146304
(1 row)
```

2. 计算表空间使用率。

```
表空间使用率=PG_TABLESPACE_SIZE/表空间所在目录的磁盘大小
```

重命名表空间

执行如下命令对表空间fastspace重命名为fspace。

```
ALTER TABLESPACE fastspace RENAME TO fspace;
```

删除表空间

> **说明**
>
> - 用户必须是表空间的owner或者系统管理员才能删除表空间。
> - 在删除表空间前，必须保证其上所有的数据库对象已经被清空。如果仍然有数据文件放在该表空间上，则DROP命令执行失败。

1. 执行如下命令删除用户jack。

```
DROP USER jack CASCADE;
```

2. 执行如下命令删除表foo和foo2。

```
DROP TABLE foo;
DROP TABLE foo2;
```

3. 执行如下命令删除表空间fspace。

```
DROP TABLESPACE fspace;
```

设置表空间配额

创建单用户指定该表空间设置最低保留值小于等于表空间最大配额。

1. 开启参数。

> **说明**
>
> - 表空间最低配额由参数vb_enable_user_reserve_space来控制，该参数用于控制是否开启单用户表空间最低配额，默认值off（关闭）。 设置为on后，表示启用单用户表空间最低配额。该参数重新读取配置文件生效（可通过 `vb_ctl reload` 或者在 vsql 中执行 `select pg_reload_conf()` 来重新读取配置文件），不需要重启数据库实例。 开启参数使用 `alter system set vb_enable_user_reserve_space = on;` 命令。
> - vb_enable_perm_space用于对持久化磁盘空间的统计和检查功能相关参数，例如单用户表空间最低配额限制、单条sql使用空间限制。 on表示开启，off表示关闭。 开启参数使用 `alter system set enable_perm_space = on;` 命令。

```
alter system set user_reserve_space = on;
alter system set enable_perm_space = on;
```

2. 重新读取配置文件。

```
select pg_reload_conf();
```

3. 创建表空间。

```
CREATE TABLESPACE ts2 RELATIVE LOCATION 'tablespace/tablespace_2' MAXSIZE '20k';
```

4. 创建单用户指定该表空间设置最低保留值小于等于表空间最大配额。

```
CREATE USER jack2 IDENTIFIED BY 'Vbase@123' reserve space '10k' default tablespace ts2;
```

5. 清理环境。

```
ALTER SYSTEM SET user_reserve_space = default;
ALTER SYSTEM SET enable_perm_space = default;
DROP USER jack2;
DROP TABLESPACE ts2;
```

数据库对象管理

- 创建和管理 schema
- 创建和管理普通表
- 创建和管理分区表
- 创建和管理索引
- 创建和管理视图
- 创建和管理序列
- 创建和管理用户与角色

创建和管理 schema

背景信息

schema 又称作模式。通过管理 schema，允许多个用户使用同一数据库而不相互干扰，可以将数据库对象组织成易于管理的逻辑组，同时便于将第三方应用添加到相应的 schema 下而不引起冲突。管理 schema 包括：创建 schema、使用 schema、删除 schema、设置 schema 的搜索路径以及schema的权限控制。

注意事项

- VexDB 包含一个或多个已命名数据库。用户和用户组在VexDB 范围内是共享的，但是其数据并不共享。任何与服务器连接的用户都只能访问连接请求里声明的那个数据库。
- 一个数据库可以包含一个或多个已命名的schema，schema又包含表及其他数据库对象，包括数据类型、函数、操作符等。同一对象名可以在不同的schema中使用而不会引起冲突。例如，schema1和schema2都可以包含一个名为mytable的表。
- 和数据库不同，schema不是严格分离的。用户根据其对schema的权限，可以访问所连接数据库的schema中的对象。进行schema权限管理首先需要对数据库的权限控制进行了解。
- 不能创建以PG_为前缀的schema名，该类schema为数据库系统预留的。
- 在每次创建新用户时，系统会在当前登录的数据库中为新用户创建一个同名Schema。对于其他数据库，若需要同名Schema，则需要用户手动创建。
- 通过未修饰的表名（名称中只含有表名，没有“schema名”）引用表时，系统会通过search_path（搜索路径）来判断该表是哪个schema下的表。pg_temp和pg_catalog始终会作为搜索路径顺序中的前两位，无论二者是否出现在search_path中，或者出现在search_path中的任何位置。search_path（搜索路径）是一个schema名列表，在其中找到的第一个表就是目标表，如果没有找到则报错。（某个表即使存在，如果它的schema不在search_path中，依然会查找失败）在搜索路径中的第一个schema叫做“当前schema”。它是搜索时查询的第一个schema，同时在没有声明schema名时，新创建的数据库对象会默认存放在该schema下。
- 每个数据库都包含一个pg_catalog schema，它包含系统表和所有内置数据类型、函数、操作符。pg_catalog是搜索路径中的一部分，始终在临时表所属的模式后面，并在search_path中所有模式的前面，即具有第二搜索优先级。这样确保可以搜索到数据库内置对象。如果用户需要使用和系统内置对象重名的自定义对象时，可以在操作自定义对象时带上自己的模式。

示例

创建、修改和删除Schema

- 要创建Schema，请参考CREATE SCHEMA。默认初始用户和系统管理员可以创建Schema，其他用户需要具备数据库的CREATE权限才可以在该数据库中创建Schema，赋权方式请参考章节GRANT中将数据库的访问权限赋予指定的用户或角色中的语法。
- 要更改Schema名称或者所有者，请参考ALTER SCHEMA。Schema所有者可以更改Schema。
- 要删除Schema及其对象，请参考DROP SCHEMA。Schema所有者可以删除Schema。

- 要在Schema内创建表，请以schema_name.table_name格式创建表。不指定schema_name时，对象默认创建到搜索路径中的第一个Schema内。
- 要查看Schema所有者，请对系统表PG_NAMESPACE和PG_USER执行如下关联查询。语句中的schema_name请替换为实际要查找的Schema名称。

```
SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s, pg_user u WHERE nspname='schema_name' AND s.nspowner = u.usesysid;
```

- 要查看所有Schema的列表，请查询PG_NAMESPACE系统表。

```
SELECT * FROM pg_namespace;
```

- 要查看属于某Schema下的表列表，请查询系统视图PG_TABLES。例如，以下查询会返回Schema PG_CATALOG中的表列表。

```
SELECT distinct(tablename),schemaname from pg_tables where schemaname = 'pg_catalog';
```

搜索路径

搜索路径定义在 search_path 参数中，参数取值形式为采用逗号分隔的 Schema 名称列表。如果创建对象时未指定目标 Schema，则将该对象会被添加到搜索路径中列出的第一个 Schema 中。当不同 Schema 中存在同名的对象时，查询对象未指定 Schema 的情况下，将从搜索路径中包含该对象的第一个 Schema 中返回对象。

- 要查看当前搜索路径，请使用SHOW命令。

```
SHOW SEARCH_PATH;
```

返回结果为：

```
search_path
----------------
"$user",public
(1 row)
```

> **说明**
>
> search_path参数的默认值为： `"$user", public` 。 `$user` 表示与当前会话用户名同名的Schema名，如果这样的模式不存在，$user将被忽略。所以默认情况下，用户连接数据库后，如果数据库下存在同名Schema，则对象会添加到同名Schema下，否则对象被添加到Public Schema下。

- 要更改当前会话的默认Schema，请使用 SET 命令。

执行如下命令将搜索路径设置为myschema、public，首先搜索myschema。

```
SET SEARCH_PATH TO myschema, public;
```

创建和管理普通表

创建表

表是建立在数据库中的，在不同的数据库中可以存放相同的表。甚至可以通过使用模式在同一个数据库中创建相同名称的表。创建表前请先规划存储模型。

1. 创建普通表，其中c_customer_sk 、c_customer_id、c_first_name和c_last_name是表的字段名，integer、char(5)、char(6)和char(8)分别是这四个字段的类型。

```
CREATE TABLE customer_t1
(
    c_customer_sk       integer,
    c_customer_id       char(5),
    c_first_name        char(6),
    c_last_name         char(8)
);
```

2. 删除表。

```
DROP TABLE customer_t1;
```

向表中插入数据

在创建一个表后，表中并没有数据，在使用这个表之前，需要向表中插入数据。本小节介绍如何使用 INSERT 命令插入一行或多行数据，及从指定表插入数据。

背景信息

服务端与客户端使用不同的字符集时，两者字符集中单个字符的长度也会不同，客户端输入的字符串会以服务端字符集的格式进行处理，所以产生的最终结果可能会与预期不一致。

表1 客户端和服务端设置字符集的输出结果对比

操作过程	服务端和客户端编码一致	服务端和客户端编码不一致
存入和取出过程中没有对字符串进行操作	输出预期结果	输出预期结果（输入与显示的客户端编码必须一致）。
存入取出过程对字符串有做一定的操作（如字符串函数操作）	输出预期结果	根据对字符串具体操作可能产生非预期结果。
存入过程中对超长字符串有截断处理	输出预期结果	字符集中字符编码长度是否一致，如果不一致可能会产生非预期的结果。

上述字符串函数操作和自动截断产生的效果会有叠加效果，例如：在客户端与服务端字符集不一致的场景下，如果既有字符串操作，又有字符串截断，在字符串被处理完以后的情况下继续截断，这样也会产生非预期的效果。详细的示例请参见表2。

```
CREATE TABLE table1(id int, a char(6), b varchar(6),c varchar(6));
CREATE TABLE table2(id int, a char(20), b varchar(20),c varchar(20));
```

表2 示例

编号	服务端字符集	客户端字符集	是否启用自动截断	示例	结果	说明
1	SQL_ASCII	UTF8	是	autoINSERT INTO table1 VALUES(1,reverse('123 A A 78'),reverse('123 A A 78'),reverse('123 A A 78'));	autoid |a|b|c —-+——+——+—— 1 | 87| 87| 87	字符串在服务端翻转后，并进行截断，由于服务端和客户端的字符集不一致，字符 A 在客户端由多个字节表示，结果产生异常。
2	SQL_ASCII	UTF8	是	autoINSERT INTO table1 VALUES(2,reverse('123 A 78'),reverse('123 A 78'),reverse('123 A 78'));	autoid |a|b|c —-+——+——+—— 2 | 873| 873| 873	字符串翻转后，又进行了自动截断，所以产生了非预期的效果。
3	SQL_ASCII	UTF8	是	autoINSERT INTO table1 VALUES(3,'87 A 123','87 A 123','87 A 123');	autoid | a | b | c —-+——-+——-+——- 3 | 87 A 1 | 87 A 1 | 87 A 1	字符串类型的字段长度是客户端字符编码长度的整数倍，所以截断后产生结果正常。
4	SQL_ASCII	UTF8	否	autoINSERT INTO table2 VALUES(1,reverse('123 A A 78'),reverse('123 A A 78'),reverse('123 A A 78')); INSERT INTO table2 VALUES(2,reverse('123 A 78'),reverse('123 A 78'),reverse('123 A 78'));	autoid |a|b|c —-+————————-+———+——— 1 | 87 321| 87 321 | 87 321 2 | 87321| 87321| 87321	与示例1类似，多字节字符翻转之后不再表示原来的字符。

操作步骤

1. 创建测试表。

```
CREATE TABLE test_a4740
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
);
```

2. 向表test_a4740中插入一行：

   数据值是按照这些字段在表中出现的顺序列出的，并且用逗号分隔。通常数据值是文本（常量），但也允许使用标量表达式。

```
INSERT INTO test_a4740(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
```

   如果用户已经知道表中字段的顺序，也可无需列出表中的字段。例如以下命令与上面的命令效果相同。

```
INSERT INTO test_a4740 VALUES (3769, 'hello', 'Grace');
```

如果用户不知道所有字段的数值，可以忽略其中的一些。没有数值的字段将被填充为字段的缺省值。例如：

```
INSERT INTO test_a4740 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');
INSERT INTO test_a4740 VALUES (3769, 'hello');
```

用户也可以对独立的字段或者整个行明确缺省值：

```
INSERT INTO test_a4740 (c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', DEFAULT);
INSERT INTO test_a4740 DEFAULT VALUES;
```

3. 如果需要在表中插入多行，请使用以下命令：

```
INSERT INTO test_a4740 (c_customer_sk, c_customer_id, c_first_name) VALUES
    (6885, 'maps', 'Joes'),
    (4321, 'tpcds', 'Lily'),
    (9527, 'world', 'James');
```

如果需要向表中插入多条数据，除此命令外，也可以多次执行插入一行数据命令实现。但是建议使用此命令可以提升效率。

4. 如果从指定表插入数据到当前表，例如在数据库中创建了一个表test_a4740的备份表test_a4740_1，现在需要将表test_a4740中的数据插入到表test_a4740_1中，则可以执行如下命令。

```
CREATE TABLE test_a4740_1
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
);
INSERT INTO test_a4740_1 SELECT * FROM test_a4740;
```

从指定表插入数据到当前表时，若指定表与当前表对应的字段数据类型之间不存在隐式转换，则这两种数据类型必须相同。

5. 清理环境。

```
DROP TABLE test_a4740_1 CASCADE;
DROP TABLE test_a4740 CASCADE;
```

在删除表的时候，若当前需删除的表与其他表有依赖关系，需先删除关联的表，然后再删除当前表。

更新表中数据

修改已经存储在数据库中数据的行为叫做更新。用户可以更新单独一行、所有行或者指定的部分行。还可以独立更新某个字段，而其它字段则不受影响。

使用 UPDATE 命令更新现有行，需要提供以下三种信息：

- 表的名称和要更新的字段名
- 字段的新值
- 要更新哪些行

SQL通常不会为数据行提供唯一标识，因此无法直接声明需要更新哪一行。但是可以通过声明一个被更新的行必须满足的条件来更新数据行。只有在表里存在主键的时候，才可以通过主键指定一个独立的行。

1. 执行如下命令创建表并插入数据：

```
CREATE TABLE customer_t1
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
);
INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
    (6885, 'maps', 'Joes'),
    (4321, 'tpcds', 'Lily'),
    (9527, 'world', 'James');
```

2. 将表customer_t1中c_customer_sk为9527的地域重新定义为9876：

```
UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;
```

   这里的表名称也可以使用模式名修饰，否则会从默认的模式路径找到这个表。SET后面紧跟字段和新的字段值。新的字段值不仅可以是常量，也可以是变量表达式。

3. 把所有c_customer_sk的值增加100：

```
UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100;
```

   在这里省略了WHERE子句，表示表中的所有行都要被更新。如果出现了WHERE子句，那么只有匹配其条件的行才会被更新。

   在SET子句中的等号是一个赋值，而在WHERE子句中的等号是比较。WHERE条件不一定是相等测试，许多其他的操作符也可以使用。

4. 在一个UPDATE命令中更新更多的字段，方法是在SET子句中列出更多赋值，比如：

```
UPDATE customer_t1 SET  c_customer_id = 'Admin', c_first_name = 'Local' WHERE c_customer_sk = 4421;
```

批量更新或删除数据后，会在数据文件中产生大量的删除标记，查询过程中标记删除的数据也是需要扫描的。故多次批量更新/删除后，标记删除的数据量过大会严重影响查询的性能。建议在批量更新/删除业务会反复执行的场景下，定期执行VACUUM FULL以保持查询性能。

5. 清理环境。

```
DROP TABLE customer_t1 CASCADE;
```

查看数据

1. 执行如下命令创建表并插入数据：

```
CREATE TABLE customer_t1
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
);
INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
    (6885, 'maps', 'Joes'),
    (4321, 'tpcds', 'Lily'),
    (9527, 'world', 'James');
```

2. 执行以下操作。
   - 使用系统表pg_tables查询数据库所有表的信息。

     ```
     SELECT * FROM pg_tables;
     ```

   - 使用vsql的命令查询表的属性。

     ```
     \d+ customer_t1;
     ```

   - 执行如下命令查询表customer_t1的数据量。

     ```
     SELECT count(*) FROM customer_t1;
     ```

   - 执行如下命令查询表customer_t1的所有数据。

```
SELECT * FROM customer_t1;
```

- 执行如下命令只查询字段c_customer_sk的数据。

```
SELECT c_customer_sk FROM customer_t1;
```

- 执行如下命令过滤字段c_customer_sk的重复数据。

```
SELECT DISTINCT( c_customer_sk ) FROM customer_t1;
```

- 执行如下命令查询字段c_customer_sk为3869的所有数据。

```
SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
```

- 执行如下命令按照字段c_customer_sk进行排序。

```
SELECT * FROM customer_t1 ORDER BY c_customer_sk;
```

3. 清理环境。

```
DROP TABLE customer_t1 CASCADE;
```

删除表中数据

在使用表的过程中，可能会需要删除已过期的数据，删除数据必须从表中整行的删除。

SQL不能直接访问独立的行，只能通过声明被删除行匹配的条件进行。如果表中有一个主键，用户可以指定准确的行。用户可以删除匹配条件的一组行或者一次删除表中的所有行。

- 使用DELETE命令删除行，如果删除表table_name中所有col_name字段的值为value的记录：

```
DELETE FROM table_name WHERE col_name = value;
```

- 如果执行如下命令之一，会删除table_name表中所有的行。

```
DELETE FROM table_name;
```

或：

```
TRUNCATE TABLE table_name;
```

> **说明**
>
> 全表删除的场景下，建议使用TRUNCATE，不建议使用DELETE。

- 删除创建的表table_name：

```
DROP TABLE table_name;
```

创建和管理分区表

分区表概述

分区表通过对分区列的判断，把分区列不同的记录放到不同的分区中。查询时可以通过查询表来访问各个分区中的数据，也可以通过在查询时直接指定分区的方法来进行查询。

本章节主要介绍在 VexDB 数据库中如何创建和管理分区表，指导用户正确创建表和管理分区表。

背景信息

VexDB 数据库支持的分区表：Range分区表、List分区表、Hash分区表、system分区表、Interval分区表和垂直分区。

- Range分区表：范围分区根据用户为每个分区建立的分区键值的范围将数据映射到分区。这种分区方式是最常见的，并且分区键值经常采用日期作为分区键，例如将销售数据按照月份进行分区。Range分区支持使用like进行模糊查询（即like的前缀匹配），参见示例2。当前使用like进行模糊查询支持字符串相关类型的分区键，如name、text、bpchar等。
- List分区表：将数据中包含的键值分别存储在不同的分区中，依次将数据映射到每一个分区，分区中包含的键值由创建分区表时指定，键值最多不超过127个。
- Hash分区表：将数据根据内部哈希算法依次映射到每一个分区中，包含的分区个数由创建分区表时指定。

  Hash分区只支持1个分区键，分区键值能用表的普通字段来指定，分区范围的值只支持常量表达式、数值或字符串常量，不支持新增和删除分区。
- system分区表：系统分区可以对没有分区键的表进行分区。创建分区时可以指定分区名称，也可以不指定分区名称，由系统自动分配。
  - system分区表暂不支持二级分区、组合分区。
- Interval分区表：interval分区是range分区的扩展。对于连续数据类型的Range分区，如果插入的新数据值与当前分区均不匹配，自动interval分区可以自动创建分区。详细请参考INTERVAL分区表。
- 垂直分区表：垂直分区功能可将一个数据表按列进行分区。常用于以下场景：
  - 字段冷热分离，将频繁访问字段放在一个分区，不常访问的放在另外分区，从而减少无关的物理I/O。

- 大字段列单独存放，如BLOB列，可将BLOB列单独放在一个分区，其他列放在一个分区，从而减少单独查询其他列的时间。
- 关联关系列放一个分区，例如查询中倾向于一起访问的某些字段，如 a、b、C 列始终一起访问，则将它们放在一个分区，d、e 列总是一起访问，则将它们放在另一个分区。

分区表和普通表相比具有以下优点：

- 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
- 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
- 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。
- 均衡I/O：可以把不同的分区映射到不同的磁盘以平衡I/O，改善整个系统性能。

VexDB 支持分区剪枝功能，分区剪枝是指优化器自动提取出需要扫描的分区，减少扫描的数据块，从而避免全表扫描，提高性能。详细内容请参考分区剪枝。

注意事项

- 普通表若要转成分区表，需要新建分区表，然后把普通表中的数据导入到新建的分区表中。因此在初始设计表时，请根据业务提前规划是否使用分区表。
- 交换分区的普通表，必须与分区表的字段个数、字段类型都完全一致时才可以进行exchange操作，且普通表不能是临时表（支持增删列之后的表进行分区交换）。
- 不支持macaddr、macaddr8作为分区键。
- 垂直分区表使用限制：
  - 不支持二级子分区。
  - 存储方式只支持列存。
  - 不支持本地分区索引。
  - 不支持列存表不支持的特性如trigger、foreign key、alter table、modify partition等。

示例

示例1： 按照以下方式对Range分区表进行操作。

1. 创建表空间。

```
CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace/tablespace_1';
CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace/tablespace_2';
CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace/tablespace_3';
CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace/tablespace_4';
```

2. 创建分区表并插入数据。

```
CREATE TABLE customer_address
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country           character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)
TABLESPACE example1
PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
TABLESPACE example2
)
ENABLE ROW MOVEMENT;

insert into customer_address(ca_address_sk,ca_address_id,ca_street_number) values(12000,'a4','a3'),(20000,'w1','w2'),(50000,'w1','w2');
```

3. 插入数据。

   将表customer_address的数据插入到表web_returns_p2中。

   例如在数据库中创建了一个表customer_address的备份表web_returns_p2，现在需要将表customer_address中的数据插入到表web_returns_p2中，则可以执行如下命令。

   创建备份表web_returns_p2并插入数据。

```
CREATE TABLE web_returns_p2
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country           character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)
TABLESPACE example1
PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;

INSERT INTO web_returns_p2 SELECT * FROM customer_address;
```

4. 重命名分区。

```
ALTER TABLE web_returns_p2 RENAME PARTITION P8 TO P_9;
ALTER TABLE web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
```

5. 查询分区P8。

```
SELECT * FROM web_returns_p2 PARTITION (P8);
```

返回结果如下：

```
ca_address_sk |  ca_address_id   | ca_street_number | ca_street_name | ca_street_type | ca_suite_number | ca_city | ca_county | c
--------------+------------------+------------------+----------------+----------------+-----------------+---------+-----------+--
        50000 | w1               | w2               |                |                |                 |         |           |
(1 row)
```

6. 删除分区表。

```
DROP TABLE web_returns_p2;
DROP TABLE customer_address;
DROP TABLESPACE example1;
DROP TABLESPACE example2;
DROP TABLESPACE example3;
DROP TABLESPACE example4;
```

示例2： 按照以下方式对Hash分区表进行操作。

创建分区表（ Hash分区只支持1个分区键，分区键值能用表的普通字段来指定，分区范围的值只支持常量表达式、数值或字符串常量，不支持新增和删除分区。）

语法格式如下：

```
CREATE TABLE [IF NOT EXISTS] table_name
…
PARTITION BY HASH (columns_name)
hash_partition_desc
…
```

1. 创建分区表。

```
CREATE TABLE t_hash_1
(c1 integer,
c2 date,
c3 text)
PARTITION BY HASH (c1)
(
PARTITION t_hash_p1,
PARTITION t_hash_p2
);
```

2. 插入数据并查询结果。

```
insert into t_hash_1 values(1,'2020-07-29','a');
SELECT * FROM t_hash_1 PARTITION (t_hash_p1);
```

返回结果如下：

```
c1 |         c2          | c3
----+---------------------+----
1 | 2020-07-29 00:00:00 | a
(1 row)
```

3. 更新数据并查询结果。

```
UPDATE t_hash_1 set c1=3 where c1=1;
SELECT * FROM t_hash_1 PARTITION (t_hash_p2);
```

返回结果如下：

```
c1 |         c2          | c3
----+---------------------+----
3 | 2020-07-29 00:00:00 | a
(1 row)
```

4. 删除数据并查询结果。

```
DELETE t_hash_1 where c1=3;
SELECT * FROM t_hash_1;
```

返回结果如下：

```
c1 | c2 | c3
----+----+----
(0 rows)
```

5. 删除分区表。

```
DROP TABLE t_hash_1;
```

示例3： 按照以下方式对List分区表进行操作。

1. 创建分区表。

```
CREATE TABLE t_list
(c1 integer,
c2 date,
c3 text)
PARTITION BY LIST (c2)
(
PARTITION p1 VALUES ('2019-10-12'),
PARTITION p2 VALUES ('2019-10-13'),
PARTITION p3 VALUES ('2019-10-14')
);
```

2. 修改分区表行迁移属性。

```
alter table t_list enable row movement;
```

3. 新增分区。

```
ALTER TABLE t_list ADD PARTITION P4 VALUES ('2019-10-15');
```

4. 删除分区。

```
ALTER TABLE t_list DROP PARTITION p4;
```

5. 插入数据并查询结果。

```
insert into t_list values(1,'2019-10-13','test');
SELECT * FROM t_list PARTITION (p2);
```

返回结果如下：

```
c1 |         c2          |  c3
----+---------------------+------
1 | 2019-10-13 00:00:00 | test
(1 row)
```

6. 查看其它的分区：

```
SELECT * FROM t_list PARTITION (p1);
```

返回结果如下：

```
c1 | c2 | c3
----+----+----
(0 rows)
```

7. 新数据并查询结果。

```
update t_list set c2='2019-10-12' where c1=1;
SELECT * FROM t_list PARTITION (p2);
SELECT * FROM t_list PARTITION (p1);
```

返回结果如下：

```
c1 | c2 | c3
----+----+----
(0 rows)

c1 |         c2          |  c3
----+---------------------+------
1 | 2019-10-12 00:00:00 | test
(1 row)
```

8. 删除分区表。

```
DROP TABLE t_list;
```

示例4： 按照以下方式对system分区表进行操作。

1. 创建分区表（指定分区名称），并向其中插入数据。

```
create table system_par_tab(
c1 integer,
c2 date,
c3 text
)
partition by system
(
partition part_1,
partition part_2,
partition part_3
);
```

2. 创建分区表（系统自动分配分区名称），并向其中插入数据。

```
create table system_par_tab2(
id number,
code varchar2(10),
description varchar2(50)
)
partition by system;
```

3. 查询系统自动生成的分区名称。

   （1）查询系统表pg_partition。

```
\d+ system_par_tab2
```

返回结果如下：

```
Table "public.system_par_tab2"
Column    |    Type     | Modifiers | Storage  | Stats target | Description
------------+-------------+-----------+----------+--------------+-------------
id          | numeric     |           | main     |              |
code        | varchar(10) |           | extended |              |
description | varchar(50) |           | extended |              |
Partition By SYSTEM
Number of partitions: 1 (View pg_partition to check each partition range.)
Has OIDs: no
Options: orientation=row, compression=no, fillfactor=80
```

（2）查询表记录。

```
select relname,parentid,partstrategy from pg_partition where relname='system_par_tab2';
```

返回结果如下：

```
relname     | parentid | partstrategy
----------------+----------+--------------
system_par_tab2 |    16707 | h
system_par_tab2 |    16707 | s
(2 rows)
```

（3）执行。

```
\x

select * from pg_partition where parentid='16707';
```

返回结果如下，其中system_par_tab2 就是系统自动生成的分区名：

```
-[ RECORD 1 ]------+-----------------------------------------------------------------
relname            | system_par_tab2
parttype           | r
parentid           | 16707
rangenum           | 0
intervalnum        | 0
partstrategy       | s
subpartstrategy    | n
relfilenode        | 0
reltablespace      | 0
relpages           | 0
reltuples          | 0
relallvisible      | 0
reltoastrelid      | 0
reltoastidxid      | 0
indextblid         | 0
indisusable        | t
reldeltarelid      | 0
reldeltaidx        | 0
relcudescrelid     | 0
relcudescidx       | 0
relfrozenxid       | 0
intspnum           |
partkey            |
subpartkey         |
intervaltablespace |
interval           |
boundaries         |
transit            |
reloptions         | {orientation=row,compression=no,fillfactor=80,wait_clean_gpi=n}
subparttemplate    |
relfrozenxid64     | 0
relminmxid         | 0
partkeyexpr        |
partitionno        | -1
subpartitionno     |
-[ RECORD 2 ]------+-----------------------------------------------------------------
relname            | system_par_tab2
parttype           | p
parentid           | 16707
rangenum           | 0
intervalnum        | 0
partstrategy       | h
subpartstrategy    | n
relfilenode        | 16711
reltablespace      | 0
relpages           | 0
reltuples          | 0
relallvisible      | 0
reltoastrelid      | 16712
reltoastidxid      | 0
indextblid         | 0
indisusable        | t
reldeltarelid      | 0
reldeltaidx        | 0
relcudescrelid     | 0
```

```
relcudescidx       | 0
relfrozenxid       | 14918
intspnum           |
partkey            |
subpartkey         |
intervaltablespace |
interval           |
boundaries         |
transit            |
reloptions         | {orientation=row,compression=no,fillfactor=80}
subparttemplate    |
relfrozenxid64     | 14918
relminmxid         | 2
partkeyexpr        |
partitionno        | 1
subpartitionno     |
```

4. 关闭列式输出模式向分区表中插入数据。

```
\x off

insert into system_par_tab partition(part_1) values(1,'2022-01-01','p1');
insert into system_par_tab partition(part_2) values(2,'2022-02-01','p2');
insert into system_par_tab partition(part_3) values(3,'2022-03-01','p3');
```

向一个system分区表中插入数据时必须指定其分区名称。

5. 查询分区表。

```
select * from system_par_tab;
```

返回结果如下：

```
c1 |         c2          | c3
---+---------------------+----
1 | 2022-01-01 00:00:00 | p1
2 | 2022-02-01 00:00:00 | p2
3 | 2022-03-01 00:00:00 | p3
```

6. 更新数据。

```
update system_par_tab partition(part_2) set c3='p5' where c1 ='2';
```

7. 新增分区。

```
alter table system_par_tab add partition part_4;
```

8. 删除分区。

```
alter table system_par_tab drop partition part_4;
```

9. 清空分区表。

```
truncate table system_par_tab2;
```

10. 清空指定分区。

```
alter table system_par_tab truncate partition part_1;
```

11. 删除分区表。

```
DROP TABLE system_par_tab2;
DROP TABLE system_par_tab;
```

default 分区

功能描述

针对Range和List分区，支持定义和使用default分区。创建分区表的时候可以指定创建default分区，对于已经创建的分区表，可以新增和删除default分区。

Range类型的分区表，可以通过指定分区范围值为MAXVALUE来包含所有情况的值。

注意事项

- 已存在default分区的LIST分区不能新增分区。
- 删除分区时不能只剩下一个default分区。
- 在创建分区表时，不能只定义一个default分区，并且default分区必须定义在最后。
- 已经存在default分区的分区表，不能新增分区。
- 删除分区时，不能只剩下一个default分区。
- default分区不能进行SPLIT和MERGE操作。

语法格式

- List分区表创建分区表时指定创建default分区，default分区必须定义为最后一个分区，SQL语法如下：

```
CREATE TABLE [IF NOT EXISTS]table_name
......
PARTITION BY LIST })
(
PARTITION partition_name...,
PARTITION partition_name VALUES(DEFAULT)[TABLESPACE tablespace_name]
)
```

- List分区表新增default分区的SQL语法如下：

```
ALTER TABLE table_name ADD PARTITION partition_name VALUES(DEFAULT)
[TABLESPACE tablespace_name];
```

- List分区表删除default分区SQL语法如下：

```
ALTER TABLE partition_name DROP PARTITION partition_name;
```

- Range分区表创建分区表时指定创建default分区，SQL语法如下：

```
CREATE TABLE [IF NOT EXISTS]table_name
......
PARTITION BY RANGE })
(
PARTITION partition_name...,
PARTITION partition_name VALUES LESS THAN (MAXVALUE) [TABLESPACE tablespace_name]
)
```

- Range分区新增default分区的SQL语法如下：

```
ALTER TABLE table_name ADD PARTITION partition_name VALUES LESS THAN (MAXVALUE)
[TABLESPACE tablespace_name];
```

- Range分区表删除default分区SQL语法如下：

```
ALTER TABLE partition_name DROP PARTITION partition_name;
```

示例

1. 创建一个分区表。

```
create table t_partition_list4(col number,name varchar2(20))
partition by list(col)(
partition t_list_p1 values(1,3,5,7,9) ,
partition t_list_p2 values(2,4,6,8,10) );
```

2. 添加一个默认分区并插入数据。

```
alter table t_partition_list4 add partition t_list_default values(default);
insert into t_partition_list4 values(1,'t_list_p1');
insert into t_partition_list4 values(2,'t_list_p2');
insert into t_partition_list4 values(31,'t_list_default');
```

3. 查询分区表中的数据。

   - 查询分区t_list_p1的数据。

     ```
     select * from t_partition_list4 partition(t_list_p1);
     ```

     返回结果如下：

     ```
     col |   name
     -----+-----------
     1 | t_list_p1
     (1 row)
     ```

   - 查询分区t_list_p2的数据。

     ```
     select * from t_partition_list4 partition(t_list_p2);
     ```

     返回结果如下：

     ```
     col |   name
     -----+-----------
     2 | t_list_p2
     (1 row)
     ```

   - 查询分区t_list_default的数据。

```
select * from t_partition_list4 partition(t_list_default);
```

返回结果如下：

```
col |      name
-----+----------------
31 | t_list_default
(1 row)
```

4. 清理环境。

```
drop table t_partition_list4;
```

interval分区表

功能描述

interval分区是range分区的扩展。

在没有自动interval分区特性之前，在创建范围（Range）类型分区表时，用户通常会定义一个maxvalue分区，将没有落在当前关注范围内的数据，都放到该分区中，以免发生插入的元组的分区键值不能映射到任何分区的错误。然而当业务场景发生变化时，如果没有及时根据数据增长创建新的分区，就可能会导致分区表的数据发生倾斜，大多数数据都放到了这个未进行细分的maxvlaue分区中，这就违背了用户对表进行分区的初衷：用户希望各个分区的数据量均衡，这样才能加快查询。

自动interval分区可以改善这个问题，对于连续数据类型的Range分区，如果插入的新数据值与当前分区均不匹配，自动interval分区可以自动创建分区。

注意事项

列存表不支持间隔分区。

语法格式

```
CREATE TABLE table_name
(
  ...
)
PARTITION BY RANGE(column1)
INTERVAL ('interval_expr') [ STORE IN (tablespace_name [, … ] ) ]
(
  PARTITION partition_name1 VALUESLESS THAN ( { partition_value | MAXVALUE } )
  PARTITION partition_name2 VALUES LESS THAN ( { partition_value | MAXVALUE } )
);
```

参数说明

- PARTITION BY RANGE(column1)

  指定一个range分区的列。

- INTERVAL (‘interval_expr’) [STORE IN (tablespace_name [, ...])]

  间隔分区定义信息。

  - interval_expr：自动创建分区的间隔，例如：1 day、1 month。
  - STORE IN (tablespace_name [, ...])：指定存放自动创建分区的表空间列表，如果有指定，则自动创建的分区从表空间列表中循环选择使用，否则使用分区表默认的表空间。

- PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE })

  指定各分区的信息。partition_name为范围分区的名称。partition_value为范围分区的上边界，取值依赖于partition_key的类型。MAXVALUE表示分区的上边界，它通常用于设置最后一个范围分区的上边界。

  - 每个分区都需要指定一个上边界。
  - 分区上边界的类型应当和分区键的类型一致。
  - 分区列表是按照分区上边界升序排列的，值较小的分区位于值较大的分区之前。

示例

1. 创建interval分区表。

```
CREATE TABLE interval_normal_exchange (logdate date not null)
PARTITION BY RANGE(logdate)
INTERVAL('1 month')
(
PARTITION interval_normal_exchange_p1 VALUES LESS THAN('2020-03-01'),
PARTITION interval_normal_exchange_p2 VALUES LESS THAN('2020-04-01'),
PARTITION interval_normal_exchange_p3 VALUES LESS THAN('2020-05-01')
);
```

2. 执行以下语句查询分区情况。

```
SELECT relname,parttype,partstrategy,boundaries FROM pg_partition
WHERE parentid = (SELECT oid FROM pg_class WHERE relname = 'interval_normal_exchange')
ORDER BY relname;
```

返回结果如下：

```
relname           | parttype | partstrategy |  boundaries
--------------------------+----------+--------------+--------------
interval_normal_exchange    | r        | i            |
interval_normal_exchange_p1 | p        | r            | {2020-03-01}
interval_normal_exchange_p2 | p        | r            | {2020-04-01}
interval_normal_exchange_p3 | p        | r            | {2020-05-01}
(4 rows)
```

3. 清理环境。

```
drop table interval_normal_exchange;
```

二级分区

功能描述

二级分区功能，即在原有的range分区、list分区、hash分区、interval分区的基础上再次进行分区。

二级分区可以对表中的每个分区再次进行分区。分区类型有range、list、hash三种，一级与二级分区的分区类型可以任意组合。二级分区支持使用CREATE/ALTER/SELECT语法，用于二级分区的创建与增删改查。二级分区的相关信息可在系统表PG_PARTITION中获取。

注意事项

interval分区不能作为二级分区。

语法格式

创建、修改和删除二级分区的语法分别是：CREATE TABLE SUBPARTITION、ALTER TABLE SUBPARTITION。用户也可以使用如下命令在vsql客户端中查询相关SQL语法的使用帮助信息。

```
\h create table subpartition
\h alter table subpartition
```

二级分区的增删改查

对二级分区表插入数据的语法与普通表的语法没有任何差别。当向二级分区表插入数据时，会根据一级分区和二级分区规则，把数据分布到匹配的二级分区中进行存储。

以下列举了部分与二级分区相关的常用SQL句式。

- 新增一级与二级分区：

```
ALTER TABLE table_name ADD partition_desc [ ( subpartition_desc [, ... ] ) ]
```

- 删除指定一级分区包括属于它的所有二级分区：

```
ALTER TABLE table_name DROP PARTITION { partition_name | { FOR (value [ , … ] )}
```

- 为指定一级分区新增二级分区：

```
ALTER TABLE table_name MODIFY PARTITION partition_name ADD subpartition_desc
```

- 删除指定二级分区：

```
ALTER TABLE table_name DROP SUBPARTITION { subpartition_name | { FOR (value [ , … ])}
```

- 删除数据：

```
DELETE FROM table_name [ PARTITION { ( partition_name ) | { FOR ( value [ , ... ] ) } } ] [ WHERE ...]
DELETE FROM table_name [ SUBPARTITION { ( partition_name ) | { FOR ( value [ , ... ] ) } } ] [ WHERE ...]
```

- 更新数据：

```
UPDATE table_name [ PARTITION { ( partition_name ) | { FOR ( value [ , ... ] ) } } ] SET [ WHERE ...]
UPDATE table_name [ SUBPARTITION { ( partition_name ) | { FOR ( value [ , ... ] ) } } ] SET [ WHERE ...]
```

- 在指定二级分区查找数据：

```
SELECT ... FROM table_name SUBPARTITION (subpartition_name)
```

- 单独查询某个分区或子分区数据：

```
SELECT ... FROM table_name PARTITION { ( partition_name ) | FOR ( value [ , ... ] ) }
SELECT ... FROM table_name SUBPARTITION { ( subpartition_name ) | {FOR ( value [, … ] ) }
```

- 当查询二级分区表的数据时（不能指定分区或子分区），如果查询条件中包含有分区键的条件，则在生成执行计划时会根据分区键条件过滤掉不符合条件的分区或子分区：

```
EXPLAIN SELECT ... FROM table_name WHERE ...
```

参数说明

- table_name

表名。

- partition_name

  分区表的名称。

- subpartition_name

  二级分区表的名称。

- SUBPARTITION BY [RANGE | LIST | HASH]

  根据column_name指定的字段进行二级分区，分区类型可以是RANGE、LIST、 HASH其一。

- SUBPARTITION TEMPLATE (subpartition_desc [, ...])

  常规子分区模板定义语法，适用于Range/List/Hash子分区，当分区定义中没有指定子分区的定义时会根据子分区模板自动生成二级分区。

  若没有提供子分区模板，同时partition_desc也不指定subpartition_desc，则将创建一个默认子分区。

- hash_subpartition_by_quantity

  Hash分区持有的子分区模板定义语法（也可用与子分区定义语法），指定创建N个Hash子分区，同时可选指定Hash子分区的tablespace名列表。

示例

示例1： 二级分区的创建，修改和删除。

1. 创建一个分区表并插入数据。

```
create table t_part_list_range
( id number not null,
partition_key int,
subpartition_key int,
col2 varchar2(10)
)
partition by list(partition_key)
subpartition by range(subpartition_key)
(
partition t_partition_01 values (100)
(subpartition sub_1_1 values less than (10),
subpartition sub_1_2 values less than (20)
),
partition t_partition_02 values (200)
(subpartition sub_2_1 values less than (10),
subpartition sub_2_2 values less than (20)
)
);
insert into t_part_list_range values(1,100,5,'sub_1_1');
insert into t_part_list_range values(2,100,15,'sub_1_2');
insert into t_part_list_range values(3,200,5,'sub_2_1');
insert into t_part_list_range values(4,200,15,'sub_2_2');
insert into t_part_list_range values(5,200,16,'sub_2_2');
select * from t_part_list_range subpartition for (100,5);
```

2. 新增一级与二级分区。

```
alter table t_part_list_range add partition t_partition_03 values (300)
( subpartition sub_3_1 values less than (10),
subpartition sub_3_2 values less than (20)
);
```

3. 删除指定一级分区包括属于它的所有二级分区。

```
alter table t_part_list_range drop partition t_partition_02;
```

4. 为指定一级分区新增二级分区。

```
alter table t_part_list_range modify partition t_partition_01 add subpartition sub_1_3 values less than (30);
```

5. 删除指定二级分区。

```
alter table t_part_list_range drop subpartition sub_1_3;
```

6. 清理环境。

```
drop table t_part_list_range;
```

示例2： 一级分区为interval分区，二级分区为list分区，分区键类型为字符类型；创建并查看分区结果。

1. 创建测试表，包含一级分区和二级分区。

```
CREATE TABLE t_subpartition_interval_list(
partition_key date,
subpartition_key varchar(20),
test varchar(20)
)partition by range(partition_key)
interval('12 month')
subpartition by list(subpartition_key)
(partition partition_p1 VALUES LESS THAN ('2019-01-01')
(subpartition sub_1_1 values('test1'),
subpartition sub_1_2 values('test2'),
subpartition sub_1_3 values (default)
),
partition partition_p2 VALUES LESS THAN ('2021-01-01')
(subpartition sub_2_1 values('test1'),
subpartition sub_2_2 values('test2'),
subpartition sub_2_3 values (default)
)
);
```

2. 插入测试数据。

```
INSERT INTO t_subpartition_interval_list VALUES('2018-09-02','test','test1');
INSERT INTO t_subpartition_interval_list VALUES('2020-12-12','test2','test2');
INSERT INTO t_subpartition_interval_list VALUES('2022-02-02','interval','test3');
```

3. 查询已有分区对应数据。

```
select * from t_subpartition_interval_list subpartition(sub_1_3);
select * from t_subpartition_interval_list subpartition(sub_2_2);
```

返回结果如下：

```
partition_key    | subpartition_key | test
---------------------+------------------+-------
2018-09-02 00:00:00 | test             | test1
(1 row)

    partition_key    | subpartition_key | test
---------------------+------------------+-------
2020-12-12 00:00:00 | test2            | test2
(1 row)
```

4. 查询表新增分区名。

```
select relname,parttype,partstrategy,boundaries
from pg_partition
where parentid = (select oid from pg_class where relname = 't_subpartition_interval_list')
union all
select relname,parttype,partstrategy,boundaries from pg_partition
where parentid in (
select oid from pg_partition
where parentid = (
select oid from pg_class where relname ='t_subpartition_interval_list'
)
order by relname)
order by relname;
```

返回结果如下：

```
relname            | parttype | partstrategy |       boundaries
-------------------------------+----------+--------------+-------------------------
partition_p1                | p        | r            | {2019-01-01}
partition_p2                | p        | r            | {2021-01-01}
sub_1_1                     | s        | l            | {test1}
sub_1_2                     | s        | l            | {test2}
sub_1_3                     | s        | l            |
sub_2_1                     | s        | l            | {test1}
sub_2_2                     | s        | l            | {test2}
sub_2_3                     | s        | l            |
sys_p1                      | p        | i            | {"2023-01-01 00:00:00"}
sys_p1_subpartdefault1      | s        | l            |
t_subpartition_interval_list | r        | i            |
(11 rows)
```

5. 查询新增分区对应数据。

```
select * from t_subpartition_interval_list subpartition(sys_p1_subpartdefault1);
```

返回结果如下：

```
partition_key    | subpartition_key | test
--------------------+------------------+-------
2022-02-02 00:00:00 | interval         | test3
(1 row)
```

6. 清理环境。

```
drop table t_subpartition_interval_list;
```

分区剪枝

分区剪枝是指优化器自动提取出需要扫描的分区，减少扫描的数据块，从而避免全表扫描，提高性能。VexDB 支持静态分区剪枝和动态分区剪枝。

静态分区剪枝

功能描述

静态分区剪枝是一种优化技术，用于提高查询性能。当在对分区表执行SQL查询时，如果查询条件中包含分区键，优化器会在编译阶段就能够判断出哪些分区是不必要的，并且可以直接排除这些分区的扫描。

表1 支持的分区类型及表达式类型

分区类型	表达式	分区等级
范围(Range)分区	<,<=,=,>,>=,like,any,all	一级、二级分区
列表(List)分区	<,<=,>,>=,<>(不等于)	一级、二级分区
哈希(Hash)分区	=	一级、二级分区
间隔(Interval)分区	<,<=,=,>,>=,any,all	一级、二级分区

注意事项

- 多个分区键的List分区表，操作符<>存在剪枝不完全的情况，不建议使用。
- List分区表查询中如果有多个<>操作符的查询条件，存在剪枝不完全的情况，不建议使用。
- 二级分区表分区键一致时存在剪枝不完全的情况，不建议使用。
- 表达式like仅支持字符类型。

示例

示例1: 一级分区表静态剪枝。

1. 创建测试表并插入数据。

```
create table par1_1187644(id int,a1 text,a2 date,a3 varchar(30))
partition by list (id)
(
partition p1 values(100),
partition p2 values(200),
partition p3 values(300),
partition p4 values(400),
partition p5 values(null));

insert into par1_1187644 values(100,chr(65 + (generate_series(1,100)-1)%25),generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1187644 values(200,chr(65 + (generate_series(1,100)-1)%25),generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1187644 values(300,chr(65 + (generate_series(1,100)-1)%25),generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1187644 values(400,chr(65 + (generate_series(1,100)-1)%25),generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1187644 values(null,chr(65 + (generate_series(1,100)-1)%25),generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
```

2. 指定条件查询数据。

```
select * from par1_1187644 where id>212 limit 5;
```

返回结果如下：

```
id  | a1 |         a2          | a3
-----+----+---------------------+----
300 | A  | 2022-01-01 00:00:00 | A
300 | B  | 2022-01-02 00:00:00 | B
300 | C  | 2022-01-03 00:00:00 | C
300 | D  | 2022-01-04 00:00:00 | D
300 | E  | 2022-01-05 00:00:00 | E
(5 rows)
```

3. 查看执行计划验证剪枝效果。

```
explain (costs off) select * from par1_1187644 where id>212 limit 5;
```

返回结果如下，可以看出只扫描了部分分区：

```
               QUERY PLAN
-------------------------------------------------
Limit
->  Partition Iterator
        Iterations: 2
        ->  Partitioned Seq Scan on par1_1187644
            Filter: (id > 212)
            Selected Partitions:  3..4
(6 rows)
```

4. 清理环境。

```
drop table par1_1187644;
```

示例2: 二级分区表静态剪枝。

1. 创建二级分区表。

```
create table par1_1187650(id int,a1 text,a2 date,a3 varchar(30))
partition by range (id)
subpartition by list(a1)
(
partition p1 values less than(100)(
subpartition p1_1 values('g'),
subpartition p1_2 values('n'),
subpartition p1_3 values('z'),
subpartition p1_4 values(null)),
partition p2 values less than(200)(
subpartition p2_1 values('g'),
subpartition p2_2 values('n'),
subpartition p2_3 values('z'),
subpartition p2_4 values(null)),
partition p3 values less than(300)(
subpartition p3_1 values('g'),
subpartition p3_2 values('n'),
subpartition p3_3 values('z'),
subpartition p3_4 values(null)),
partition p4 values less than(401)(
subpartition p4_1 values('g'),
subpartition p4_2 values('n'),
subpartition p4_3 values('z'),
subpartition p4_4 values(null)));
```

2. 插入测试数据。

```
insert into par1_1187650 values(generate_series(1,100),'g',generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1187650 values(generate_series(101,200),'n',generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1187650 values(generate_series(201,300),'z',generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1187650 values(generate_series(301,400),null,generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
```

3. 指定条件查询数据。

```
select * from par1_1187650 where id>212 and a1>'d' limit 5;
```

返回结果如下：

```
id  | a1 |         a2          | a3
-----+----+---------------------+----
213 | z  | 2022-01-13 00:00:00 | M
214 | z  | 2022-01-14 00:00:00 | N
215 | z  | 2022-01-15 00:00:00 | O
216 | z  | 2022-01-16 00:00:00 | P
217 | z  | 2022-01-17 00:00:00 | Q
(5 rows)
```

4. 查看执行计划验证剪枝效果。

```
explain (costs off) select * from par1_1187650 where id>212 and a1>'d' limit 5;
```

返回结果如下，可以看出只扫描了部分一级、二级分区，其中Selected Subpartitions字段的值代表二级分区的数量。

```
                      QUERY PLAN
------------------------------------------------------------
Limit
->  Partition Iterator
        Iterations: 2, Sub Iterations: 6
        ->  Partitioned Seq Scan on par1_1187650
            Filter: ((id > 212) AND (a1 > 'd'::text))
            Selected Partitions:  3..4
            Selected Subpartitions:  3:3, 4:3
(7 rows)
```

5. 清理环境。

```
drop table par1_1187650;
```

动态分区剪枝

功能描述

动态分区剪枝是与静态分区剪枝相对的一种优化技术，同样用于提高查询性能。在处理分区表时，当SQL查询条件中包含变量或者表达式，它们的值只有在运行时才能确定的情况下，数据库引擎能够在执行阶段根据实际的查询参数值来决定应当访问哪些分区，并排除不必要的分区扫描。

VexDB 支持对Range、List、Hash、Interval分区表进行动态分区剪枝，支持的场景如下：

- 支持PREPARE/EXECUTE语法对一级、二级分区表动态分区剪枝。需注意查询条件需包含分区键，且条件值为参数。
- 支持InitPlan场景的一级、二级分区表动态分区剪枝。需注意分区键的条件为子连接，且为非相关子连接。
- 支持Nested Loop且连接条件可下推至分区表场景的一级、二级分区表动态分区剪枝，查询走nestloop计划，且包含分区键的条件，其中分区键存在本地分区索引。

前置条件

使用动态分区剪枝需设置GUC参数enable_runtime_prune为on，该参数默认值为on，参数类型为USERSET。

示例

1. 创建测试表并插入测试数据。

```
create table par1_1188069(id int,a1 text,a2 date,a3 varchar(30))
partition by range (id)
subpartition by list(a1)
(
partition p1 values less than(100)(
subpartition p1_1 values('d'),
subpartition p1_2 values('k'),
subpartition p1_3 values('q'),
subpartition p1_4 values(null)),
partition p2 values less than(200)(
subpartition p2_1 values('d'),
subpartition p2_2 values('k'),
subpartition p2_3 values('q'),
subpartition p2_4 values(null)),
partition p3 values less than(300)(
subpartition p3_1 values('d'),
subpartition p3_2 values('k'),
subpartition p3_3 values('q'),
subpartition p3_4 values(null)),
partition p4 values less than(401)(
subpartition p4_1 values('d'),
subpartition p4_2 values('k'),
subpartition p4_3 values('q'),
subpartition p4_4 values(null)));

create index on par1_1188069(id)local;

insert into par1_1188069 values(generate_series(1,100),'d',generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1188069 values(generate_series(101,200),'k',generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1188069 values(generate_series(201,300),'q',generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
insert into par1_1188069 values(generate_series(301,400),null,generate_series(DATE '2022-01-01', DATE '2022-4-10', '1 day'),chr(65 + (generate_series(1,100)-1)%25));
```

2. 确认GUC参数enable_runtime_prune的值是否为on。

```
show enable_runtime_prune;
```

返回结果如下：

```
enable_runtime_prune
----------------------
on
(1 row)
```

3. 使用PREPARE语法创建预备查询语句。

```
prepare i1_1188069(int,int) as select * from par1_1188069 where id>$1 and id<$2 limit 3;
```

4. 查看执行计划。

```
explain (analyze,costs off) execute i1_1188069(130,230);
```

返回结果如下，可以看出只扫描了部分分区：

```
                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
Limit (actual time=0.107..0.109 rows=3 loops=1)
->  Partition Iterator (actual time=0.106..0.108 rows=3 loops=1)
        Iterations: PART
        ->  Partitioned Index Scan using par1_1188069_id_idx on par1_1188069 (actual time=0.018..0.019 rows=3 loops=2)
            Index Cond: ((id > $1) AND (id < $2))
            Selected Partitions:  PART
            Selected Subpartitions:  PART
Total runtime: 0.288 ms
(8 rows)
```

5. 关闭GUC参数enable_runtime_prune对比查询效果。

```
set enable_runtime_prune off;
show enable_runtime_prune;
```

返回结果如下：

```
enable_runtime_prune
----------------------
off
(1 row)
```

6. 使用PREPARE语法创建预备查询语句。

```
prepare i1_1188070(int,int) as select * from par1_1188069 where id>$1 and id<$2 limit 3;
```

7. 查看执行计划。

```
explain (analyze,costs off) execute i1_1188070(130,230);
```

返回结果如下，可以看出执行了全表扫描。

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
Limit (actual time=0.045..0.047 rows=3 loops=1)
->  Partition Iterator (actual time=0.044..0.045 rows=3 loops=1)
        Iterations: 4, Sub Iterations: 16
        ->  Partitioned Index Scan using par1_1188069_id_idx on par1_1188069 (actual time=0.025..0.026 rows=3 loops=6)
            Index Cond: ((id > $1) AND (id < $2))
            Selected Partitions:  1..4
            Selected Subpartitions:  ALL
Total runtime: 0.200 ms
(8 rows)
```

8. 清理环境。

```
drop table par1_1188069;
```

创建和管理索引

背景信息

索引可以提高数据访问速度，但会增加插入、更新和删除操作的处理时间。因此，在创建索引前，需要仔细分析应用程序的业务处理、数据使用情况、以及经常用于查询或排序的字段，以确定是否建立索引及索引应建立在哪些字段上。

在创建索引时，应考虑在哪些列上创建索引，可参考以下原则：

- 在经常需要搜索查询的列上创建索引，可以加快搜索的速度。
- 在作为主键的列上创建索引，强制该列的唯一性和组织表中数据的排列结构。
- 在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的。
- 在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间。
- 在经常使用 WHERE 子句的列上创建索引，加快条件的判断速度。
- 为经常出现在关键字 ORDER BY、GROUP BY、DISTINCT 后面的字段建立索引。

注意事项

- 索引创建成功后，系统会自动判断何时引用索引。当系统认为使用索引比顺序扫描更快时，就会使用索引。
- 索引创建成功后，必须和表保持同步以保证能够准确地找到新数据，这样就增加了数据操作的负荷。因此请定期删除无用的索引。
- 分区表索引分为 LOCAL 索引与 GLOBAL 索引，一个 LOCAL 索引对应一个具体分区，而 GLOBAL 索引则对应整个分区表。

- 表字段大于等于 2074 字节不支持创建 BTree 索引，建议使用 HASH 或 GIN 索引。
- 在开启逻辑复制的场景下，如需创建包含系统列的主键索引，必须将该表的REPLICA IDENTITY 属性设置为 FULL 或使用 USING INDEX 指定不包含系统列的、唯一的、非局部的、不可延迟的、仅包括标记为 NOT NULL 的列的索引。

功能描述

不同的索引方式与索引类型适用于不同的场景。

索引方式

VexDB 支持的索引方式如下表所示：

表1 索引方式

索引方式	描述
BTree	使用一种类似于 B+ 树的结构来存储数据的键值，通过这种结构能够快速的查找索引。适合比较查询以及查询范围。
GiST	适用于几何和地理等多维数据类型和集合数据类型。支持的数据类型为： box、point、polygon、circle、tsvector、tsquery、range。
Hash	使用 Hash 函数对索引的关键字进行散列。仅支持等值比较。适用于索引列较长的场景使用。
Gin	倒排索引，可以处理包含多个键的值。支持的数据类型为：array、json、tsvector、tsquery。
Psort	是针对列存表进行局部排序的索引。
UBTree	多版本 BTree 索引，适用于 Ustore 表。

索引类型

VexDB 支持的索引类型如下表所示：

表2 索引类型

索引类型	描述
唯一索引	可用于约束索引属性值的唯一性，或者属性组合值的唯一性。如果一个表声明了唯一约束或者主键，则 VexDB 自动在组成主键或唯一约束的字段上创建唯一索引（可能是多字段索引），以实现这些约束。目前，VexDB 只有 BTree 可以创建唯一索引。
多字段索引	一个索引可以定义在表中的多个属性上，在一些场合也被称为联合索引。目前，VexDB 中的 BTree 支持多字段索引，且最多可在32个字段上创建索引（全局分区索引最多支持31个字段）。
部分索引	建立在一个表的子集上的索引，这种索引方式只包含满足条件表达式的元组。
表达式索引	索引建立在一个函数或者从表中一个或多个属性计算出来的表达式上。表达式索引只有在查询时使用与创建时相同的表达式才会起作用。
前缀索引	取指定字段数据的前缀作为索引键值。对于 MySQL 兼容模式，支持 column(N) 形式的前缀索引。对于 Oracle 兼容模式，支持 substr(col,1,offset) 形式的前缀索引。

创建索引

使用 CREATE INDEX 语句创建索引。

语法格式

创建索引的基础语法如下：

```
CREATE INDEX index_name ON table_name [ USING method] (column_name[(N)] [,...])
```

参数说明

- index_name

  要创建的索引名称，默认与 table_name 所声明的表名创建在相同的模式中。

- table_name

  需要为其创建索引的表的名称，可以用模式修饰。

- method

  指定创建索引的方法。默认会将索引创建为 B-Tree 索引。

  支持的取值为：

  - btree：创建 BTree 索引。
  - hash：创建 Hash 索引。
  - gin：创建 Gin 索引。

- gist：创建 Gist 索引。
- Psort：创建 Psort 索引。
- ubtree：创建 UBtree 索引。

- column_name(N)

  表中需要创建索引的列的名称（字段名）。

  指定多列时，创建为多字段索引。

  指定 (N) 时，将创建前缀索引，其中 N 为前缀长度。

示例

示例1： 基于普通表创建索引

1. 创建普通表。

```
CREATE TABLE customer_address
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    integer                            ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country           character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
);
```

2. 创建普通索引。

   如果对于 customer_address 表，需要经常进行以下查询。

```
SELECT ca_address_sk FROM customer_address WHERE ca_address_sk=14888;
```

   通常，数据库系统需要逐行扫描整个 customer_address 表以寻找所有匹配的元组。如果表 customer_address 的规模很大，但满足 WHERE 条件的只有少数几个（可能是零个或一个），则这种顺序扫描的性能就比较差。如果让数据库系统在 ca_address_sk 属性上维护一个索引，用于快速定位匹配的元组，则数据库系统只需要在搜索树上查找少数的几层就可以找到匹配的元组，这将会大大提高数据查询的性能。同样，在数据库中进行更新和删除操作时，索引也可以提升这些操作的性能。

   使用如下命令创建索引：

```
CREATE INDEX index_wr_returned_date_sk ON customer_address (ca_address_sk);
```

3. 创建多字段索引。

   假如用户需要经常查询表 customer_address 中 ca_address_sk 是 5050，且ca_street_number 小于 1000 的记录，使用以下命令进行查询。

```
SELECT ca_address_sk,ca_address_id FROM customer_address WHERE ca_address_sk = 5050 AND ca_street_number < 1000;
```

   使用以下命令在字段 ca_address_sk 和 ca_street_number 上定义一个多字段索引。

```
CREATE INDEX more_column_index ON customer_address(ca_address_sk ,ca_street_number );
```

4. 创建部分索引。

```
CREATE INDEX part_index ON customer_address(ca_address_sk) WHERE ca_address_sk = 5050;
```

5. 创建表达式索引。

   假如经常需要查询 ca_street_number 小于 1000 的信息，执行如下命令进行查询。

```
SELECT * FROM customer_address WHERE trunc(ca_street_number) < 1000;
```

   可以为上面的查询创建表达式索引：

```
CREATE INDEX para_index ON customer_address (trunc(ca_street_number));
```

6. 清理环境。

```
DROP TABLE customer_address CASCADE;
```

示例2： 基于分区表创建索引

1. 创建分区表。

```
CREATE TABLE web_returns_p2
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country           character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)
PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

2. 创建分区表 LOCAL 索引 tpcds_web_returns_p2_index1，不指定索引分区的名称。

```
CREATE INDEX tpcds_web_returns_p2_index1 ON web_returns_p2 (ca_address_id) LOCAL;
```

3. 创建分区表 LOCAL 索引tpcds_web_returns_p2_index2，并指定索引分区的名称。

```
CREATE INDEX tpcds_web_returns_p2_index2 ON web_returns_p2 (ca_address_sk) LOCAL
(
    PARTITION web_returns_p2_P1_index,
    PARTITION web_returns_p2_P2_index,
    PARTITION web_returns_p2_P3_index,
    PARTITION web_returns_p2_P4_index,
    PARTITION web_returns_p2_P5_index,
    PARTITION web_returns_p2_P6_index,
    PARTITION web_returns_p2_P7_index,
    PARTITION web_returns_p2_P8_index
);
```

4. 创建分区表 GLOBAL 索引 tpcds_web_returns_p2_global_index。

```
CREATE INDEX tpcds_web_returns_p2_global_index ON web_returns_p2 (ca_street_number) GLOBAL;
```

5. 清理测试环境。

```
DROP TABLE web_returns_p2  CASCADE;
```

修改索引

使用 ALTER INDEX 语句修改索引。

语法格式

```
ALTER INDEX index_name SET tablespace_name;
ALTER INDEX index_name MOVE PARTITION partition_index_name TABLESPACE tablespace_name;
ALTER INDEX index_name UNUSABLE;
ALTER INDEX index_name REBUILD;
```

参数说明

- index_name

  索引名称。

- partition_index_name

  分区上的局部索引名称。

- table_spacename

  表空间名称。

- UNUSABLE

  将索引设置为不可用，不可用的索引不会参与查询优化。

- REBUILD

  重建索引。对应不可用的索引，进行重建后将变为可用状态。

  也可以使用 REINDEX 语句重建索引，请参阅 REIDNEX。

示例

示例3： 修改索引

1. 创建分区表。

```
CREATE TABLE web_returns_p2
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country           character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)
PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

2. 创建表空间。

```
CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace/tablespace_1';
CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace/tablespace_2';
```

3. 修改索引分区_web_returns_p2_P2_index 的表空间为 example1。

```
ALTER INDEX tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P2_index TABLESPACE example1;
```

4. 修改索引分区_web_returns_p2_P3_index 的表空间为 example2。

```
ALTER INDEX tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P3_index TABLESPACE example2;
```

5. 将索引 tpcds_web_returns_p2_index2 设置为不可用。

```
ALTER INDEX tpcds_web_returns_p2_index2 UNUSABLE;
```

6. 重建 tpcds_web_returns_p2_index2 索引。

```
ALTER INDEX tpcds_web_returns_p2_index2 REBUILD;
```

7. 清理测试环境。

```
DROP TABLE web_returns_p2 CASCADE;
DROP TABLESPACE example1;
DROP TABLESPACE example2;
```

查询索引

查询方式

PG_CLASS 系统表记录了所有的数据库对象，其中 relkind 列为 i 或 I 分别表示该对象为索引或分区表上的全局索引。此外，还可以通过 vsql 的元命令 `\di` 查看索引信息。

示例

示例4： 查询索引

1. 创建分区表。

```
CREATE TABLE web_returns_p2
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country           character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)
PARTITION BY RANGE (ca_address_sk)
(
        PARTITION P1 VALUES LESS THAN(5000),
        PARTITION P2 VALUES LESS THAN(10000),
        PARTITION P3 VALUES LESS THAN(15000),
        PARTITION P4 VALUES LESS THAN(20000),
        PARTITION P5 VALUES LESS THAN(25000),
        PARTITION P6 VALUES LESS THAN(30000),
        PARTITION P7 VALUES LESS THAN(40000),
        PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

2. 创建表空间。

```
CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace/tablespace_1';
CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace/tablespace_2';
```

3. 执行如下命令查询系统和用户定义的所有索引。

```
SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i' or RELKIND='I';
```

4. 使用 vsql 元命令查询指定索引的信息。

```
\di+ tpcds_web_returns_p2_index2
```

返回结果如下：

```
                                      List of relations
 Schema |            Name             | Type  |  Owner   |     Table      | Size  | Storage | Description
--------+-----------------------------+-------+----------+----------------+-------+---------+-------------
 public | tpcds_web_returns_p2_index2 | index | vexdb    | web_returns_p2 | 64 kB |         |
(1 row)
```

5. 清理环境。

```
DROP TABLE web_returns_p2 CASCADE;
DROP TABLESPACE example1;
DROP TABLESPACE example2;
```

删除索引

删除索引。

```
DROP INDEX tpcds_web_returns_p2_index1;
DROP INDEX tpcds_web_returns_p2_index2;
```

当结果显示为如下信息，则表示删除成功：

```
DROP INDEX
```

创建和管理视图

背景信息

当用户需要组合数据库中一张或多张表的某些字段的数据，而又不想每次键入这些查询时，就可以定义一个视图，以解决这个问题。

视图与基本表不同，不是物理上实际存在的，是一个虚表。数据库中仅存放视图的定义，而不存放视图对应的数据，这些数据仍存放在原来的基本表中。若基本表中的数据发生变化，从视图中查询出的数据也随之改变。从这个意义上讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据及变化。视图每次被引用的时候都会运行一次。

功能描述

VexDB 数据库对视图的管理包含如下功能：

- 创建视图。
- 查询视图。
- 对简单视图进行插入、更新和删除数据的操作。
- 支持视图基表更新时自动刷新视图，在查询一次视图后刷新视图定义。支持场景如下：
  - 基表增加字符串类型列的宽度。
  - 修改基表列类型。
  - 在基表删除列后重新添加被删除的列。

> **说明**
>
> - 当视图中SELECT的目标不是基表而是表达式时，如果基表字段类型修改导致表达式结果类型变化，视图不会自动刷新。
> - 基表删除字段后导出数据，导入数据后在基表添加被删除字段，此场景视图无法自动刷新。
> - 使用SELECT * 创建的视图，在基表变更后，使用命令查看视图的定义时会出错。
> - 使用SELECT * 创建的视图不支持自动刷新。使用SELECT * 创建的视图在基表变更后，使用SELECT查询视图将自动重新编译，重新编译后的视图targetlist会变化。不建议使用SELECT * 创建视图。

- 支持对视图依赖的对象进行DDL操作，除上述视图自动刷新场景外，对象修改后视图变为Invalid状态，需重新编译或重新创建视图。支持的对象如下：
  - 表、分区表
  - 视图
  - 函数

- 类型
- 操作符

注意事项

支持将无效视图导出为FORCE VIEW，因FORCE VIEW的局限性，可能导致复杂的无效视图导入失败。

示例

示例1： 创建视图

1. 创建测试表，并向表中插入数据。

```
CREATE TABLE t_normal2(id INT,
                    col_varying CHARACTER VARYING(30),
                    col_varchar VARCHAR(30),
                    col_character CHARACTER(30),
                    col_char CHAR(30),
                    col_text TEXT,
                    col_name NAME);
INSERT INTO t_normal2 (id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(1,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
```

当CREATE VIEW中存在OR REPLACE时，表示若以前存在该视图就进行替换，但新查询不能改变原查询的列定义，包括顺序、列名、数据类型、类型精度等，只可在列表末尾添加其他的列。

2. 创建视图。

```
CREATE OR REPLACE VIEW v_normal2 AS SELECT * FROM t_normal2;
```

3. 清理测试数据。

```
DROP TABLE t_normal2 CASCADE;
```

示例2： 查询视图

1. 创建视图。

```
CREATE TABLE t_normal2(id INT,
                    col_varying CHARACTER VARYING(30),
                    col_varchar VARCHAR(30),
                    col_character CHARACTER(30),
                    col_char CHAR(30),
                    col_text TEXT,
                    col_name NAME);
INSERT INTO t_normal2 (id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(1,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
CREATE OR REPLACE VIEW v_normal2 AS SELECT * FROM t_normal2;
```

2. 查询视图。

```
SELECT * FROM t_normal2;
```

返回结果如下：

```
 id | col_varying | col_varchar |        col_character         |           col_char            |  col_text  | col_name
----+-------------+-------------+------------------------------+-------------------------------+------------+----------
  1 | 测试abc123  | 测试abc123   | 测试abc123                   | 测试abc123                    | 测试abc123 | NAME
(1 row)
```

3. 查看视图定义。

```
\d+ t_normal2;
```

返回结果如下：

```
                     Table "public.t_normal2"
   Column     |     Type      | Modifiers | Storage  | Stats target | Description
--------------+---------------+-----------+----------+--------------+-------------
id            | integer       |           | plain    |              |
col_varying   | varchar(30)   |           | extended |              |
col_varchar   | varchar(30)   |           | extended |              |
col_character | character(30) |           | extended |              |
col_char      | character(30) |           | extended |              |
col_text      | text          |           | extended |              |
col_name      | name          |           | plain    |              |
Has OIDs: no
Options: orientation=row, compression=no, fillfactor=80
```

4. 清理测试数据。

```
DROP TABLE t_normal2 CASCADE;
```

示例3： 向视图中插入数据

1. 创建视图。

```
CREATE TABLE t_normal2(id INT,
                    col_varying CHARACTER VARYING(30),
                    col_varchar VARCHAR(30),
                    col_character CHARACTER(30),
                    col_char CHAR(30),
                    col_text TEXT,
                    col_name NAME);
INSERT INTO t_normal2 (id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(1,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
CREATE OR REPLACE VIEW v_normal2 AS SELECT * FROM t_normal2;
```

2. 向视图中插入数据。

```
INSERT INTO v_normal2(id,col_varying,col_varchar,col_character,col_char,col_text,col_name) VALUES(2,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
SELECT * FROM t_normal2;
```

返回结果如下：

```
id | col_varying | col_varchar |        col_character         |           col_char           |  col_text  | col_name
----+-------------+-------------+------------------------------+------------------------------+------------+----------
1 | 测试abc123  | 测试abc123  | 测试abc123                   | 测试abc123                   | 测试abc123 | NAME
2 | 测试abc123  | 测试abc123  | 测试abc123                   | 测试abc123                   | 测试abc123 | NAME
(2 rows)
```

3. 清理测试数据。

```
DROP TABLE t_normal2 CASCADE;
```

示例4： 更新视图数据

1. 创建视图。

```
CREATE TABLE t_normal2(id INT,
                    col_varying CHARACTER VARYING(30),
                    col_varchar VARCHAR(30),
                    col_character CHARACTER(30),
                    col_char CHAR(30),
                    col_text TEXT,
                    col_name NAME);
INSERT INTO t_normal2 (id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(1,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
INSERT INTO t_normal2(id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(2,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
SELECT * FROM t_normal2;
CREATE OR REPLACE VIEW v_normal2 AS SELECT * FROM t_normal2;
```

2. 更新视图数据。

```
UPDATE v_normal2 SET col_varying = '更新def456';
SELECT * FROM t_normal2;
```

返回结果如下：

```
id | col_varying | col_varchar |        col_character         |           col_char           |  col_text  | col_name
---+-------------+-------------+------------------------------+------------------------------+------------+----------
1 | 更新def456  | 测试abc123  | 测试abc123                   | 测试abc123                   | 测试abc123 | NAME
2 | 更新def456  | 测试abc123  | 测试abc123                   | 测试abc123                   | 测试abc123 | NAME
(2 rows)
```

3. 清理测试数据。

```
DROP TABLE t_normal2 CASCADE;
```

示例5： 删除视图数据

1. 创建视图。

```
CREATE TABLE t_normal2(id INT,
                    col_varying CHARACTER VARYING(30),
                    col_varchar VARCHAR(30),
                    col_character CHARACTER(30),
                    col_char CHAR(30),
                    col_text TEXT,
                    col_name NAME);
INSERT INTO t_normal2 (id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(1,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
INSERT INTO t_normal2(id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(2,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
SELECT * FROM t_normal2;
CREATE OR REPLACE VIEW v_normal2 AS SELECT * FROM t_normal2;
```

2. 删除视图数据。

```
DELETE FROM v_normal2 WHERE id = 1;
SELECT * FROM v_normal2;
SELECT * FROM t_normal2;
```

返回结果如下：

```
 id | col_varying | col_varchar |        col_character         |           col_char           |  col_text  | col_name
----+-------------+-------------+------------------------------+------------------------------+------------+----------
  2 | 更新def456  | 测试abc123   | 测试abc123                   | 测试abc123                   | 测试abc123 | NAME
(1 row)

 id | col_varying | col_varchar |        col_character         |           col_char           |  col_text  | col_name
----+-------------+-------------+------------------------------+------------------------------+------------+----------
  2 | 更新def456  | 测试abc123  | 测试abc123                   | 测试abc123                   | 测试abc123 | NAME
(1 row)
```

3. 清理测试数据。

```
DROP TABLE t_normal2 CASCADE;
```

示例6： 删除视图

1. 创建视图。

```
CREATE TABLE t_normal2(id INT,
                    col_varying CHARACTER VARYING(30),
                    col_varchar VARCHAR(30),
                    col_character CHARACTER(30),
                    col_char CHAR(30),
                    col_text TEXT,
                    col_name NAME);
INSERT INTO t_normal2 (id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(1,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
INSERT INTO t_normal2(id, col_varying, col_varchar, col_character, col_char, col_text, col_name) VALUES(2,'测试abc123','测试abc123','测试abc123','测试abc123','测试abc123','NAME');
SELECT * FROM t_normal2;
CREATE OR REPLACE VIEW v_normal2 AS SELECT * FROM t_normal2;
```

2. 删除视图。

```
DROP VIEW v_normal2;
```

3. 清理测试数据。

```
DROP TABLE t_normal2;
```

示例7： 基表数据长度修改后，视图更新定义。

1. 创建测试表。

```
CREATE TABLE t1(
c1 int,
c2 char(20),
c3 varchar(20),
c4 varchar2(30),
c5 nvarchar(20),
c6 nvarchar2(20),
c7 character(10),
c8 character varying(30)
);
```

2. 创建基于 t1 表的视图 view_1_1。

```
CREATE VIEW view_1_1 AS SELECT * FROM t1;
```

3. 查看视图定义。

```
\d+ view_1_1;
```

返回结果如下：

```
                      View "public.view_1_1"
Column |     Type      | Modifiers | Storage  | Description | Attalias
--------+---------------+-----------+----------+-------------+----------
c1     | integer       |           | plain    |             |
c2     | character(20) |           | extended |             |
c3     | varchar(20)   |           | extended |             |
c4     | varchar(30)   |           | extended |             |
c5     | nvarchar2(20) |           | extended |             |
c6     | nvarchar2(20) |           | extended |             |
c7     | character(10) |           | extended |             |
c8     | varchar(30)   |           | extended |             |
View definition:
SELECT  *
FROM t1;
```

4. 修改基表类型长度。

```
ALTER TABLE t1 MODIFY c2 char(30);
ALTER TABLE t1 MODIFY c3 varchar(30);
ALTER TABLE t1 MODIFY c4 varchar2(20);
ALTER TABLE t1 MODIFY c5 nvarchar(30);
ALTER TABLE t1 MODIFY c6 nvarchar2(30);
ALTER TABLE t1 MODIFY c7 character(20);
ALTER TABLE t1 MODIFY c8 character varying(20);
```

5. 查看视图定义。

```
\d+ view_1_1;
```

返回结果如下：

```
                      View "public.view_1_1"
Column |     Type      | Modifiers | Storage  | Description | Attalias
--------+---------------+-----------+----------+-------------+----------
c1     | integer       |           | plain    |             |
c2     | character(20) |           | extended |             |
c3     | varchar(20)   |           | extended |             |
c4     | varchar(30)   |           | extended |             |
c5     | nvarchar2(20) |           | extended |             |
c6     | nvarchar2(20) |           | extended |             |
c7     | character(10) |           | extended |             |
c8     | varchar(30)   |           | extended |             |
View definition:
SELECT  *
FROM t1;
```

查询到的视图定义与修改表前的定义一致，这是由于在修改表后，需要在视图上执行一次查询才能够刷新视图。

6. 查询视图。

```
SELECT * FROM view_1_1;
```

7. 重新查看视图定义。

```
\d+ view_1_1;
```

返回结果如下：

```
                      View "public.view_1_1"
Column |     Type      | Modifiers | Storage  | Description | Attalias
--------+---------------+-----------+----------+-------------+----------
c1     | integer       |           | plain    |             |
c2     | character(30) |           | extended |             |
c3     | varchar(30)   |           | extended |             |
c4     | varchar(20)   |           | extended |             |
c5     | nvarchar2(30) |           | extended |             |
c6     | nvarchar2(30) |           | extended |             |
c7     | character(20) |           | extended |             |
c8     | varchar(20)   |           | extended |             |
View definition:
SELECT t1.c1, t1.c2, t1.c3, t1.c4, t1.c5, t1.c6, t1.c7, t1.c8
FROM t1;
```

8. 清理测试数据。

```
DROP TABLE t1;
DROP VIEW view_1_1;
```

创建和管理序列

背景信息

序列Sequence是用来产生唯一整数的数据库对象。序列的值是按照一定规则自增的整数。因为自增所以不重复，因此说Sequence具有唯一标识性。这也是Sequence常被用作主键的原因。

通过序列使某字段成为唯一标识符的方法有两种：

- 一种是声明字段的类型为序列整型，由数据库在后台自动创建一个对应的Sequence。
- 另一种是使用CREATE SEQUENCE自定义一个新的Sequence，然后将nextval（‘sequence_name’）函数读取的序列值，指定为某一字段的默认值，这样该字段就可以作为唯一标识符。

操作步骤

方法一： 声明字段类型为序列整型来定义标识符字段。例如：

```
CREATE TABLE t1(id serial,name  text);
```

返回结果如下，表示创建成功。

```
NOTICE:  CREATE TABLE will create implicit sequence "t1_id_seq" for serial column "t1.id"
CREATE TABLE
```

方法二： 创建序列，并通过nextval（‘sequence_name’）函数指定为某一字段的默认值。

1. 创建序列。

   ```
   CREATE SEQUENCE seq1 cache 100;
   ```

2. 指定为某一字段的默认值，使该字段具有唯一标识属性。

   ```
   CREATE TABLE t2 (
           id   int not null default nextval('seq1'),
           name text
       );
   ```

3. 指定序列与列的归属关系。

将序列和一个表的指定字段进行关联。这样，在删除此指定字段或序列所在表的时候会自动删除已关联的序列。

```
ALTER SEQUENCE seq1 OWNED BY t2.id;
```

4. 清理环境。

```
DROP TABLE t1,t2 CASCADE;
```

> **说明**
>
> 除了为序列指定了cache，方法二所实现的功能基本与方法一类似。但是一旦定义cache，序列将会产生空洞（序列值为不连贯的数值，如：1.4.5），并且不能保序。另外为某序列指定从属列后，若该列删除，对应的sequence也会被删除。虽然数据库并不限制序列只能为一列产生默认值，但最好不要多列共用同一个序列。

创建和管理用户与角色

用户与角色是操作数据库的主体。本节主要介绍 VexDB 数据库中创建与管理用户、角色的方式。

用户

VexDB 包含一个或多个已命名数据库。用户和角色在整个VexDB范围内是共享的，但是其数据并不共享。即用户可以连接任何数据库，但当连接成功后，任何用户都只能访问连接请求里声明的那个数据库。

非三权分立下，VexDB用户帐户只能由系统管理员、初始用户或拥有CREATEROLE属性的用户创建和删除。三权分立时，只允许系统管理员创建系统管理员，审计员创建审计用户，安全员创建安全用户。不需要再指定用户的管理类型标识。只允许系统管理员删除系统管理员，审计员删除审计用户，安全员删除安全用户。

在用户登录VexDB时会对其进行身份验证。用户可以拥有数据库和数据库对象（例如表），并且可以向用户和角色授予对这些对象的权限以控制谁可以访问哪个对象。除系统管理员外，具有CREATEDB属性的用户可以创建数据库并授予对这些数据库的权限。

创建、修改和删除用户

- 要创建用户，请使用CREATE USER语句。

  例如：创建用户joe，并设置用户拥有CREATEDB属性。

```
CREATE USER joe WITH CREATEDB PASSWORD "Vbase@123";
```

- 要创建系统管理员，请使用带有 SYSADMIN 选项的CREATE USER语句 。

- 要删除现有用户，请使用DROP USER语句。
- 要更改用户帐户（例如，重命名用户或更改密码），请使用ALTER USER语句。
- 要查看用户列表，请查询视图PG_USER：

```
SELECT * FROM pg_user;
```

- 要查看用户属性，请查询系统表PG_AUTHID：

```
SELECT * FROM pg_authid;
```

普通用户口令

VexDB支持设置强制新建用户首次登录修改密码，可通过设置vb_enable_password_force_alter参数为on实现，参数默认启用。

- 在password_force_alter参数为on时，管理员新建普通用户，为新建用户设置初始密码（初始密码复杂度受密码复杂度相关参数控制）。新建用户在首次登录时，必须修改密码后才可正常操作。
- 在password_force_alter参数为on时，管理员可以按照密码复杂度要求修改普通用户密码。普通用户在使用管理员修改的密码首次登录后，必须修改密码方可正常操作。
- 在password_force_alter参数为off时，对普通用户首次登录或连接不强制要求修改密码。

私有用户

对于有多个业务部门，各部门间使用不同的数据库用户进行业务操作，同时有一个同级的数据库维护部门使用数据库管理员进行维护操作的场景下，业务部门可能希望在未经授权的情况下，管理员用户只能对各部门的数据进行控制操作（DROP、ALTER、TRUNCATE），但是不能进行访问操作（INSERT、DELETE、UPDATE、SELECT、COPY）。即针对管理员用户，表对象的控制权和访问权要能够分离，提高普通用户数据安全性。

三权分立情况下，管理员对其他用户放在属于各自模式下的表无权限。但是，这种无权限包含了无控制权限，因此不能满足上面的诉求。为此，VexDB 提供了私有用户方案。即在非三权分立模式下，创建具有 INDEPENDENT 属性的私有用户。

```
CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY "Vbase@123";
```

针对该用户的对象，系统管理员和拥有 CREATEROLE 属性的安全管理员在未经其授权前，只能进行控制操作（DROP、ALTER、TRUNCATE），无权进行INSERT、DELETE、SELECT、UPDATE、COPY、GRANT、REVOKE、ALTER OWNER操作。

> **说明**

- PG_STATISTIC系统表和PG_STATISTIC_EXT系统表存储了统计对象的一些敏感信息，如高频值MCV。进行三权分立后系统管理员仍可以通过访问这两张系统表，得到统计信息里的这些信息。
- 在某些特殊场景下，系统无法保证私有用户数据对初始用户或系统管理员绝对不可见。例如，初始用户或系统管理员通过解析本地数据文件或通过构造特殊函数或操作符被私有用户调用等方式，可能会获取私有用户数据。私有用户如果希望进一步提升数据的安全性，建议对数据进行加密保护。

永久用户

VexDB 提供永久用户方案，即创建具有 PERSISTENCE属性的永久用户。

仅允许初始用户创建、修改和删除具有 PERSISTENCE 属性的永久用户。

```
CREATE USER user_persistence WITH persistence IDENTIFIED BY "Admin@123";
```

角色

角色是一组用户的集合。通过GRANT把角色授予用户后，用户即具有了角色的所有权限。推荐使用角色进行高效权限分配。例如，可以为设计、开发和维护人员创建不同的角色，将角色GRANT给用户后，再向每个角色中的用户授予其工作所需数据的差异权限。在角色级别授予或撤消权限时，这些更改将作用到角色下的所有成员。

VexDB 提供了一个隐式定义的拥有所有角色的组：PUBLIC，所有创建的用户和角色默认拥有PUBLIC所拥有的权限。关于PUBLIC默认拥有的权限请参考GRANT。要撤销或重新授予用户和角色对PUBLIC的权限， 可通过在GRANT和REVOKE指定关键字PUBLIC实现。

注意事项

- 以gs_role_开头的角色名作为数据库的内置角色保留名，禁止新建以 `gs_role_` 开头的用户或角色，也禁止将已有的用户或角色重命名为以 `gs_role_` 开头。
- 禁止对内置角色的ALTER和DROP操作。
- 内置角色默认没有LOGIN权限，不设预置密码。
- vsql 元命令 `\du` 和 `\dg` 不显示内置角色的相关信息，但若显示指定了 pattern 为特定内置角色则会显示。
- 三权分立关闭时，初始用户、具有SYSADMIN权限的用户和具有内置角色ADMIN OPTION权限的用户有权对内置角色执行GRANT/REVOKE管理。三权分立打开时，初始用户和具有内置角色ADMIN OPTION权限的用户有权对内置角色执行GRANT/REVOKE管理。例如：

  ```
  GRANT gs_role_signal_backend TO user1;
  REVOKE gs_role_signal_backend FROM user1;
  ```

查询角色

要查看所有角色，请查询系统表PG_ROLES：

```
SELECT * FROM PG_ROLES;
```

创建、修改和删除角色

非三权分立时，只有系统管理员和具有CREATEROLE属性的用户才能创建、修改或删除角色。三权分立下，只有初始用户和具有CREATEROLE属性的用户才能创建、修改或删除角色。

- 要创建角色，请使用CREATE ROLE语句。
- 要在现有角色中添加或删除用户，请使用ALTER ROLE语句。
- 要删除角色，请使用DROP ROLE语句。DROP ROLE只会删除角色，并不会删除角色中的成员用户帐户。

启用/禁用角色

VexDB 支持数据库角色的启用和禁用，若角色被禁用，则被禁用角色的成员角色将无法使用被禁用角色所拥有的权限。启用/禁用语法如下：

- 启用角色

  ```
  SELECT vb_set_role('ROLE_NAME',0);
  ```

- 禁用角色

  ```
  SELECT vb_set_role('ROLE_NAME',1);
  ```

内置角色

VexDB 提供了一组默认角色，以gs_role_开头命名。它们提供对特定的、通常需要高权限的操作的访问，可以将这些角色GRANT给数据库内的其他用户或角色，让这些用户能够使用特定的功能。在授予这些角色时应当非常小心，以确保它们被用在需要的地方。**表1** 描述了内置角色允许的权限范围：

表1 内置角色权限描述

角色	权限描述
gs_role_copy_files	具有执行 copy ... to/from filename 的权限，但需要先打开 GUC 参数 enable_copy_server_files。
gs_role_signal_backend	具有调用函数 pg_cancel_backend、pg_terminate_backend 和 pg_terminate_session 来取消或终止其他会话的权限，但不能操作属于初始用户和 PERSISTENCE 用户的会话。
gs_role_tablespace	具有创建表空间（tablespace）的权限。
gs_role_replication	具有调用逻辑复制相关函数的权限，例如 kill_snapshot、pg_create_logical_replication_slot、pg_create_physical_replication_slot、pg_drop_replication_slot、pg_replication_slot_advance、pg_create_physical_replication_slot_extern、pg_logical_slot_get_changes、pg_logical_slot_peek_changes、pg_logical_slot_get_binary_changes、pg_logical_slot_peek_binary_changes。
gs_role_account_lock	具有加解锁用户的权限，但不能加解锁初始用户和 PERSISTENCE 用户。
gs_role_pldebugger	具有执行 dbe_pldebugger 模式下的 A 调试函数的权限。
gs_role_directory_create	具有执行创建 directory 对象的权限，但需要先打开 GUC 参数 enable_access_server_directory。
gs_role_directory_drop	具有执行删除 directory 对象的权限，但需要先打开 GUC 参数 enable_access_server_directory。

用户权限设置

直接授予权限

给用户直接授予某对象的权限，请使用 GRANT。

将 Schema 中的表或者视图对象授权给其他用户或角色时，需要将表或视图所属 Schema 的 USAGE 权限同时授予该用户或角色。否则用户或角色将只能看到这些对象的名称，并不能实际进行对象访问。

例如，下面示例将 Schema tpcds（tpcds Schema 已提前创建）的权限赋给用户 joe 后，将表 tpcds.web_returns 的 select 权限赋给用户 joe。

```
GRANT USAGE ON SCHEMA tpcds TO joe;
GRANT SELECT ON TABLE tpcds.web_returns to joe;
```

指定用户继承角色的权限

给用户指定角色，使用户继承角色所拥有的对象权限。

1. 创建角色。

   新建一个角色lily，同时给角色指定系统权限CREATEDB：

   ```
   CREATE ROLE lily WITH CREATEDB PASSWORD "Vbase@123";
   ```

2. 给角色赋予对象权限。

   例如，将模式 tpcds 的权限赋给角色 lily 后，将表 tpcds.web_returns 的 select 权限赋给角色 lily。

```
GRANT USAGE ON SCHEMA tpcds TO lily;
GRANT SELECT ON TABLE tpcds.web_returns to lily;
```

3. 将角色的权限赋予用户。

```
GRANT lily to joe;
```

   当将角色的权限赋予用户时，角色的属性并不会传递到用户。

回收权限

回收用户权限，请使用REVOKE。

```
REVOKE {privileges_options} [ON ... ] FROM username;
```

数据库权限

表2 数据库权限说明

权限	说明
SELECT	允许对指定的表、视图、序列执行 SELECT 命令，UPDATE 或 DELETE 时也需要对应字段上的 SELECT 权限。
INSERT	允许对指定的表执行 INSERT 命令。
UPDATE	允许对声明的表中任意字段执行 UPDATE 命令。 通常，UPDATE 命令也需要 SELECT 权限来查询出哪些行需要更新。SELECT... FOR UPDATE 和 SELECT... FOR SHARE 除了需要 SELECT 权限外，还需要 UPDATE 权限。
DELETE	允许执行 DELETE 命令删除指定表中的数据。通常，DELETE 命令也需要 SELECT 权限来查询出哪些行需要删除。
TRUNCATE	允许执行 TRUNCATE 语句删除指定表中的所有记录。
REFERENCES	允许创建一个外键约束，必须拥有参考表和被参考表的 REFERENCES 权限。
TRIGGER	允许创建触发器。
CREATE	CREATE 行为根据级别而不同，包括： • 对于数据库级别的 CREATE 权限，允许在数据库里创建新的模式。 • 超级用户和数据库所有者默认拥有此权限。 • 对于模式级别的 CREATE 权限，允许在模式中创建新的对象。例如表、视图、索引、函数、数据类型等。模式所有者和超级用户默认拥有此权限。如果要重命名一个对象，用户除了必须是该对象的所有者外，还必须拥有该对象所在模式的 CREATE 权限。 • 对于表空间级别的 CREATE 权限，允许在表空间中创建表，允许在创建数据库和模式的时候把该表空间指定为缺省表空间。超级用户默认拥有此权限。
CONNECT	允许用户连接到指定的数据库。
TEMPORARY	允许在指定的数据库中创建临时表。
EXECUTE	允许使用指定的函数，以及利用这些函数实现的操作符。
USAGE	• 对于过程语言，允许用户在创建函数的时候指定过程语言。 • 对于模式，USAGE 允许访问包含在指定模式中的对象，若没有该权限，则只能看到这些对象的名称。 • 对于序列，USAGE 允许使用 nextval 函数。 • 对于 Data Source 对象，USAGE 是指访问权限，也是可赋予的所有权限，即 USAGE 与 ALL PRIVILEGES 等价。
ALTER	允许用户修改指定对象的属性，但不包括修改对象的所有者和修改对象所在的模式。
DROP	允许用户删除指定的对象。
COMMENT	允许用户定义或修改指定对象的注释。
INDEX	允许用户在指定表上创建索引，并管理指定表上的索引，还允许用户对指定表执行 REINDEX 和 CLUSTER 操作。
VACUUM	允许用户对指定的表执行 ANALYZE 和 VACUUM 操作。

ALL PRIVILEGES	一次性给指定用户/角色赋予所有可赋予的权限。只有系统管理员有权执行 GRANT ALL PRIVILEGES。
ANY	ANY 权限是一类特殊的权限，被授予 ANY 权限的用户可以在数据库范围内执行上述权限允许的操作。例如，被授予 CREATE ANY TABLE 权限的用户能够在数据库范围内创建数据库。 说明：被授予 CREATE ANY TABLE 权限的用户在同名 schema 下创建表，则该表的属主是该 schema 的属主。ANY 权限与原有的权限相互无影响。

数据库工具

本章介绍 VexDB 为用户提供的工具，可以帮助用户更加方便的管理数据库。

分类	工具名称	简介
数据库管理工具	vsql	数据库连接工具，用户可以通过此工具连接服务器并对其进行操作和维护，除了具备操作数据库的基本功能，vsql 还提供了若干高级特性。
	vb_ctl	数据库服务控制工具，可以用来启停数据库服务和查询数据库状态。
	vb_guc	设置数据库的配置参数。
	vb_initdb	用于初始化数据库。
	vb_licensetool	用于查看 license 的相关信息。
备份恢复工具	gs_tar	用于解压 vb_basebackup 命令生成的归档 tar 文件。
	vb_basebackup	基础的物理备份工具，可以将整个实例的数据进行备份。
	vb_dump	用于导出数据库相关信息的一种逻辑备份工具，用户可以自定义导出一个数据库或其中的对象（模式、表、视图等），回收站对象除外。
	vb_dumpall	用于导出所有数据库相关信息的工具。它可以导出 VexDB 数据库的所有数据，包括默认数据库 postgres 的数据、自定义数据库的数据以及 VexDB 所有数据库公共的全局对象。
	vb_probackup	用于管理 VexDB 数据库物理备份和恢复的工具。它可以对 VexDB 实例进行定期备份，以便在数据库出现故障时能够恢复服务器。
	vb_restore	导入由 vb_dump 生成的备份文件。
统一数据库管理工具	has_ctl	在高可用场景下控制数据库实例服务的工具。
	cm_install/cm_uninstall	部署和卸载 HAS。
	gs_install	用于完成 VexDB 的安装和部署。
	gs_postuninstall	清理准备 VexDB 环境阶段所做配置，使得卸载后的环境得到清理。
	gs_preinstall	完成 VexDB 的环境配置，以保证 VexDB 安装的正常进行。
	gs_uninstall	完成 VexDB 的卸载。
其他工具	vexdb	vexdb 是 VexDB 数据库的主线程，也是一个可执行的命令，能够通过其启动一个数据库线程。

cm_install/cm_uninstall

使用cm_install工具可以在未部署HAS的VexDB数据库集群上部署HAS，cm_uninstall可以在已部署HAS的VexDB集群上卸载掉HAS，且不会影响DN集群。

注意事项

- 执行cm_install之后默认会启动集群。
- 需要以个人用户执行。
- 本工具允许在集群停止状态下进行安装或卸载HAS工具，但是需要注意的是，集群停止之前的主机应该为XML配置的初始主机，否则安装或卸载HAS工具后再次启动后集群的主机会与集群停止前的主机不一致。
- 如果是在集群停止状态下卸载HAS工具，卸载后会删除动态文件，但是由于集群处于停止状态，所以无法生成，如果需要的话可以在gs_om启动集群后使用 `gs_om -t refreshconf` 命令重新生成即可。
- 使用该工具前需要进入到该工具所在的cm_tool目录下。
- 如果执行cm_install后检查到主机的term值为0，为了保证后续业务正常，则需要用户根据提示执行 `cm_ctl stop` 和 `cm_ctl start` 重启。

前提条件

- VexDB集群是通过OM工具安装的，需要有OM工具。
- 集群各节点之间存在互信，通常通过OM工具安装的集群就能够保证这一点。
- VexDB集群的版本号与待安装的HAS的版本号相同。
- 部署HAS前集群状态需要为正常或停止，并且主机的term必须为非0值且为集群中最大。

使用方法

- 安装：

```
cm_install -? | --help
cm_install -X XMLFILE [-e envFile] --cmpkg=cmpkgPath
```

- 卸载：

```
cm_uninstall -? | --help
cm_uninstall -X XMLFILE [-e envFile] [--deleteData] [--deleteBinary]
```

参数说明

- -X

  XML文件的路径。

- -e

  环境变量文件的路径，默认值为 `~/.bashrc` 。

- –cmpkg

  HAS包的路径。

- –deleteData

  删除HAS数据目录，默认不删除。

- –deleteBinary

  删除HAS相关二进制，包括om_monitor、has_agent、has_server、has_ctl，默认不删除。

- -?, –help

  显示帮助信息。

gs_install

背景信息

数据库的部署是一个复杂的过程。VexDB提供了gs_install工具来帮助完成VexDB的安装和部署。

VexDB安装部署，要求用户指定配置文件，配置文件中会指定程序安装路径、实例数据目录、主备关系、实例数、各实例的业务IP端口等信息。

前提条件

- 已成功执行前置脚本gs_preinstall。
- 用户需确保各个节点上的locale保持一致。
- 需要使用前置时设置的VexDB用户进行安装操作。

语法格式

- 安装VexDB

```
gs_install -X XMLFILE [--init-parameter="PARAMETER" [...]] [--dn-guc="PARAMETER" [...]] [--alarm-component=ALARMCOMPONENT] [--time-out=SECS] [-l LOGFILE] [--enable-perf-config] [--dorado-cluster-mode="P
```

> **说明**
>
> 安装时若不指定字符集，默认字符集为SQL_ASCII，为简化和统一区域locale默认设置为C，若想指定其他字符集和区域，请在安装时使用参数 `--init-parameter="--locale=LOCALE"` 来指定，LOCALE为新数据库设置缺省的区域。

- 显示帮助信息

```
gs_install -? | --help
```

- 显示版本号信息

```
gs_install -V | --version
```

参数说明

- -X

  VexDB配置文件。

  取值范围：xml配置文件。

- -l

  指定安装日志文件及日志文件存放的路径。

  当不明确指定-l，但在XML文件中配置了gaussdbLogPath时，默认值为“gaussdbLogPath的值、用户名和om/gs_install-YYYY-MM-DD_hhmmss.log”的组合。

- –init-parameter=PARAM

  数据实例参数指定。

  取值范围请参见vb_initdb的参数说明，其中对-A、-D、-U、-C、-X参数的设置不生效。

- –dn-guc=PARAM

  配置参数。

取值范围请参见vb_guc的参数说明。

- –time-out=SECS

  启动超时等待时间。单位：s。

  若–time-out取值过小，有可能会因为需要启动的实例数过多而引起超时。若在超时时间内启动不成功，则会报错启动超时，但是VexDB会在后台继续启动。可以等待一段时间后，查询VexDB状态，检查VexDB是否启动成功。

  取值范围：正整数，若无特殊需求，不需要配置该参数，系统会自动调整超时时间。

- –alarm-component=alarm_component

  指定告警上报组件的绝对路径。

- –enable-perf-config

  在install结束后，调用gs_perfconfig调整数据库配置，以使VexDB获得比较好的性能。

- –dorado-cluster-mode=PARAM

  指定存储设备的集群模式，dorado-cluster-mode 主集群or备集群。

  取值范围：

  - primary：以资源池化dorado双集群主集群安装。
  - standby：以资源池化dorado双集群备集群安装。

- -?, –help

  显示帮助信息。

- -V, –version

  显示版本号信息。

示例

使用gs_install脚本进行VexDB安装。

```
gs_install -X /opt/software/vexdb/clusterconfig.xml
Parsing the configuration file.
Check preinstall on every node.
Successfully checked preinstall on every node.
Creating the backup directory.
Successfully created the backup directory.
begin deploy..
Installing the cluster.
begin prepare Install Cluster..
Checking the InstallationGuide environment on all nodes.
begin install Cluster..
Installing applications on all nodes.
Successfully installed APP.
begin init Instance..
encrypt cipher and rand files for database.
Please enter password for database:
Please repeat for database:
begin to create CA cert files
The sslcert will be generated in /opt/gaussdb/cluster/app/share/sslcert/om
Cluster InstallationGuide is completed.
Configuring.
Deleting instances from all nodes.
Successfully deleted instances from all nodes.
Checking node configuration on all nodes.
Initializing instances on all nodes.
Updating instance configuration on all nodes.
Check consistence of memCheck and coresCheck on DN nodes.
Successful check consistence of memCheck and coresCheck on all nodes.
Configuring pg_hba on all nodes.
Configuration is completed.
Successfully started cluster.
Successfully installed application.
```

相关命令

gs_postuninstall、gs_preinstall、gs_uninstall

gs_postuninstall

背景信息

VexDB提供了gs_postuninstall工具来帮助清理准备VexDB环境阶段所做配置，使得卸载后的环境得到清理。

前提条件

- VexDB卸载执行成功。

- root用户互信可用。
- 只能使用root用户执行gs_postuninstall命令(执行前需source安装用户的环境变量)。
- 若为环境变量分离的模式安装的数据库，root和普通用户在使用该工具前都需要source环境变量分离文件ENVFILE。
- 若为环境变量不分离的模式，root用户在使用该工具时需要source安装用户的 `.bashrc` 文件。

语法格式

- VexDB清理用户、用户组以及虚拟IP清理

```
gs_postuninstall -U USER -X XMLFILE [-L] [--delete-user] [--delete-group] [-l LOGFILE]
```

- 显示帮助信息

```
gs_postuninstall -? | --help
```

- 显示版本号信息

```
gs_postuninstall -V | --version
```

参数说明

- -U

  运行VexDB的操作系统用户名。

  取值范围：字符串，要符合标识符的命名规范。

- -X

  VexDB配置文件路径。

  取值范围：xml文件的存储路径。

- -L

  只清理本主机的环境。

  如果VexDB内某主机做单机环境清理后，VexDB不能再做全量环境清理。

- –delete-user

删除-U参数指定的操作系统用户。

如果在Redhat环境下，且用户名与用户组名相同，选择此项必须指定 `--delete-group` 参数。

- –delete-group

  删除操作系统用户所在的用户组（选择此选项必须指定 `--delete-user` 参数）。

- -l

  指定日志文件名及路径。在内部会自动给日志名添加一个时间戳。

  当既不明确指定-l，又不在XML文件中配置gaussdbLogPath时，默认值为： `“/var/log/gaussdb/om/gs_local-YYYY-MMDD_hhmmss.log”`。

  > **说明**
  >
  > 由于在执行gs_postuninstall后，系统会自动删除VexDB相关目录（包含$GAUSSLOG目录）。因此建议用户通过该参数指定日志文件到非VexDB相关路径。

- -?, –help

  显示帮助信息。

- -V, –version

  显示版本号信息。

示例

清理主机的环境：

```
#环境变量分离模式
source /home/vexdb/om_env
#环境变量不分离模式
source /home/vexdb/.bashrc
Parsing the configuration file.
Successfully parsed the configuration file.
Check log file path.
Successfully checked log file path.
Checking unpreInstallationGuide.
Successfully checked unpreInstallationGuide.
Deleting the instance's directory.
Successfully deleted the instance's directory.
Deleting the InstallationGuide directory.
Successfully deleted the InstallationGuide directory.
Deleting the temporary directory.
Successfully deleted the temporary directory.
Deleting remote OS user.
Successfully deleted remote OS user.
Deleting software packages and environmental variables of other nodes.
Successfully deleted software packages and environmental variables of other nodes.
Deleting logs of other nodes.
Successfully deleted logs of other nodes.
Deleting software packages and environmental variables of the local node.
Successfully deleted software packages and environmental variables of the local nodes.
Deleting local OS user.
Successfully deleted local OS user.
Deleting local node's logs.
Successfully deleted local node's logs.
Successfully cleaned environment.
```

相关命令

gs_preinstall、gs_uninstall

gs_preinstall

背景信息

VexDB提供了gs_preinstall工具来帮助完成VexDB的环境配置，以保证VexDB安装的正常进行。

注意事项

- 用户需要检查上层目录权限，保证安装用户对安装包和配置文件目录读写执行的权限。
- xml文件中各主机的名称与IP映射配置正确。

- 只能使用 root 用户执行 gs_preinstall 命令。
- 执行 gs_preinstall 会清理/etc/hosts中的VexDB映射信息，可能存在已部署数据库连接丢失风险，可以在安装结束后手动添加其它数据库映射至/etc/hosts文件中。
- gs_preinstall 指定的用户不支持使用 `-d` 手动选定家目录，仅支持系统默认创建的 `/home` 下和用户同名的 `/home` 目录。

本产品支持同一操作系统大版本下的小版本混合部署，其支持列表如下：

```
CentOS 6.4/6.5/6.6/6.7/6.8/6.9
CentOS 7.0/7.1/7.2/7.3/7.4
openEuler
```

语法格式

- 准备VexDB环境

```
gs_preinstall -U USER -G GROUP -X XMLFILE [-L] [--skip-os-set] [--env-var="ENVVAR" [...]] [--sep-env-file=MPPRCFILE] [--skip-hostname-set] [-l LOGFILE] [--non-interactive]
```

- 显示帮助信息

```
gs_preinstall -? | --help
```

- 显示版本号信息

```
gs_preinstall -V | --version
```

参数说明

- -U

  运行VexDB的操作系统用户名。

  取值范围：字符串，要符合标识符的命名规范。

  > **说明**
  >
  > 配置-U参数值时，字符串中不能包含 | 、; 、& 、$ 、< 、> 、`、’ 、 、{ 、} 、(、) 、[、] 、~ 、* 、? 特殊字符。

- -G

  运行VexDB的操作系统用户的群组名。

取值范围：字符串，要符合标识符的命名规范。

- -X

  VexDB配置文件路径。

  取值范围：xml文件的存储路径。

- -L

  指定该参数则仅给当前节点准备好VexDB安装环境。适用于不允许使用root互信的场景和设置安全加固参数PermitRootLogin为no的场景。需要满足如下前提条件：

  （1）安装包和VexDB配置文件已提前上传到所有主机；

  （2）VexDB安装用户已提前创建好并建立好互信关系；

  （3）用户需要自己保证各台主机上所执行命令的一致性；

  （4）手动将所有节点的主机名和ip映射关系写入各个主机的/etc/hosts，并在每个映射关系后边加入注释内容：`#Gauss OM IP Hosts Mapping`。

- –skip-os-set

  是否设置操作系统参数。默认设置系统参数。如果指定该参数则不设置。

- –env-var= “ENVVAR”

  配置普通用户环境变量。可以指定多个。

  取值范围：字符串。

  > **说明**
  >
  > 本参数不能指定为VexDB默认创建的环境变量。否则，VexDB的环境变量将被覆盖。VexDB默认创建的环境变量见表1。

- –sep-env-file=MPPRCFILE

  保存环境变量的文件。如果指定该参数，表示将使用环境变量分离的版本。

  > **说明**
  >
  > 不能指定环境变量文件为VexDB相关目录或者和这些目录同名，不能和VexDB用户的目录相同。

- –skip-hostname-set

  是否将xml配置文件中主机名与IP的映射关系写入`“/etc/hosts”`文件中。默认写入，如果指定该参数则不写入。

- –unused-third-party

  是否使用om提供的三方库。默认使用，如果指定该参数则不使用。

- -l

  指定日志文件名及路径。在内部会自动给日志名添加一个时间戳。

  默认值：`/var/log/gaussdb/用户名/om/gs_preinstall.log`

  当不明确指定-l，但在XML文件中配置了gaussdbLogPath时，默认值为gaussdbLogPath的值、用户名和om/gs_preinstall-YYYY-MM-DD_hhmmss.log的组合。

- –non-interactive

  指定前置执行模式。

  - 当不指定该参数时，则为安全交互模式，在此模式下用户需要人机交互输入密码。
  - 当指定该参数时，为非交互模式，不需要进行人机交互。

- -?, –help

  显示帮助信息。

- -V, –version

  显示版本号信息。

表1 VexDB默认创建的环境变量

环境变量名称	说明
MPPDB_ENV_SEPARATE_PATH	VexDB 环境变量分离文件路径
GPHOME	VexDB 工具目录
PATH	VexDB 工具脚本目录
LD_LIBRARY_PATH	VexDB 引用第三方动态库路径
PYTHONPATH	python 软件路径
GAUSS_WARNING_TYPE	告警类型
GAUSSHOME	VexDB 安装路径
GAUSS_VERSION	VexDB 版本号
PGHOST	VexDB 用户的临时目录路径
GS_CLUSTER_NAME	VexDB 名称
GAUSSLOG	VexDB 日志路径
GAUSS_ENV	VexDB 环境变量标识

手动设置服务端SYSLOG配置

设置“/etc/syslog-ng/syslog-ng.conf”文件，在文件中添加如下内容：

```
template t_gaussdb {template("$DATE $SOURCEIP $MSGONLY\n");template_escape(no);};
source s_gaussdb{ udp(); };
filter f_gaussdb    { level(err,  crit) and match('GaussDB'); };
destination d_gaussdb { file("/var/log/syslog_MPPDB", template(t_gaussdb)); };
log { source(s_gaussdb); filter(f_gaussdb); destination(d_gaussdb); };
```

设置“/etc/sysconfig/syslog”文件，在文件中添加如下内容：

```
SYSLOGD_OPTIONS="-r -m 0"
KLOGD_OPTIONS="-x"
```

> **说明**
>
> 该配置需要在VexDB每台机器上都要修改。

示例

使用非交互方式进行VexDB环境准备：

```
./gs_preinstall -U vexdb -G dbgrp -X /opt/software/vexdb/clusterconfig.xml --non-interactive
Parsing the configuration file.
Successfully parsed the configuration file.
Installing the tools on the local node.
Successfully installed the tools on the local node.
Distributing package.
Successfully distributed package.
Installing the tools in the cluster.
Successfully installed the tools in the cluster.
Checking hostname mapping.
Successfully checked hostname mapping.
Checking OS version.
Successfully checked OS version.
Creating cluster's path.
Successfully created cluster's path.
Set and check OS parameter.
Setting OS parameters.
Successfully set OS parameters.
Warning: InstallationGuide environment contains some warning messages.
Please get more details by "/package_r8/sudo/gs_checkos -i A -h host179174,host179175,host179176".
Set and check OS parameter completed.
Preparing CRON service.
Successfully prepared CRON service.
Preparing SSH service.
Successfully prepared SSH service.
Setting user environmental variables.
Successfully set user environmental variables.
Configuring alarms on the cluster nodes.
Successfully configured alarms on the cluster nodes.
Setting the dynamic link library.
Successfully set the dynamic link library.
Setting finish flag.
Successfully set finish flag.
PreInstallationGuide succeeded.
```

使用交互模式进行VexDB环境准备：

```
./gs_preinstall -U vexdb -G dbgrp -X /opt/software/vexdb/clusterconfig.xml
Parsing the configuration file.
Successfully parsed the configuration file.
Installing the tools on the local node.
Successfully installed the tools on the local node.
Are you sure you want to create trust for root (yes/no)? yes
Please enter password for root.
Password:
Creating SSH trust for the root permission user.
Please enter password for current user[root].
Password:
Checking network information.
All nodes in the network are Normal.
Successfully checked network information.
Creating SSH trust.
Creating the local key file.
Successfully created the local key files.
Appending local ID to authorized_keys.
Successfully appended local ID to authorized_keys.
Updating the known_hosts file.
Successfully updated the known_hosts file.
Appending authorized_key on the remote node.
Successfully appended authorized_key on all remote node.
Checking common authentication file content.
Successfully checked common authentication content.
Distributing SSH trust file to all node.
Successfully distributed SSH trust file to all node.
Verifying SSH trust on all hosts.
Successfully verified SSH trust on all hosts.
Successfully created SSH trust.
Successfully created SSH trust for the root permission user.
Distributing package.
Successfully distributed package.
Are you sure you want to create the user[vexdb] and create trust for it (yes/no)? yes
Please enter password for cluster user.
Password:
Please enter password for cluster user again.
Password:
Creating [vexdb] user on all nodes.
Successfully created [vexdb] user on all nodes.
Installing the tools in the cluster.
Successfully installed the tools in the cluster.
Checking hostname mapping.
Successfully checked hostname mapping.
Creating SSH trust for [vexdb] user.
Please enter password for current user[vexdb].
Password:
Checking network information.
All nodes in the network are Normal.
Successfully checked network information.
Creating SSH trust.
Creating the local key file.
Successfully created the local key files.
Appending local ID to authorized_keys.
Successfully appended local ID to authorized_keys.
Updating the known_hosts file.
```

```
Successfully updated the known_hosts file.
Appending authorized_key on the remote node.
Successfully appended authorized_key on all remote node.
Checking common authentication file content.
Successfully checked common authentication content.
Distributing SSH trust file to all node.
Successfully distributed SSH trust file to all node.
Verifying SSH trust on all hosts.
Successfully verified SSH trust on all hosts.
Successfully created SSH trust.
Successfully created SSH trust for [omm] user.
Checking OS version.
Successfully checked OS version.
Creating cluster's path.
Successfully created cluster's path.
Set and check OS parameter.
Setting OS parameters.
Successfully set OS parameters.
Warning: InstallationGuide environment contains some warning messages.
Please get more details by "/package_r8/sudo/gs_checkos -i A -h host179174,host179175,host179176".
Set and check OS parameter completed.
Preparing CRON service.
Successfully prepared CRON service.
Preparing SSH service.
Successfully prepared SSH service.
Setting user environmental variables.
Successfully set user environmental variables.
Configuring alarms on the cluster nodes.
Successfully configured alarms on the cluster nodes.
Setting the dynamic link library.
Successfully set the dynamic link library.
Setting finish flag.
Successfully set finish flag.
PreInstallationGuide succeeded.
```

常见问题

GAUSS-53700: “Failed to execute the command: %s.”

错误原因： 这是由于OpenSSL的随机数生成器没有正常工作导致的。这可能是因为系统缺少足够的熵，或者 OpenSSL 的配置有问题。 以下是常用的检查项。

解决办法:

1. 增加系统熵池：如果系统缺少足够的熵，可以通过增加系统活动来增加熵池的随机性。例如，可以打开几个终端窗口并执行一些随机的操作，如移动鼠标、键盘输入等。
2. 检查 OpenSSL 配置：确保 OpenSSL 的配置正确，可以检查 OpenSSL 的配置文件和环境变量，确保它们指向正确的路径和文件。
3. 更新 OpenSSL 版本：如果 OpenSSL 版本较旧，可能会存在一些已知的问题和 bug。尝试更新到最新的稳定版本。
4. 检查文件权限：确保正在生成证书的目录和文件具有足够的权限，以便 OpenSSL 可以在其中生成证书。

5. 使用其他随机数生成器：如果以上方法都没有解决问题，可以尝试使用其他的随机数生成器替代 OpenSSL 默认的随机数生成器。

相关命令

gs_install

gs_postuninstall

gs_tar

功能描述

gs_tar工具用于解压vb_basebackup命令生成的归档tar文件。

通过vb_basebackup的压缩格式备份，主数据目录将被写入到一个名为base.tar的文件中，并且其他表空间将被以其 OID 命名，使用gs_tar可以解压该文件。

> **说明**
>
> - gs_tar命令当前仅支持解压通过vb_basebackup生成的归档文件。
> - 如果vb_basebackup制定了压缩等级，会生成以gz结尾的文件。此时需要使用gzip命令先解压缩生成tar包，之后才可以使用gs_tar命令解压生成的tar文件。

语法格式

```
gs_tar [OPTION]...
```

参数说明

- -D, –destination=DIRECTORY

  指定解压文件的输出目录，必选项。

- -F, –filename=FILENAME

  指定解压文件，必选项。

使用示例

1、创建存储备份文件的文件夹。

```
mkdir -p /home/vexdb/data/bak
```

2、通过vb_basebackup备份数据库（tar模式）。

```
vb_basebackup -D /home/vexdb/data/bak -F t -X fetch -h 127.0.0.1 -p 5432
```

3、切换至备份文件夹可以看到一个名为 base.tar 的文件。

```
cd /home/vexdb/data/bak
```

4、创建解压文件的输出目录。

```
mkdir -p /home/vexdb/data/bak1
```

5、解压tar文件至bak1目录下。

```
gs_tar -D  /home/vexdb/data/bak1  -F  base.tar
```

解压完成切换至/home/vexdb/data/bak1可以看到解压后的文件。

gs_uninstall

背景信息

VexDB提供了gs_uninstall工具来帮助完成VexDB的卸载。

语法

- 卸载VexDB

  ```
  gs_uninstall [--delete-data] [--clear-disk] [-L] [-l LOGFILE]
  ```

- 显示帮助信息

```
gs_uninstall -? | --help
```

- 显示版本号信息

```
gs_uninstall -V | --version
```

参数说明

- –delete-data

  删除数据文件。

- –clear-disk

  清除资源池化场景下LUN上的数据。

- -L

  只卸载本地主机。如果VexDB内某主机做单点卸载后，VexDB不能再做全量卸载。

- -l

  指定日志文件名及可访问的绝对路径。在内部会自动给日志名添加一个时间戳。

  - 当既不明确指定-l，又不在XML文件中配置gaussdbLogPath时，默认值为“$GAUSSLOG/om/gs_uninstall-YYYY-MM-DD_hhmmss.log”。
  - 当不明确指定-l，但在XML文件中配置了gaussdbLogPath时，默认值为“gaussdbLogPath/用户名/om/gs_uninstall-YYYY-MM-DD_hhmmss.log”。

  > **说明**
  >
  > 由于在执行gs_uninstall时，系统会自动删除VexDB相关目录。因此建议用户通过该参数指定日志文件到非VexDB相关路径。

- -?, –help

  显示帮助信息。

- -V, –version

  显示版本号信息。

示例

使用数据库用户执行gs_uninstall脚本进行卸载VexDB。

```
gs_uninstall --delete-data
Checking unInstallationGuide.
Successfully checked unInstallationGuide.
Stopping the cluster.
Successfully stopped the cluster.
Successfully deleted instances.
Uninstalling application.
Successfully uninstalled application.
UnInstallationGuide succeeded.
```

相关命令

gs_install、gs_postuninstall

has_ctl

功能描述

has_ctl是VexDB提供的在高可用场景下控制数据库实例服务的工具。该工具主要供OM调用，及数据库实例服务自恢复时使用。

has_ctl的主要功能有：

- 启动数据库实例服务、AZ的所有实例、单个主机上的所有实例或单独启动某个实例进程。
- 停止数据库实例服务、AZ的所有实例、单个主机上的所有实例或单独停止某个节点实例进程。
- 重启逻辑数据库实例服务。
- 查询数据库实例状态或者单个主机的状态。
- 切换主备实例或重置实例状态。
- 重建备机。
- 查看数据库实例配置文件。
- 设置日志级别，一主多备数据库实例部署下has_server的仲裁模式、AZ之间的切换模式。
- 获取日志级别，一主多备数据库实例部署下has_server的仲裁模式、AZ之间的切换模式。
- 检测实例进程状态。
- 支持一主多备数据库集群切换数据库主备实例。

与has_ctl工具相关的文件：

- cluster_manual_start

该文件是数据库实例启停标志文件。文件位于$GAUSSHOME/bin下。其中，GAUSSHOME为环境变量。启动数据库实例时，has_ctl会删除该文件；停止数据库实例时，has_ctl会生成该文件，并向文件写入停止模式。

- instance_manual_start_X（X是实例编号）

  该文件是单个实例启停标志文件。文件位于$GAUSSHOME/bin下。其中，GAUSSHOME为环境变量。启动实例时，has_ctl会删除该文件；停止实例时，has_ctl会生成该文件，并向文件写入停止模式。

has_ctl的相关约束：

在集群模式下，使用has_ctl集群工具来切换数据库角色，而不是vb_ctl数据库工具。

语法格式

- start：一主多备数据库部署模式下启动数据库实例服务、单个主机上的所有实例或单独启动某个节点实例进程，或者直接启动整个AZ。

```
has_ctl start [-z AVAILABILITY_ZONE [--cm_arbitration_mode=ARBITRATION_MODE]] | [-n NODEID [-D DATADIR]] [-t SECS]
```

- switchover：一主多备数据库部署模式下切换数据库主备实例。

```
has_ctl switchover [-z AVAILABILITY_ZONE] | [-n NODEID -D DATADIR [-f]] | [-a] | [-A] [-t SECS]
```

- finishredo：所有备机停止回放，每个分片中选择一个强制升主。

```
has_ctl finishredo
```

> **说明**
>
> 该参数属于高风险操作，请谨慎执行。

- build：重建备实例。

```
has_ctl build [-c] [-n NODEID] [-D DATADIR [-t SECS] [-f] [-b full] [-j NUM]]
```

- check：检测实例进程运行状态，用户无需关注，不建议使用。

```
has_ctl check -B BINNAME -T DATAPATH
```

- stop：一主多备数据库部署模式下停止数据库实例服务、单个主机上的所有实例或单独停止某个节点实例进程。

```
has_ctl stop [[-z AVAILABILITY_ZONE] | [-n NODEID [-D DATADIR]]] [-t SECS] [-m SHUTDOWN-MODE]
```

- query：一主多备数据库部署模式下查询数据库实例状态或者单个主机的状态。

```
has_ctl query [-z ALL] [-l FILENAME] [-v [-C [-s] [-S] [-d] [-i] [-F] [-x] [-p]] | [-r]] [-t SECS] [--minorityAz=AZ_NAME]
```

- view：查看数据库实例配置文件。

```
has_ctl view [-v | -N | -n NODEID] [-l FILENAME]
```

- set：设置日志级别，一主多备数据库部署模式下has_server的仲裁模式、AZ之间的切换模式、has_server升主模式。

```
has_ctl set [--log_level=LOG_LEVEL] [--cm_arbitration_mode=ARBITRATION_MODE] [--cm_switchover_az_mode=SWITCHOVER_AZ_MODE] [--cmsPromoteMode=CMS_PROMOTE_MODE -I INSTANCEID]
```

- set –paramver：设置HAS参数，默认set所有节点上的参数，也可以通过-n参数指定set某个节点。

```
has_ctl set --param --agent | --server [-n [NODEID]] -k [PARAMETER]="[value]"
```

- get：获取日志级别，一主多备数据库部署模式下has_server的仲裁模式、AZ之间的切换模式。

```
has_ctl get [--log_level] [--cm_arbitration_mode] [--cm_switchover_az_mode]
```

- setrunmode：DCF部署方式下，设置DCF投票数，主要用于DCF强启。

```
has_ctl setrunmode -n NODEID -D DATADIR  [[--xmode=normal] | [--xmode=minority --votenum=NUM]]
```

- changerole：DCF模式下，将角色为primary的修改为passive或者follower。

```
has_ctl changerole [--role=PASSIVE | --role=FOLLOWER] -n NODEID -D DATADIR [-t SECS]
```

- changemember：DCF模式下，改变指定DCF节点属性，包括节点角色、节点所在的逻辑组、节点的选举优先级等。

```
has_ctl changemember [--role=PASSIVE | --role=FOLLOWER] [--group=xx] [--priority=xx] -n NODEID -D DATADIR [-t SECS]
```

- reload：在线加载数据库实例静态配置文件，用户无需关注，不建议使用。

  reload –param：加载可以动态生效的HAS参数，部分参数不支持reload，只能重启HAS才能生效。

```
has_ctl  reload --param [--agent | --server]
```

- list：列出has_agent或has_server所有的参数。

```
has_ctl  list --param --agent | --server
```

- encrypt：对输入的密码进行加密操作，密码支持8~15位，且必须包含三种字符（数字，字母，符号）。

```
has_ctl   encrypt [-M MODE] -D DATADIR
```

- ddb：DCC或者share disk模式下，执行对DCC组件或者share disk的配置数据存取命令。HAS依赖DCC组件或者share disk对配置数据分布式存取，实现集群配置管理高可用能力。

```
has_ctl ddb DCC_CMD
```

- switch：执行ddb模式的切换。

```
has_ctl switch DCC_CMD
```

参数说明

has_ctl参数可分为如下几类：

> **说明**
>
> 此处列出的公共参数并不一定适用于所有命令，而是多个命令支持，为避免冗余信息，所以统一在此说明，详细的使用方法见以上使用方法，也可以使用 `has_ctl --help` 进行查询。

公共参数

参数	参数说明
-D DATADIR	指定实例数据目录。仅用于对数据库节点进行操作的命令，如 start、stop、switchover、build、setrunmode、changerole、changemember、encrypt。
-l FILENAME	查询结果输出到指定文件。仅用于查询类的命令，如 query、view。
-n NODEID	指定节点。
-z AVAILABILITY_ZONE	指定 AZ 名称。
-t SECS	指定超时时间。超时后，会退出并报错。
-V, –version	打印 has_ctl 版本信息，然后退出。
-?, -h,–help	显示关于 has_ctl 命令行参数的帮助信息，然后退出。

start模式参数

参数	参数说明
–cm_arbitration_mode=ARBITRATION_MODE	一主多备功能，获取 has_server 的仲裁模式。共有 MAJORITY、MINORITY 两种模式，MAJORITY 为多数派模式，MINORITY 为少数派模式。少数派模式适用于一主多备数据库部署并且只有 AZ3存活时，此时 has_server 可以进行正常的仲裁业务，非此模式下将仲裁模式设置成少数派成功后，HAS 会自动将仲裁模式改为多数派，以保证集群正常运转；多数派模式适用于一主多备数据库部署并且各个组件（has_server，节点）存活数量大于一半的场景。数据库实例正常情况下默认为多数派模式。
-I	指定需要启动的资源实例号，可以通过 `has_ctl query -Cv` 查看资源实例和实例号的映射关系。

switchover参数

参数	参数说明
-a	将集群的主机重置到初始配置的节点。
-A	将主机切换到一个合适的备机节点上。
-f	指定进行-f 类型 switchover。不等待客户端中断连接，所有活跃事务都被回滚并且客户端都被强制断开，然后服务器将被切换，且不做 checkpoint。 使用方式： `has_ctl switchover -n NODEID -D DATADIR -f` 。
-z	将主机切换到-z 参数指定的 AZ。 注意，switchover 为维护操作：确保数据库实例状态正常，所有业务结束，并使用 `pg_get_senders_catchup_time()` 视图查询无主备追赶后，再进行 switchover 操作。

build参数

参数	参数说明
-f	强制重建备机。
-b full	指定进行全量 build。不指定情况下，对于一主多备数据库实例部署模式进行 auto build。auto build 指：先调用增量 build，失败之后调用全量 build。
-c	重建 has_server（将主节点的 DCC 数据目录拷贝到指定节点，只适用于一主一备模式）。

check参数

参数	参数说明
-B BINNAME	指定进程名，其进程名包括“cm_agent”，“vexdb”和“cm_server”。
-T DATAPATH	指定实例数据目录。

stop参数

参数	参数说明
-m SHUTDOWN-MODE	指定停止模式，停止模式有以下几种： • smart（s）：等待用户业务结束后，停止所有数据库实例。 • fast（f）：不等待用户业务结束，指定数据库实例退出。 • immediate（i）：不等待用户业务结束，指定数据库实例强制退出。
-I	指定需要停止的资源实例号，可以通过 `has_ctl query -Cv` 查看资源实例和实例号的映射关系。

query参数

参数	参数说明
-s	显示导致各个节点主备实例数量不均衡的实例。 说明：-s 参数需要和-v 以及-C 参数一起使用才能按主备关系成对显示导致各个节点主备实例数量不均衡的实例，使用-s 参数时，必须指定-C、-v 参数。
-C	按主备关系成对显示数据库实例状态。 说明：-C 参数需要和-v 参数一起使用才能按主备关系成对显示数据库实例详细状态信息，使用-C 时，必须指定-v 参数。
-v	显示详细数据库实例状态。 数据库实例有如下几种状态： • Normal：表示数据库实例可用，且数据有冗余备份。所有进程都在运行，主备关系正常。 • Degraded：表示数据库实例可用，但数据没有冗余备份。 • Unavailable：表示数据库实例不可用。
-d	显示实例数据目录。 说明：-d 参数需要和-v 以及-C 参数一起使用。
-i	显示物理节点 ip。 说明：-i 参数需要和-v 以及-C 参数一起使用。
-F	显示各个节点 Fenced UDF 状态。 说明：-F 参数需要和-v 以及-C 参数一起使用才能显示各个节点 Fenced UDF 状态，使用-F 参数时，必须指定-C、-v 参数。
-z ALL	显示数据库实例所在 AZ 名称。 说明：-z 参数需要和-v 以及-C 参数一起使用，并且-z 后面需要跟参数 ALL。
-r	显示备机 redo 状态。 说明：使用-r 时，必须指定-v 参数。
-g	显示备份和恢复群集信息。
-x	显示所有异常的数据库实例。 说明：-x 参数需要和-v 以及-C 参数一起使用。
-S	显示数据库实例启动时的状态检查结果。 说明：-S 参数需要和-v 以及-C 参数一起使用才能按显示数据库实例的状态检查结果。有以下三种状态： • Normal：表示数据库实例可用，且数据有冗余备份。所有进程都在运行，主备关系正常。 • Degraded：表示数据库实例可用，但数据没有冗余备份。 • Unavailable：表示数据库实例不可用。
–minorityAz	只查询指定 AZ 的 has_server。 说明：此参数会忽略非指定 AZ 的 has_server 节点，可以在少数派场景提高查询速度。
-p	显示数据库实例所有节点端口。 说明：-p 参数需要和-v 以及-C 参数一起使用。

view参数

参数	参数说明
-v	显示数据库实例所有节点的静态配置详细信息。
-N	只显示本节点的静态配置信息，即执行has_ctl view命令的节点的信息，N 表示 Native。

set参数

参数	参数说明
–log_level=LOG_LEVEL	设置主 has_server 日志级别。共分为 DEBUG5、DEBUG1、WARNING、LOG、ERROR 和 FATAL 六个级别，日志打印信息级别越来越高。设置日志级别越高，输出日志信息就越少。
–cm_arbitration_mode=ARBITRATION_MODE	一主多备功能，设置 has_server 的仲裁模式。共有 MAJORITY、MINORITY 两种模式，MAJORITY 为多数派模式，MINORITY 为少数派模式。VexDB 不支持少数派，此参数可以设置成 MINORITY，但不会生效。
–cm_switchover_az_mode=SWITCHOVER_AZ_MODE	一主多备功能，设置 AZ 之间的自动切换开关。共有 NON_AUTO、AUTO 两种模式，NON_AUTO 为非自动切换模式，AUTO 为自动切换模式。AUTO 模式由主 has_server 自动控制 AZ1和 AZ2之间的节点实例切换。
–cmsPromoteMode=CMS_PROMOTE_MODE -I INSTANCEID	设置 has_server 的升主方式，共有 AUTO、PRIMARY_F 两种模式，AUTO 为默认自选主方式，PRIMARY_F 会强制使-I 指定的节点升主，无论当前是否有主，因此，有可能会导致 has_server 存在多主情况。
–param	表明需要设置 HAS 参数，不带此参数则不能执行设置 HAS 参数。
–agent | –server	此参数为必带参数，表明当前需要设置 has_server 还是 has_agent 的参数。
-k parameter="value"	指定设置的参数和参数的值，只能设置已经存在的参数，不支持增加或删减参数。

get参数

参数	参数说明
–log_level=LOG_LEVEL	获取主 has_server 日志级别。共分为 DEBUG5、DEBUG1、WARNING、LOG、ERROR 和 FATAL 六个级别，日志打印信息级别越来越高。设置日志级别越高，输出日志信息就越少。
–cm_arbitration_mode=ARBITRATION_MODE	一主多备功能，获取 has_server 的仲裁模式。共有 MAJORITY、MINORITY 两种模式，MAJORITY 为多数派模式，MINORITY 为少数派模式。少数派模式适用于一主多备数据库部署并且只有 AZ3存活时，此时 has_server 可以进行正常的仲裁业务，非此模式下将仲裁模式设置成少数派成功后，HAS 会自动将仲裁模式改为多数派，以保证集群正常运转；多数派模式适用于一主多备数据库部署并且各个组件（has_server，节点）存活数量大于一半的场景。数据库实例正常情况下默认为多数派模式。
–cm_switchover_az_mode=SWITCHOVER_AZ_MODE	一主多备功能，获取 AZ 之间的自动切换开关。共有 NON_AUTO、AUTO 两种模式，NON_AUTO 为非自动切换模式，AUTO 为自动切换模式。AUTO 模式由主 has_server 自动控制 AZ1和 AZ2之间的节点实例切换。

setrunmode参数

参数	参数说明	取值范围
–xmode	指定 DCF 的运行模式。	• normal：正常模式。 • minority：少数派模式，需要–votenum 指定投票数。
–votenum	指定 DCF 少数派运行方式的投票数。	正整数，不高于 DCF 总副本数。

changerole参数

参数	参数说明	取值范围
–role	DCF模式下，将角色为primary的修改为passive或者follower。	• passive：passive角色。 • follower：follow角色。

changemember参数

参数	参数说明	取值范围
–role	DCF 模式下，将角色为 primary 的修改为 passive 或者 follower。	• passive：passive 角色。 • follower：follow 角色。
–group	DCF 模式下，修改 group 的值。	0~2147483647
–priority	DCF 模式下，修改 priority 的值。	0~2147483647

reload参数

参数	参数说明
–param	表明需要加载 HAS 参数，不带此参数则不能执行加载 HAS 参数。
–agent | –server	表明当前需要动态加载 has_server 还是 has_agent 的参数。

list参数

参数	参数说明
–param	此参数为必带参数，表明需要列出 HAS 参数信息。
–agent | –server	此参数为必带参数，表明当前需要查看 has_server 还是 has_agent 的参数。

encrypt参数

参数	参数说明
-M	指定加密类型，支持 server、client。默认类型为 server。
-D	指定生成的加密密码文件路径。

ddb参数

参数	参数说明
–put [key] [value]	往 DCC 或者 share disk 中插入键值对，如果键值对已存在则会修改键 key 所对应的值 value。
–get [key]	查询 DCC 或者 share disk 中 key 对应的 value。
–delete [key]	删除 DCC 或者 share disk 中指定的键值对。
–prefix	get 或者 delete 后添加 prefix 参数，可以实现模糊匹配查询和删除。
–cluster_info	DCC 部署模式下获取数据库实例信息，share disk 部署模式下不支持该命令。
–leader_info	DCC 部署模式下获取主节点信息，share disk 部署模式下不支持该命令。
–help，-h	显示 DDB 命令帮助信息。
–version，-v	DCC 部署模式下显示 DCC 版本信息，share disk 部署模式下不支持该命令。

switch参数

参数	参数说明
-M–ddb_type=[DDB]	选择需要切换 DDB 模式（VexDB 只支持 DCC 模式）。
–commit	执行切换会使数据库实例无法选主，需要执行 commit 恢复数据库实例。
–rollback	回滚操作，执行切换失败需要执行回滚操作。

res参数

参数	参数说明
–add –res_name=[name] –res_attr=[res_info]	新增资源配置，name为新增资源的名称，res_info为新增资源的信息。
–del –res_name=[name]	删除资源配置，name为删除资源的名称。
–edit –res_name=[name] –res_attr=[res_info]	修改资源配置，name为被修改资源的名称，res_info为新修改的资源信息。
–edit –res_name=[name] –add_inst=[inst_info]	新增资源实例，name为新增实例的资源名称，inst_info为新增实例的信息。
–edit –res_name=[name] –del_inst=[inst_info]	删除资源实例，name为删除实例的资源名称，inst_info为删除实例的信息。
–check	检查资源配置。不合理处会显示Warning，配置不可用才会报错。

> **说明**
>
> 在部署有HAS工具的情况下，对于某些既可以直接调用内核工具，也可以调用HAS工具进行执行的命令，如：switchover、build等，请优先使用HAS工具，因为如果直接调用内核工具，有可能HAS感知不到用户正在手动执行指令，进而误判集群状态异常。

命令参考

- 启动实例。

```
has_ctl start [-z AVAILABILITY_ZONE [--cm_arbitration_mode=ARBITRATION_MODE]] | [-n NODEID [-D DATADIR]] [-t SECS]
```

- 数据库主备切换。

```
has_ctl switchover [-z AVAILABILITY_ZONE] | [-n NODEID -D DATADIR [-f]] | [-a] | [-A] [-t SECS]
```

- 所有备机停止回放，每个分片中选择一个强制升主。

```
has_ctl finishredo
```

- 重建备节点。

```
has_ctl build -n NODEID -D DATADIR [-t SECS] [-f] [-b full]
```

- 检测实例进程运行状态。

```
has_ctl check -B BINNAME -T DATAPATH
```

- 停止实例。

```
has_ctl stop [[-z AVAILABILITY_ZONE] | [-n NODEID [-D DATADIR [-R]]]] [-t SECS] [-m SHUTDOWN-MODE]
```

- 查询集群状态。

```
has_ctl query [-z ALL] [-l FILENAME] [-v [-C [-s] [-S] [-d] [-i] [-F] [-x] [-p]] | [-r]] [-t SECS] [--minorityAz=AZ_NAME]
```

- 查看集群配置文件。

```
has_ctl view [-v | -N | -n NODEID] [-l FILENAME]
```

- 设置参数。

```
has_ctl set [--log_level=LOG_LEVEL] [--cm_arbitration_mode=ARBITRATION_MODE] [--cm_switchover_az_mode=SWITCHOVER_AZ_MODE]
```

- 设置HAS参数。

```
has_ctl set --param --agent | --server [-n NODEID] -k "PARAMETER='value'"
```

- 获取参数。

```
has_ctl get [--log_level] [--cm_arbitration_mode] [--cm_switchover_az_mode]
```

- 设置DCF投票数。

```
has_ctl  setrunmode -n NODEID -D DATADIR  [[--xmode=normal] | [--xmode=minority --votenum=NUM]]
```

- 改变DCF角色信息。

```
has_ctl changerole [--role=PASSIVE | --role=FOLLOWER] -n NODEID -D DATADIR [-t SECS]
```

- 改变DCF节点属性。

```
has_ctl changemember [--role=PASSIVE | --role=FOLLOWER] [--group=xx] [--priority=xx] -n NODEID -D DATADIR [-t SECS]
```

- 动态加载HAS参数。

```
has_ctl reload --param [--agent | --server]
```

- 列出所有HAS参数。

```
has_ctl list --param [--agent | --server]
```

- 加密。

```
has_ctl  encrypt [-M MODE] -D DATADIR
```

- 执行DDB命令行。

```
has_ctl ddb DDB_CMD
has_ctl ddb --put [key] [value]  --设置
has_ctl ddb --delete [key]  --删除
has_ctl ddb --help  --查看DDB命令帮助信息
```

- 执行switch ddb命令。

```
has_ctl switch [--ddb_type=[DDB]] [--commit] [--rollback]
```

- 执行res命令。

```
has_ctl res --add --res_name=[name] --res_attr=[res_info] --新增资源
has_ctl res --del --res_name=[name]  --删除资源
has_ctl res --edit --res_name=[name] --res_attr=[res_info]  --修改资源
has_ctl res --edit --res_name=[name] --add_inst=[inst_info] --新增资源实例
has_ctl res --edit --res_name=[name] --del_inst=[inst_info] --删除资源实例
has_ctl res --check  --检查资源
```

vb_basebackup

VexDB部署成功后，在数据库运行的过程中，可能会遇到各种问题及异常状态。VexDB提供了vb_basebackup工具做基础的物理备份。该工具可以将整个实例的数据进行备份。

vb_basebackup的实现目标是对服务器数据库文件的二进制进行拷贝，其实现原理使用了复制协议。远程执行vb_basebackup时，需要使用系统管理员账户。vb_basebackup当前支持热备份模式和压缩格式备份模式。

关于vb_basebackup的详细用法请参考vb_basebackup。

vb_ctl

功能描述

vb_ctl是VexDB提供的数据库服务控制工具，可以用来启停数据库服务和查询数据库状态。

vb_ctl工具由数据库安装用户执行，具有以下功能：

- 启动、停止、重启VexDB节点。
- 在不停止数据库的情况下，重新加载配置文件（postgresql.conf，pg_hba.conf）。
- 主备切换、主备状态查询、重建和重建状态查询。

工具运行产生的日志位于 `$GAUSSLOG/bin/gs_ctl` 目录下，每个日志最大 16 MB，最多保留 50 个。如果没有配置环境变量 $GAUSSLOG，则不会产生日志文件。

语法格式

- 初始化数据库。

  ```
  vb_ctl init[db] [-D DATADIR] [-s] [-o "--nodename=NODENAME --OPTIONS=VALUES"]
  ```

  运行 `vb_initdb --help` 命令去查看支持的OPTIONS。详细请参考vb_initdb。

- 启动/停止/重启数据库。

  ```
  vb_ctl start   [-w] [-t SECS] [-D DATADIR] [-s] [-l FILENAME] [-o "OPTIONS"] [-M SERVERMODE]
  vb_ctl stop    [-W] [-t SECS] [-D DATADIR] [-s] [-m SHUTDOWN-MODE]
  vb_ctl restart [-w] [-t SECS] [-D DATADIR] [-s] [-m SHUTDOWN-MODE] [-o "OPTIONS"]
  ```

- 在主机上重建备实例或级联备实例。

  ```
  vb_ctl build   [-D DATADIR] [-b MODE] [-r SECS] [-C CONNECTOR] [-q] [-M SERVERMODE]
  ```

- 重载配置文件（postgresql.conf, pg_hba.conf）。

  ```
  vb_ctl reload  [-D DATADIR] [-s]
  ```

- 显示数据库运行状态。

  ```
  vb_ctl status  [-D DATADIR]
  ```

数据库包括以下三种运行状态：

- server is running (PID: xxx)：VexDB服务已启动。
- no server running：VexDB服务已关闭。
- single-user server is running (PID: xxx)：VexDB数据库支持通过single单用户模式登录。

- 主机故障时，备机停止回放，强制升主。

```
vb_ctl finishredo [-D DATADIR] [-s]
```

- 在主机异常时，将备机切换为主机。

```
vb_ctl failover [-W] [-t SECS] [-D DATADIR] [-U USERNAME] [-P PASSWORD] [-T TERM]
```

- 在主备机正常时，出于维护的需要，将备机切换为主机，可保证切换过程中数据不丢失。

> **说明**
>
> - 必须在要切换为主机的备机上执行switchover命令才会生效。在主机上执行switchover命令，仅作查询使用。
> - switchover命令下发后，命令如果超时返回，后台进程的执行状态可能处于不可确定状态。

```
vb_ctl switchover  [-W] [-D DATADIR] [-m SWITCHOVER-MODE] [-U USERNAME] [-P PASSWORD] [-f]
```

- 查询主备机之间的状态信息。

```
vb_ctl query   [-D DATADIR] [-U USERNAME] [-P PASSWORD] [-L lsn]
```

- 启动后再指定主备机。

```
vb_ctl notify  -M SERVERMODE [-D DATADIR] [-U USERNAME] [-P PASSWORD]
```

- 给指定的进程发送信号量。允许用于kill的信号名称包括：ABRT、HUP、INT、QUIT、TERM、USR1、USR2。

```
vb_ctl kill  SIGNALNAME PID
```

- 查看数据库的重建进度。

```
vb_ctl querybuild   [-D DATADIR]
```

- 基于共享存储的双中心容灾模式，进行xlog日志拷贝。

```
vb_ctl copy   [-D DATADIR] [-Q COPYMODE]
```

- 获取VexDB的调用栈。

```
vb_ctl stack [-D DATADIR] [-I lwtid]
```

- 统计数据库系统的历史使用情况。

  VexDB为了方便用户查阅数据库历史使用情况，提供了vb_ctl statistic功能，可以通过参数控制查询的时间段，最终输出一个HTML文件，HTML文件以图形化方式展示出数据库历史使用情况。

```
vb_ctl statistic [-D DATADIR] [-p OUTPUT_PATH] {[-d DAYS] | [-m MONTHS]| [-r STARTDATE,STOPDATE]}
```

  使用该功能需要开启GUC参数enable_wdr_snapshot，否则生成的HTML文件内容为空。

- 集群规模在线调整。

```
vb_ctl member [-O OPERATION] [-u DCF-NODE-ID] [-i DCF-NODE-IP] [-e DCF-NODE-PORT] [-D PRIMARY-DATADIR]
```

- DCF模式下，将角色为primary的修改为passive或者follower。

```
vb_ctl changerole [-R DCF-ROLE] [-D DATADIR] [-t SEC]
```

- 集群支持少数派强起功能。

```
vb_ctl setrunmode   [-x XMODE] [-v VOTE-NUM] [-D PRIMARY-DATADIR]
```

参数说明

vb_ctl参数可分为如下几类：

- 公共参数，详细请参见表1. 公共参数。
- start和restart模式的参数，详细请参见表2. start，stop或restart相关参数。
- xlog copy参数，详细请参见表4. xlog copy参数。
- DCF参数，详细请参见表5. DCF参数。

- stack参数，详细请参见表6. stack参数。
- switchover参数，详细请参见表7. switchover参数。
- 资源池化build参数，详细请参见表8. 资源池化build参数。
- query参数，详细请参见表9. query参数。
- 系统使用情况statistic参数，详细请参见表10. 系统使用情况statistic参数。

表1 公共参数

参数	说明	取值范围
-b, –mode=MODE	指定重建备机的模式。	mode 的取值如下： • full：通过全量镜像的方式重新同步主机的数据目录。 • incremental：通过解析 Xlog 日志获取主备机差异的数据进行增量修复备机。 说明： 增量重建适用于主备双主等因日志造成的不一致场景。 增量重建不适用于一主一备并且没有开启最大高可用的场景，此种场景下需要使用全量重建或者开启最大高可用后再进行增量重建。 备机数据文件损坏、数据目录丢失等故障通过增量重建的方式无法修复，此时可通过全量重建的方式重新修复备机。 • auto（不指定）：先增量，根据失败后是否可以再增量选择继续增量或 者全量，三次增量失败后进行全量。 • standby_full：通过指定的备机 ip 和 port 全量重建故障备机。使用该参数时需要同时使用-C 参数指定镜像的 ip 和 port。 • copy_sec dure_files：在流式容灾场景下，获取指定节点数据目录下 gs_secure_files 目录对应内容。 • copy_upgrade_file：在 dorado 容灾场景下，获取指定节点下升级所需指定文件。 • cross_cluster_full：在 dorado 容灾场景下，通过全量镜像的方式重新同步主机的数据目录。 • cross_cluster_standby_full：在 dorado 容灾场景下，通过全量镜像的方式重新同步备机的数据目录。 • cross_cluster_incremental：基于共享存储的同城双中心场景下的跨集群增量 build。 • check：检测是否需要 build，如果需要，返回 build 的类型。 默认值：auto 说明 重建级联备机需要加上-M cascade_standby 参数。
-D, –pgdata=DATADIR	指定数据目录的位置。若指定目录中包含配置文件 postgresql.conf，且配置文件内 data_directory 值与-D指定目录不同，将优先按 data_directory 目录执行。 如果省略了-D选项，将使用环境变量 PGDATA。	DATADIR 的取值必须为有效的数据目录。
-s, –silent	对 reload、restart、stop 命令仅打印部分提示信息，对其他命令不生效。 打印信息如：server promoting、 server shutting down。	-

	不打印信息如：waiting for server to shut down、server stopped 等提示信息。	
-t, –timeout=SECS	等待数据库启动、关闭或者主备切换完成的最大秒数。如果等待超时，命令会结束退出，并通知不再等待。	取值范围：整型（秒）。 默认值：等待数据库启动、停止或者主备切换完成：60秒。
-V , –version	打印 vb_ctl 的版本信息然后退出。	-
-w	使用 -w 参数可使 vb_ctl 命令在执行数据库启动、停止或重启操作时等待操作完全完成才返回结果。此处提到的等待操作指的是启停数据库本身需要的操作。以下是在不同操作场景下 -w参数的具体情况： • 启动：vb_ctl 会等待数据库实例完成所有启动步骤，包括读取配置文件、初始化内存结构、打开数据文件等，并进入正常服务状态再返回。 • 关闭：vb_ctl 会等待数据库实例完成所有关闭步骤，如结束活动会话、将内存数据刷新至磁盘及释放系统资源，直到数据库进程彻底退出再返回。 • 重启：vb_ctl 会等待数据库实例先停止，再重新启动并进入正常服务状态。整个过程涵盖了停止数据库的操作和启动数据库的操作，-w 会保证这两个阶段的操作都完成后才结束等待返回结果。	默认值：省略此参数时，默认会等待操作完成。
-W	-W 参数与 -w 参数相反，它表示不等待操作完成。	-
-M	-M 后面需要跟 SERVERMODE 参数，表示在启动时指定数据库的启动模式。 说明： 当前主机处于一主零备，或单机环境时，不支持-M 参数。	SERVERMODE 的取值范围： • primary：本端以主机模式启动。 • standby：本端以备机模式启动。 • cascade_standby：本端以级联备机模式启动。 • pending：本端处于等待状态，等待提升为主机或者备机。
-T	-T 后面跟 term，升主命令时会用此 term 作为主机 term，build 时会连接大于等于此 term 的主机进行 build	term 的取值范围，无符号整型。
-q	重建结束后，是否自动重启。 指定该参数，则不会重启 未指定场合，自动重启，重启默认最大等待时间是60秒。如果在此时间内数据库未能启动成功，就会报不再等待，命令退出。	无参数
-d	打印更多调试信息。	无参数
-L	查询 lsn 并展示最大长度。	-
-P PASSWORD	与-U 参数配合使用，指定连接本地数据库的用户密码。 说明： 当省略-U 参数，且认证方法为 trust 时，-P 参数无效。	-
-U USERNAME	指定连接数据库的用户。此参数只能与 notify、query、failover、switchover 和 querybuild 参数配合使用。 对于域用户，使用 DOMAIN 格式。	取值范围：VexDB 中存在的用户。 默认值：省略此参数则使用与当前操作系统用户同名的用户。
-Z NODE-TYPE	节点类型	-
-?, -h, –help	显示关于 vb_ctl 命令行参数的帮助信息。	-

表2 start，stop或restart相关参数

参数	说明	取值范围
-c, –core-file	允许 VexDB 线程产生内核文件。	-
-l, –log=FILENAME	把服务器日志输出附加在 FILENAME 文件上。	FILENAME 的取值为启动数据库服务的用户有权限的文件。 例如，data 目录下的某文件。
-o "OPTIONS"	声明要直接传递给由 VexDB 执行的 VexDB 的命令行选项。 参数通常都用单或者双引号包围以保证它们作为一个整体传递。	VexDB 支持的参数。
-p PATH-TO-POSTGRES	声明 VexDB 可执行文件的位置。	默认值：vb_ctl 自身所在目录。
-m, –mode=MODE	声明关闭模式。	mode 的取值如下： • fast：不等待客户端中断连接，所有活跃事务都被回滚并且客户端都被强制断开，然后服务器将被关闭。 • smart：单机模式下主机立即断开，同 fast。 • immediate：强行关闭，在下次重新启动的时候将导致故障恢复。 默认值：fast

表3 init参数

参数	说明	取值范围
–nodename=NODENAME	初始化的节点名称。	字符串，节点的命名需要遵守如下规范： • 节点名称必须为小写字母（a-z）、下划线（_）、特殊符号#、数字（0-9）。 • 节点名称必须以小写字母（a-z）或下划线（_）开头。 • 节点名称不能为空，且最大的长度为64个字符。

表4 xlog copy参数

参数	说明	取值范围
-Q, –mode=MODE	指定xlog日志拷贝方向。	• copy_from_local：从本地往共享存储上拷贝。 • copy_from_share：从共享存储往本地存储上拷贝。 • force_copy_from_local：强制从本地往共享存储上拷贝。

表5 DCF参数

参数	说明	取值范围
-O, –operation=OPERATION	VexDB 数据库实例，DCF 模式下，增删节点的操作。	• add： 向数据库实例的 DCF 节点配置中增加一个节点 • remove：从数据库实例 DCF 节点配置中删除一个节点。 • change：从数据库实例 DCF 节点配置中更改一个节点。
-u, –nodeid=DCF-NODE-ID	新增节点的 DCF NODE ID。	无符号整型。
-i, –ip=DCF-NODE-IP	新增节点与数据库实例 DCF 模块通信的 IP。	字符串。
-e, –port=DCF-NODE-PORT	新增节点与数据库实例 DCF 模块通信的 port。	整型。
-R, –role=DCF-NODE-ROLE	VexDB 数据库实例，DCF 模式下节点角色（需要在角色为 primary 的节点上使用）。	• passive：passive 角色（passive 节点不参与选举，只做日志的同步以及回放，该类型节点在高负载的情况下，日志同步会做流控）。 • follower：follower 角色（响应 Leader 的日志同步请求）。
-v, –votenum=VOTE-NUM	投票个数。	无符号整型。
-x, –xmode=XMODE	运行模式（在少数派的 DN 节点上使用）。	• minority：少于一半节点的响应时可以成为 Leader（少数派强起）。 • normal：获得超过一半节点的响应时可以成为 Leader。
-G	DCF 模式下，group 的值。	0~2147483647
–priority	DCF 模式下，节点选举的优先级。	0~2147483647

表6 stack参数

参数	说明	取值范围
-I lwtid	指用于指定需要获取调用栈的线程的lwtid。	正整数

表7 switchover参数

参数	说明	取值范围
-f	指定进行-f类型switchover。不等待客户端中断连接，所有活跃事务都被回滚并且客户端都被强制断开，然后服务器将被切换，且不做checkpoint。使用方式： `vb_ctl switchover -n NODEID -D DATADIR -f` 。	无参数
-m SWITCHOVER-MODE	声明双机主备实例正常时切换的切换模式。	mode的取值如下： • fast：不等待客户端中断连接，所有活跃事务都被回滚并且客户端都被强制断开，然后服务器将被切换。 • smart：等待所有客户端断开连接以及任何在线备份结束，正在进行中的事务不中断。 默认值：fast

表8 资源池化build参数

参数	说明	取值范围
-r, –recvtimeout=INTERVAL	指定重建过程中备机日志接收等待主机响应的超时时间。	取值范围：整型（秒）。 默认值：120秒 提示：超时时间需根据业务繁忙程度设置，业务繁忙等待主机的响应时间需加长，建议超时时间=checkpoint的执行时间+1min。
-C, connector	指定连接串。	-
–enable-dss	开启资源池化开关。	无参数
–vgname	卷组名。可选，未指定时由系统自动识别。	数据类型：字符串 。 例如：一个卷组“+data”或者两个卷组中间用“,”连接“+data,+log”。
–socketpath=socketpath	dss实例进程使用的socket文件路径。可选，未指定时由系统自动识别。	实际的文件路径。
-b MODE	指定重建备机的模式。	mode的取值： • cross_cluster_full：资源池化执行build，必须结合dorado容灾场景，该参数表示在dorado容灾场景下通过全量镜像的方式重新同步主机的数据目录。 • check：判断是否需要build。 提示：资源池化build只适用于主备集群间的跨集群build，不能用于资源池化单集群内的主备实例build。

表9 query参数

参数	说明	取值范围
-D DATADIR	指定数据目录的位置，查询数据库状态，显示当前数据库的数据目录以及运行状态等信息。	DATADIR 的取值必须为有效的数据目录
-U USERNAME	查询受密码保护的数据库集群，可以使用-U 选项来指定用户名。	取值范围：VexDB 中存在的用户。 默认值：省略此参数则使用与当前操作系统用户同名的用户。
-P PASSWORD	查询受密码保护的数据库集群，可以使用-P 选项来指定密码，一般与-U 配合使用	-
-L	查询 lsn 并展示最大长度。	-

表10 系统使用情况statistic参数

参数	说明	取值范围
-D DATADIR	指定要查询的数据库实例的路径。	含义同$PGDATA。
-p OUTPUT_PATH	输出的 HTML 文件路径。	文件路径（绝对路径）。
-d DAYS	输出之前 N 天（包含查询当日）的统计情况，结果按照天为单位进行汇总。	[1,180]
-m MONTHS	输出之前 N 月（包含查询当月）的统计情况，结果按照月为单位进行汇总。	[1,36]
-r STARTDATE,STOPDATE	查询置顶日期范围的统计情况，结果按照天为单位进行汇总。	STARTDATE 和 STOPDATE 为 date 类型，时间差不能超过180天。

> **说明**
>
> 时间范围参数[-d DAYS]、[-m MONTHS]、[-r STARTDATE,STOPDATE]三者是互斥的，不能同时指定两个或两个以上参数。

使用示例

启动/停止数据库

- 启动数据库。

```
vb_ctl start
```

- 使用端口5432端口启动数据库，而且不带fsync（强制刷盘）运行。

```
vb_ctl -o "-F -p 5432" start
```

- 关闭数据库（使用-m选项控制数据库以fast方式关闭）。

```
vb_ctl stop -m fast
```

显示数据库状态

```
vb_ctl status
```

返回结果如下：

```
[2023-04-10 16:09:13.131][29655][][vb_ctl]: vb_ctl status,datadir is /home/vexdb/data/vexdb
vb_ctl: server is running (PID: 29381)
/home/vexdb/local/vexdb/bin/vexdb "-F" "-p" "5432"
```

重新加载配置文件

在配置文件中改变参数后，需要使用如下命令使参数生效。

```
vb_ctl reload
```

实例主备切换

将数据库主实例切换为备实例。

1、操作系统用户VexDB登录数据库任意节点，执行如下命令，检查主备节点状态。

```
vb_ctl query
```

2、以操作系统用户vexdb登录需要升主的备节点，执行如下命令进行主备切换。

```
vb_ctl switchover
```

返回结果为：

```
vb_ctl switchover ,datadir is /home/vexdb/data/vexdbswitchover term (1)
waiting for server to switchover.......
switchover completed (/home/vexdb/data/vexdb)
```

3、当主节点故障时，登录备节点，将备节点提升为主节点。

```
vb_ctl failover
```

返回结果如下：

```
[2022-08-02 10:37:53.887][28312][][vb_ctl]: vb_ctl failover ,datadir is /home/vexdb/data/vexdb
[2022-08-02 10:37:53.887][28312][][vb_ctl]: failover term (1)
[2022-08-02 10:37:53.893][28312][][vb_ctl]:  waiting for server to failover...
[2022-08-02 10:37:53.894][28312][][vb_ctl]:  done
[2022-08-02 10:37:53.894][28312][][vb_ctl]:  failover completed (/home/vexdb/data/vexdb)
```

统计数据库使用情况

1、打开GUC参数enable_wdr_snapshot。

```
ALTER SYSTEM SET enable_wdr_snapshot TO on;
```

2、使用vb_ctl工具获取数据库使用情况统计信息。

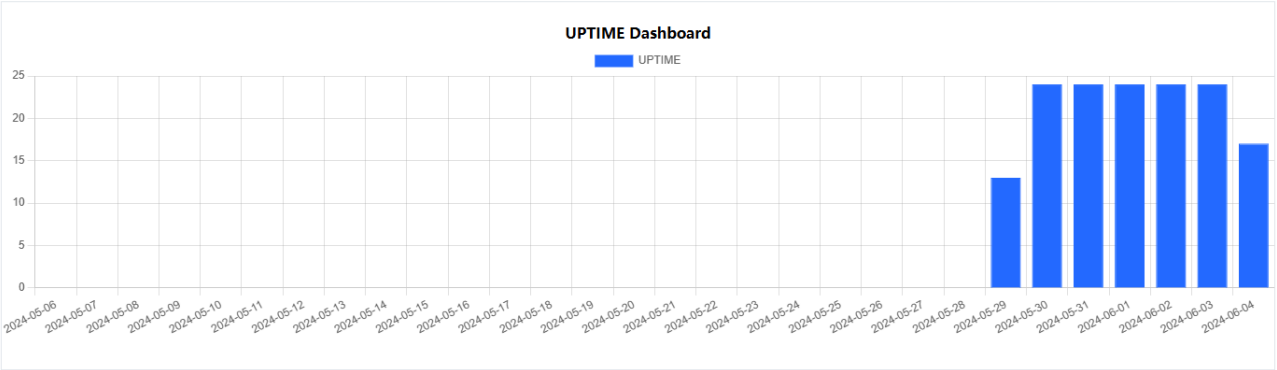
```
vb_ctl statistic -D $PGDATA -p /tmp/mystatistic.html -d 30
```

上述命令执行成功后，会在/tmp路径下生成HTML文件mystatistic.html，用浏览器打开后显示如下内容：

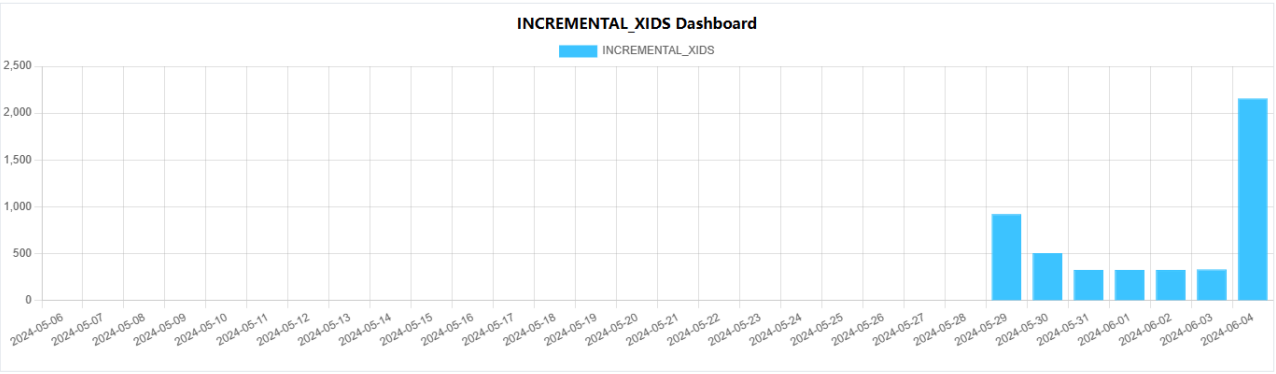
数据库使用情况统计

2024-05-14	0	0
2024-05-15	0	0
2024-05-16	0	0
2024-05-17	0	0
2024-05-18	0	0
2024-05-19	0	0
2024-05-20	0	0
2024-05-21	0	0
2024-05-22	0	0
2024-05-23	0	0
2024-05-24	0	0
2024-05-25	0	0
2024-05-26	0	0
2024-05-27	0	0
2024-05-28	0	0
2024-05-29	13	923
2024-05-30	24	507
2024-05-31	24	328
2024-06-01	24	328
2024-06-02	24	327
2024-06-03	24	332
2024-06-04	17	2157

数据库在线时长统计



数据库事务数统计



vb_dump

vb_dump是VexDB数据库中自带的用于导出数据库相关信息的一种逻辑备份工具，用户可以自定义导出一个数据库或其中的对象（模式、表、视图等），回收站对象除外。支持导出的数据库可以是默认数据库postgres，也可以是自定义数据库。

关于vb_dump的详细介绍请参考vb_dump。

vb_dumpall

vb_dumpall是VexDB用于导出所有数据库相关信息的工具。它可以导出VexDB数据库的所有数据，包括默认数据库postgres的数据、自定义数据库的数据以及VexDB所有数据库公共的全局对象。

vb_dumpall工具支持导出完整一致的数据。例如，T1时刻启动vb_dumpall导出VexDB数据库，那么导出数据的结果将会是T1时刻该VexDB数据库的数据状态，T1时刻之后对VexDB的修改不会被导出。

vb_dumpall在导出VexDB所有数据库时分为两部分：

- vb_dumpall自身对所有数据库公共的全局对象进行导出，包括有关数据库用户和组、表空间以及属性（例如，适用于数据库整体的访问权限）信息。
- vb_dumpall通过调用vb_dump来完成VexDB中各数据库的SQL脚本文件导出，该脚本文件包含将数据库恢复为其保存时的状态所需要的全部SQL语句。

以上两部分导出的结果为纯文本格式的SQL脚本文件，使用vsql运行该脚本文件即可恢复VexDB数据库。

关于vb_dumpall的详细介绍请参考vb_dumpall。

vb_guc

功能描述

应用程序可以通过调用 vb_guc 工具来设置数据库的配置参数。

工具运行产生的日志位于 `$GAUSSLOG/bin/gs_guc` 目录下，每个日志最大 16 MB，最多保留 50 个。如果没有配置环境变量 $GAUSSLOG，则不会产生日志文件。

目前 VexDB配置文件（postgresql.conf、pg_hba.conf）中的参数默认值都是单机的配置模式。

- “server.key.cipher”， “server.key.rand”

  使用vb_guc encrypt命令加密用户密码时，如果指定的-M的值为server，则会生成这两个文件。其中“server.key.cipher”存储用户密码的密文，“server.key.rand”存储的是加密因子。

- “client.key.cipher”， “client.key.rand”

  使用vb_guc encrypt命令加密用户密码时，如果指定的-M的值为client，则会生成这两个文件。其中“client.key.cipher”存储用户密码的密文，“client.key.rand”存储的是加密因子。

  > **说明**
  >
  > “client.key.cipher”和“client.key.rand”是不指定-U选项时生成的密文文件和加密因子文件。如果通过-U指定了用户名，则会生成以用户名开头的文件。以指定-U test为例，生成的文件名为:“test.key.cipher、test.key.rand”。

- “datasource.key.cipher”， “datasource.key.rand”

  使用vb_guc encrypt命令加密用户密码时，如果指定的-M的值为source，则会生成这两个文件。其中“datasource.key.cipher”存储用户密码的密文，“datasource.key.rand”存储的是加密因子。

  > **说明**

- “datasource.key.cipher”和“datasource.key.rand”是创建Data Source对象时调用的密钥文件。vb_guc生成时即有读权限。使用前需将这两个文件放入各节点目录$GAUSSHOME/bin，且确保具有读权限。
- 使用vb_guc generate命令也可以生成这两个文件，可以选择以下两种方式中的任一种，并根据提示输入密码。
  - 方式1：vb_guc encrypt -M source -D ./
  - 方式2：vb_guc generate -o datasource -D ./

注意事项

- vb_guc工具不支持参数值中包含#的设置。可以使用vi工具通过手工修改配置文件来设置。
- 如果已经在环境变量中设置PGDATA，则可以省略-D参数。否则提示设置参数失败。
- 如果设置GUC参数时使用-c “parameter”，则会将已设置的GUC参数值设置成该参数的内核默认值（注意log_directory和audit_directory不会被设置为内核参数默认值，而是设为$PGDATA/pg_audit/instance_name）。由于GUC参数间存在依赖关系，因此请慎用该功能。
- 设置-c参数时，参数都可以省略双引号。
- 如果value中含有特殊字符（如$），请转义后使用。
- 如果同一个配置参数在配置文件里面出现多行，且有两行或多于两行同时生效（即没有用“#”注释掉），那么只有最后一个配置参数会被设置，而前面的都会被忽略。
- 通过reload模式设置或修改VexDB节点配置文件（postgresql.conf）的参数，生效存在短暂延迟，有可能导致配置后VexDB各实例参数极短时间不一致。
- vb_guc设置浮点类型的参数时，由于浮点数在计算机中不能精确表示，所以参数值在误差范围1e-9内都可以设置成功。
- vb_guc设置整型类型的参数时，可以接受十进制、十六进制、八进制的数据，以0x开始表示十六进制，以0开始表示八进制，其他情况表示十进制。
- vb_guc设置string参数，参数中的单引号会计算字符。

语法格式

- 检查配置文件中参数。

```
vb_guc check [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} {-c "parameter", -c "parameter", ...}vb_guc check [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} {-c parameter,
```

- 修改配置文件中的参数。

```
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--lcname=LCNAME] [--ignore-node=NODES] {-c "parameter = value" -c "parameter = value" ...}
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--lcname=LCNAME] [--ignore-node=NODES] {-c "parameter = 'value'" -c "parameter = 'value'" ...}
```

> 说明

- set：表示只修改配置文件中的参数。
- reload：表示修改配置文件中的参数，同时发送信号量给数据库进程，使其重新加载配置文件。

- 将已设置的参数值修改为默认值。

```
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--lcname=LCNAME] [--ignore-node=NODES] {-c "parameter" -c "parameter" ...}
```

- 配置客户端接入认证。若选择reload会同时发送信号量到pg_hba.conf，即无需重启即可生效。

```
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--ignore-node=NODES] -h "HOSTTYPE DATABASE USERNAME IPADDR IPMASK AUTHMEHOD authentication-options"
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--ignore-node=NODES] -h "HOSTTYPE DATABASE USERNAME IPADDR-WITH-IPMASK AUTHMEHOD authentication-options"
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--ignore-node=NODES] -h "HOSTTYPE DATABASE USERNAME HOSTNAME AUTHMEHOD authentication-options"
```

如果身份验证策略需要设置/重新加载DEFAULT或COMMENT，然后在没有身份验证的情况下提供，请使用以下格式：

```
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--ignore-node=NODES] -h "HOSTTYPE DATABASE USERNAME IPADDR IPMASK"
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--ignore-node=NODES] -h "HOSTTYPE DATABASE USERNAME IPADDR-WITH-IPMASK "
vb_guc {set | reload} [-Z NODE-TYPE] [-N NODE-NAME] {-I INSTANCE-NAME | -D DATADIR} [--ignore-node=NODES] -h "HOSTTYPE DATABASE USERNAME HOSTNAME"
```

说明

认证策略中包含一串认证参数：

```
HOSTTYPE DATABASE USERNAME IPADDR IPMASK
HOSTTYPE DATABASE USERNAME IPADDR-WITH-IPMASK
HOSTTYPE DATABASE USERNAME HOSTNAME
```

AUTHMEHOD后的authentication-options为可选参数，AUTHMEHOD支持以下选项：

- trust：不验密，禁止远程主机使用trust方式访问数据库。
- reject：拒绝访问。
- md5：md5认证，默认不支持（MD5加密算法安全性低，存在安全风险，不建议使用）。
- sha256：sha256认证（推荐使用）。
- cert：客户端证书认证。
- gss：kerberos认证，仅用于内部节点间认证。
- sm3：sm3认证（国密SM3）

pg_hba.conf中的认证策略越靠前优先级越高，使用vb_guc工具配置时会按一定规则排序将新策略插入到原有认证策略中。配置字段比较顺序为：IPADDR/HOSTNAME > HOSTTYPE >DATABASE>USERNAME，即优先比较IPADDR或HOSTNAME，如果无法区分优先级则继续比较HOSTTYPE，以此类推。对于每个配置字段，通常越严格的配置参数优先级越高、排序越靠前，越宽松的配置参数优先级越低、排序越靠后，具体如下：

- IPADDR：当配置为全0时表示不限制IP，会放在指定具体某个IP地址的策略后面。
- DATABASE：当配置为all时表示不限制数据库，会放在指定具体某个数据库的策略后面；当数据库配置为replication时会放在其他策略后面。
- USERNAME：当配置为all时表示不限制用户，会放在指定具体某个用户的策略后面。
- HOSTTYPE：local > hostssl > hostnossl > host
- 显示帮助信息。

```
vb_guc -? | --help
```

- 显示版本号信息。

```
vb_guc -V | --version
```

- vb_guc encrypt生成加密密码和加密因子文件。

```
vb_guc encrypt [-M keymode] -K password [-U username] {-D DATADIR | -R RANDFILEDIR -C CIPHERFILEDIR}
```

  - K是用户指定的密码，vb_guc会对该密码进行长度（8<=len<16）和密码复杂度要求，如果不满足，将会报错。此密码用于保证生成密码文件的安全性和唯一性，用户无需保存或记忆。
  - M是加密类型，当前仅支持server、client和source。默认值为server。
- vb_guc generate生成其他前缀的加密密码和加密因子文件。

```
vb_guc generate [-o prefix] -S cipherkey -D DATADIR
```

  - o是输出cipher和rand文件前缀名称，默认输出文件名前缀为obsserver。其内容仅支持数字、字母和下划线。
  - S是用户指定的密码，密码需要满足长度要求（8<=len<16）和复杂度要求，如不满足将会报错。当其值为default时，会随机生成一段字符串作为密码，该密码长度为13。如果不带-S参数则会提示交互式输入密码。为了系统安全，推荐使用交互式输入密码方式。
  - 使用vb_guc encrypt或vb_guc generate命令生成加密密码和加密因子文件时只是参数不同，本质上是一样的。生成过程中会使用随机数作为加密密钥材料和盐值，因此是每次生成的文件都是不同的。每次生成的加密密码和加密因子文件需要成对使用，不能更换或交替使用，加密和解密时需要使用相同的加密密码和加密因子文件。

参数说明

- -N

  需要设置的主机名称。

  取值范围：已有主机名称。

  当参数取值为ALL时，表示设置VexDB中所有的主机。

  暂不支持一次设置中指定多个-N参数，指定多个-N会报错。

- -I

  需要设置的实例名称。

  取值范围：已有实例名称。

  当参数取值为ALL时，表示设置主机中所有的实例。

  暂不支持一次设置中指定多个-I参数，指定多个-I会报错。

- -D, –pgdata=DATADIR

  需要执行命令的VexDB实例路径。使用encrypt命令时，此参数表示指定的密码文件生成的路径。

  该参数不能与-I 一块使用。

- -c parameter=value

  要设定的VexDB配置参数的名称和参数值。

  取值范围：postgresql.conf中的所有参数。

> **说明**
>
> - 如果参数是一个字符串变量，则使用-c parameter= “ ‘value’ ” 或者使用-c “parameter = ‘value’ ” 。
> - 如果需要配置的value内容中包含双引号, 则需要在双引号前加上转义符。例如value为a” b” c, 则命令为-c ” parameter = ‘a"b"c’ “。
> - 当使用vb_guc set/reload为” log_directory” 恢复默认值时，其默认值会被置为具体的data目录。
> - 当使用vb_guc reload进行参数设定，并指定-N参数时，当指定的节点为主节点时，主备节点的参数值都会被修改；当指定节点为备节点时，只会修改备节点的参数值，不会修改主节点的参数值。

> - 当使用vb_guc reload进行参数设定，未指定-N参数时，当在主节点上执行时，主备节点的参数值都会被修改；当在备节点上执行时，只会修改备节点的参数值，不会修改主节点的参数值。

- -c parameter

  当进行check操作时，表示需要检查的参数名称。当进行set/reload操作时，参数值不允许为空，为空时不会恢复为数据库参数的默认值。

- –lcname=LCNAME

  要设定的逻辑数据库名称。

  取值范围：已经创建的逻辑数据库名称。

  该参数必需同-Z datanode一起使用。即vb_guc只允许作用于逻辑数据库的DN实例。

  逻辑数据库允许操作的参数同完整数据库不同。具体差异可参见$GAUSSHOME/bin/cluster_guc.conf。

- –ignore-node=NODES

  需要忽略的主机名称。

  > **说明**
  > - 该参数必须与set/reload一起使用，且-Z只支持datanode。
  > - 该参数不支持与-D一起使用。
  > - 在与reload一起使用时，如果–ignore-node没有指定主节点，则集群中所有节点的参数依然会全部同步修改。
  > - –ignore-node必须在-N all时才可生效。

- -h host-auth-policy

  指定需要在pg_hba.conf增加的客户端认证策略。

  取值范围：

  - HOSTTYPE DATABASE USERNAME IPADDR IPMASK [authmehod-options]
  - HOSTTYPE DATABASE USERNAME IPADDR-WITH-IPMASK [authmehod-options]
  - HOSTTYPE DATABASE USERNAME HOSTNAME [authmehod-options]

  HOSTTYPE是必选参数，取值范围如下：

  - local：Unix域套接字。

- host：普通或SSL加密的TCP/IP套接字。
- hostssl：SSL加密的TCP/IP套接字。
- hostnossl：纯TCP/IP套接字

authmehod-options是可选输入，取值为：trust、reject、md5、sha256、sm3、cert、gss。详细的参数说明请参见“pg_hba.conf”配置文件中的描述。

> **说明**
>
> 指定-h的情况下，默认会尝试修改所有节点的pg_hba.conf文件，但是如果是编译安装，由于没有静态文件，可以成功修改本节点的pg_hba.conf文件，但是会提示静态文件不存在。

- -?, –help

  显示帮助信息。

- -V, –version

  显示版本信息。

- -M, –keymode=MODE

  设定该密码在数据库运行过程中，用于服务端(server)、客户端(client)还是创建Data Source对象时调用。

  取值范围：

  - server
  - client
  - source

  默认值：server

- -K PASSWORD

  指定需要加密的密码。

  取值范围：字符串，符合密码复杂度的要求。

- -U, –keyuser=USER

  指定要加密的用户，隶属于OS用户。

> **说明**
>
> VexDB可以为每个用户配置不同的ssl证书和私钥文件，指定该选项，可以生成按用户名区分的密码文件。

- -S CIPHERKEY

  指定需要加密的字符串，长度应在8~16之间，并且至少有3种不同类型的字符。

  取值范围：字符串。

- -R RANDFILEDIR

  指定放置rand文件的目录。

- -C CIPHERFILEDIR

  指定放置cipher文件的目录。

- -o PREFIX

  指定需要输出的OBS类型的cipher和rand文件前缀名称字符串，默认输出文件名前缀为obsserver。

  取值范围：仅支持数字、字母和下划线。

- -Z NODE-TYPE

  指定数据库实例节点类型，参数默认值为datanode。NODE-TYPE用于识别配置文件。

  取值范围：datanode。

> **说明**
>
> 数据库实例节点类型有coordinator、datanode、cmserver、cmagent和gtm。VexDB只能取值为datanode，当NODE-TYPE为datanode时，对应数据库的数据目录中的配置文件postgresql.conf。

使用示例

示例1： 修改数据库允许的最大连接数为800。修改后需要重启数据库才能生效。

```
vb_guc set -D $PGDATA -c "max_connections = 800"
```

返回结果如下：

```
Total instances: 1. Failed instances: 0.
Success to perform vb_guc!
```

示例2： 将已设置的数据库允许的最大连接数值修改为默认值。修改后需要重启数据库才能生效。

```
vb_guc set -D $PGDATA -c "max_connections"
```

返回结果为：

```
Total instances: 1. Failed instances: 0.
Success to perform vb_guc!
```

示例3： 检查VexDB中各个实例的参数配置情况。

```
vb_guc check -D $PGDATA -c "max_connections"
```

返回结果为：

```
Total GUC values: 1. Failed GUC values: 0.
The value of parameter max_connections is same on all instances.
    max_connections=200
```

示例4： 设置客户端认证策略。

1、执行如下命令设置客户端认证策略。

```
vb_guc set -D $PGDATA -h "host replication testuser 10.252.95.191/32 sha256"
```

返回结果为：

```
Total instances: 1. Failed instances: 0.
Success to perform vb_guc!
```

2、查询pg_hba.conf配置文件。

```
cat $PGDATA/pg_hba.conf | grep "10.252.95.191/32"
```

结果如下所示，客户端认证策略已经成功设置。

```
host replication testuser 10.252.95.191/32 sha256
```

示例5： 注释清理已经设置的客户端认证策略。

1、执行如下命令注释已经设置的客户端认证策略。

```
vb_guc set -D $PGDATA -h "host replication testuser 10.252.95.191/32"
```

返回结果为：

```
Total instances: 1. Failed instances: 0.
Success to perform vb_guc!
```

2、查询pg_hba.conf配置文件。

```
cat $PGDATA/pg_hba.conf | grep "10.252.95.191/32"
```

结果如下所示，示例4中设置的客户端认证策略被注释。

```
#host replication testuser 10.252.95.191/32 sha256
```

vb_initdb

功能描述

vb_initdb工具用于初始化数据库，在初始化数据库时会创建数据库目录、生成大量系统表和系统视图、创建默认数据库和模板数据库。

初始化时生成的数据库如下所示，初始时如下四个数据库配置一致：

数据库名称	数据库介绍	数据库功能
template0	指初始化实例时生成的数据库模板，未经过用户或应用程序修改。 作为默认模板生成“干净”的数据库。	只读 使用 CREATE DATABASE 命令默认创建基于template0模板复制而来的数据库。
template1	自定义数据库模板	管理员可连接进行修改，修改后使用 CREATE DATABASE TEMPLATE template1 命令可创建一个基于自定义模板复制而来的数据库。
postgres	系统默认数据库	供内部管理使用，不建议用户直接操作。

> **说明**
>
> - 连接工具vsql的默认数据库为数据库安装用户同名的数据库。
> - 若使用postgres以外的安装用户，不指定数据库连接，则会报错数据库不存在，用户应显式指定已存在的数据库名。

注意事项

- 在初始化实例时可以通过参数来指定数据库的兼容模式，创建实例完成后，在实例中可提供并仅提供此兼容模式的特有功能。
- 在安装数据库时，推荐使用-D参数调用vb_initdb初始化数据库。如果由于故障恢复等原因，需要重新初始化一个数据库，可以通过执行vb_initdb来完成。
- 尽管vb_initdb会尝试创建相应的数据目录，但可能没有权限执行此操作，因为要创建目录的父目录通常被root所拥有。如果要创建数据目录，首先用root用户创建一个空数据目录，然后用chown命令把该目录的所有权交给数据库用户。
- vb_initdb决定初始化数据库的设置，而该设置将会成为其他数据库的默认设置。
- vb_initdb初始化数据vb库的缺省区域和字符集编码。字符集编码、字符编码排序（LC_COLLATE）和字符集类（LC_CTYPE，如大写、小写数字等）可以在创建数据库时独立设置。

语法格式

```
vb_initdb [OPTION]... [DATADIR]
```

参数说明

- -A, –auth=METHOD

  指定本地用户连接数据库时的认证方法，即“pg_hba.conf”配置文件中host和local所在行的认证方法。

  METHOD取值范围：trust、reject、md5（不安全的算法，为了兼容老版本而存在）、sha256、sm3

  默认值：trust

  > **说明**
  >
  > - 除非用户对本地用户都是信任的，否则不要使用默认值trust。
  > - 若取值为md5，则需手动修改参数文件 postgresql.conf.sample中的密码存储类型 password_encryption_type参数的值修改为0，且放开注释使之生效。vs_initdb工具需同时配合-W选项一起使用。

- –auth-host=METHOD

  指定本地用户通过TCP/IP连接数据库时的认证方法，即：“pg_hba.conf”配置文件中host所在行的认证方法。

  METHOD取值范围：trust、reject、md5（不安全的算法，为了兼容老版本而存在）、sha256、sm3

  默认值：trust

> **说明**
>
> 指定此参数则会覆盖-A参数的值。

- –auth-local=METHOD

  指定本地用户通过Unix域套接字连接数据库时的认证方法，即“pg_hba.conf”配置文件中local所在行的认证方法。

  METHOD取值范围：trust、reject、md5（不安全的算法，为了兼容老版本而存在）、sha256、sm3、peer（仅用于local模式）

  默认值：trust

  > **说明**
  >
  > 指定此参数则会覆盖-A参数的值。

- -c, –enable-dcf

  设置安装的节点为DCF模式。

- –enable-dss

  开启资源池化参数功能。

- -I

  指定节点ID，初始化资源池化参数ss_instance_id。

  取值范围：[0-63]， 需要从0开始指定。

- [-D, –pgdata=]DATADIR

  指定数据目录的位置。

  DATADIR的取值：用户自定义。不能包括“|”， “;”， “&”， “$”， “<”， “>”， “`”， “”， “!”这几个字符。

- –nodename=NODENAME

  初始化的节点名称。

  节点的命名需要遵守如下规范：

  - 节点名称必须为小写字母（a-z）、下划线（_）、特殊符号#、数字（0-9）。
  - 节点名称必须以小写字母（a-z）或下划线（_）开头。
  - 节点名称不能为空，且最大的长度为64个字符。

- –vgname=VGNAME

  卷组名。

  数据类型：字符串 。

  例如：一个卷组“+data”或者两个卷组中间用“，”连接“+data,+log”。

- –socketpath=SOCKETPATH

  dss实例进程使用的socket文件路径。支持绝对路径。

- –dms_url=URL

  节点之间mes通信url。

  数据类型：字符串。

  格式：节点id:ip:port,节点id:ip:port,.....

  例如：“0:127.0.0.1:1611,1:127.0.0.1:1711”

- -E, –encoding=ENCODING

  指定数据库编码格式。不能包括“|”，“;”，“&”，“$”，“<”，“>”，“`”，“”，“!”这几个字符。

  > **说明**
  >
  > - 如果使用此参数，需要加上–locale选项指定支持此编码格式的区域。如果不加–locale选项，则采用系统默认的区域，如果系统默认区域的编码格式和用此参数指定的编码格式不匹配则会导致数据库初始化失败。
  > - 如果不指定此参数，则使用系统默认区域的编码格式。系统默认区域和编码格式可以使用locale命令查看，如下：
  >
  > ```
  > locale | grep LC_CTYPE
  > ```

- –locale=LOCALE

  指定数据库的缺省区域，如果不希望指定特定的区域，则可以用C。可以用locale -a命令查看可用的区域。

  例如用户要将数据库编码格式初始化为GBK，可以采用如下步骤：

  1、用命令查看系统支持gbk编码的区域。

  ```
  locale -a | grep gbk
  ```

显示结果为：

```
zh_CN.gbk
zh_SG.gbk
```

2、初始化数据库时加入 `--locale=zh_CN.gbk` 选项。

> **说明**
>
> - 如果用户设置了数据库的编码格式，则用户选择区域的编码格式必须与用户设置的编码格式一致，否则数据库初始化会失败。
> - 不能包括“|”，“;”，“&”，“$”，“<”，“>”，“`”，“”，“!”这几个字符。

- –enable-oralob-type

  启用兼容 Oracle的 LOB 实现方案。

  初始化实例时指定此选项表示 LOB 使用定位器存储方式，不指定此选项则默认使用直接存储数据的存储方式。

- –pad-attribute=PADATTRIBUTE

  指定数据库默认的列校对规则。

  取值范围：

  - N,NO PAD：把字符串尾端的空格当作一个字符处理，即字符串等值比较不忽略尾端空格。
  - S,PAD SPACE：字符串等值比较忽略尾端空格。

  默认值：N,NO PAD

- –lc-collate=LOCALE, –lc-ctype=LOCALE, –lc-messages=LOCALE,–lc-monetary=LOCALE, –lc-numeric=LOCALE, –lc-time=LOCALE

  为新数据库设置指定范畴的区域。

  > **说明**
  >
  > 各参数的取值必须是操作系统支持的值。不能包括“|”，“;”，“&”，“$”，“<”，“>”，“`”，“”，“!”这几个字符。

- –no-locale

  等价于和–locale=C。

- –pwfile=FILE

执行vb_initdb命令时，从文件FILE中读取数据库中系统管理员的密码。该文件的第一行将被当作密码使用。

用户密码规则如下：

- 取值范围：字符串。
- 密码长度大于等于8个字符，小于等于32个字符。
- 不能与用户名及用户名倒序相同。
- 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。

> **说明**
>
> - FILE可以是“相对路径+文件”的形式，也可以是“绝对路径+文件”的形式。相对路径是相对当前路径的。
> - 支持对pwfile参数指定的密码做合法性校验。

- -T, –text-search-config=CFG

  设置缺省的文本搜索方式。此配置项的值不会做正确性校验，配置成功后，有日志记录提醒当前配置项的取值。

  取值范围：

- english：全文搜索。
- simple：普通文本搜索。

默认值：simple

- -U, –username=NAME

  选择数据库系统管理员的用户名。

  用户自定义创建的的数据库系统管理员的用户名（通常称为数据库初始用户）。

  默认值：运行vb_initdb的操作系统用户名（数据库安装用户同名）。

  取值范围：用户名不能包括“|”，“;”，“&”，“$”，“<”，“>”，“`”，“”，“!” 字符。

> **说明**
>
> - 初始化时会进行用户名合规性检查，不允许以数字、$符号开头的用户名和系统保留用户名（如public、none、gs_role_）作为初始化用户名。
> - 建议自定义的用户名或安装数据库的操作系统用户名中包含的字母均使用小写。这是因为创建初始化用户时会区分大小写，但VexDB默认执行SQL时不区分大小写，所以在执行SQL时指定含大写的初始化用户名需要被双引号包裹才能被识别，否则SQL语句无法执行。

- -W, –pwprompt

  执行vb_initdb命令时强制交互式输入数据库管理员的密码。

- -w, –pwpasswd=PASSWD

  执行vb_initdb命令时通过命令行指定的管理员用户的密码，而不是交互式输入。使用-w指定的密码VexDB会进行合法性检测。

  设置的密码要符合复杂度要求：

  - 最少包含8个字符，不能超过32个字符。
  - 不能和用户名和当前密码相同，或和当前密码反序。
  - 至少包含大写字母（A-Z）、小写字母（a-z）、数字、非字母数字字符（限定为“~”，“！”，“@”，“#”，“$”，“%”，“^”，“&”，“*”，“(”，“)”，“-”，“_”，“=”，“+”，“”，“|”，“[”，“{”，“}”，“]”，“;”，“:”，“,”，“<”，“>”，“/”，“?”）四类字符中的三类字符。

- -C, –enpwdfiledir=DIR

  执行vb_initdb命令时指定的经AES128加密算法加密过的密码文件所在目录。vb_initdb会将该目录下密码文件进行解密，同时把解密后的密码做密码复杂度校验，校验通过的话会将此密码作为用户的密码。

  > **说明**
  >
  > - 加密密码文件需使用vb_guc工具生成，例如 `vb_guc encrypt -K Gauss@123 -D Dir` 。
  > - 如果用户指定多个-w和-C参数，vb_initdb会将用户输入的最后一个-w或-C参数作为用户的需求，即输入密码的明文或经过AES128加密后的密码。
  > - 不能包括“|”，“;”，“&”，“$”，“<”，“>”，“`”，“”，“!”这几个字符。

- -X, –xlogdir=XLOGDIR

  声明事务日志存储的目录。

  > **说明**
  >
  > - 所设置的目录，必须满足运行VexDB的用户有读写权限。
  > - 只支持绝对路径。
  > - 路径不能包括“|”，“;”，“&”，“$”，“<”，“>”，“`”，“”，“!”这几个字符。

- -S, –security

  使用安全方式初始化数据库。以-S方式初始化的数据库后，创建的数据库用户权限受到限制，默认不再具有public schema的使用权限。

- -g, –xlogpath=XLOGPATH

  共享存储的XLOG文件路径。

- -d, –debug

  从初始化后端打印调试输出信息。初始化后端是vb_initdb用于创建系统表的程序。

- -L DIRECTORY

  vb_initdb初始化数据库所需要的输入文件的路径。

  > **说明**
  >
  > - 该参数通常是不必要的。如果需要明确声明的话，程序会提示输入。因该参数意为创建指定配置信息的数据库，建议将share/postgresql下的所有涉及启动相关的子目录及文件全部复制过来，避免其他因素的影响。
  > - 初始化数据库所需的输入文件路径。不能包括“|”,, “;”, “&”, “$”, “<”, “>”, “`”, “”, “!”这几个字符。

- -n, –noclean

  这个选项禁止任何清理动作，因而对调试很有用。不指定该参数时，当vb_initdb发现一些错误妨碍它完成创建数据库的工作时，它将在检测到不能结束工作之前将其创建的所有文件删除。

- -s, –show

  显示内部设置。

- -H, –host-ip

  初始化VexDB节点node_host。不能包括“|”,, “;”, “&”, “$”, “<”, “>”, “`”, “”, “!”这几个字符。

- –audit_encrypt_algorithm=AUDITENCRYPTALGORITHM

  审计日志加密参数，指定审计日志加密算法。

  取值范围：加密算法必须为SM4。

- –audit_master_key=MASTERKEY

  主密钥参数，设置主密钥。

  取值范围：数据库初始化时设置主密钥长度必须等于16位。

- –audit_encrypt_key=ENCRYPTKEY

  加密密钥参数，设置加密密钥。

取值范围：数据库初始化时设置加密密钥长度必须等于16位。

> **说明**
>
> –audit_encrypt_algorithm=AUDITENCRYPTALGORITHM，–audit_master_key=MASTERKEY，–audit_encrypt_key=ENCRYPTKEY三个选项相关内容请参考审计日志加密存储。

- -V, –version

  打印vb_initdb版本信息，然后退出。

- -?, –help

  显示关于vb_initdb命令行参数的帮助信息，然后退出。

使用示例

执行如下命令初始化数据库。

```
vb_initdb  -D /home/vexdb/data/vexdb--nodename postgres -w vbase@123
```

/home/vexdb/data/vexdb为数据库目录，需要0700权限。详细请参见安装指南步骤6：初始化数据库实例（可选）。

vb_licensetool

功能描述

vb_licensetool工具用于查看license的相关信息。

信息	说明
Customer	标识许可适用的客户名称。 临时许可显示temporary license。
Begins On	许可生效的日期时间。
Expires On	许可失效的日期时间。
MAC	申请许可时填入的设备 MAC 地址。
Model	VexDB 数据库的产品型号。
Support Module	数据库的功能支持情况。

语法格式

```
vb_licensetool [OPTION]...
```

参数说明

- –dump = < path >

  显示正式license的详细信息，其中path为license的存储路径。

- –load=

  加载license。其中license-path为license的存储路径。

- –pgdata=

  指定数据库目录。其中DATADIR为数据库目录绝对路径。

- –view-temporary

  显示自动生成临时license的详细的信息。

- –help，-?，-h

  打印帮助信息。

示例

示例1： 查看临时license的信息

```
vb_licensetool  --view-temporary
```

返回结果如下：

```
license info: Customer:'temporary license', Begins On:'2025-04-14 16:23:59', Expires On:'2025-07-13 16:23:59', MAC:'', Model:'G100', Support Module:'BASIC'
```

示例2：查看正式license的信息

```
vb_licensetool --dump=/soft/VexDB_license_20230701
```

结果如下所示，返回正式license的信息：

```
license path: /soft/vexdb_license_20230701

license info: Customer:'TEST', Begins On:'2023-02-03 16:49:01', Expires On:'2023-07-01 16:51:29', MAC:'', Model:'', Support Module:'BASIC'
```

vb_probackup

vb_probackup是一个用于管理VexDB数据库物理备份和恢复的工具。它可以对VexDB实例进行定期备份，以便在数据库出现故障时能够恢复服务器。

vb_probackup工具的主要功能如下：

- 备份：vb_probackup工具采用物理备份的方式进行备份，并且支持备份压缩。对数据库进行备份，可以增强数据库的抗风险能力。
  - 可用于备份单机数据库，也可对主机或者主节点数据库备机进行备份。
  - 可通过-E选项备份外部目录的内容，如脚本文件、配置文件、日志文件、dump文件等。
  - 支持全量备份、增量备份、备份合并、远程备份、通过定时任务脚本实现定期备份。
  - 除了基础的备份恢复操作以外，vb_probackup通过不同子命令实现了备份实例的管理、设置备份的留存策略等功能。

> **说明**
>
> 在进行全量备份前，需要配置归档目录。

- 恢复：vb_probackup工具支持全量恢复与增量恢复，提供将数据库恢复到之前状态的能力。vb_probackup工具提供的恢复能力，是数据库安全可靠的重要保证。
- 验证：vb_probackup提供了验证备份完整性和一致性的功能，确保备份数据的可靠性。

关于vb_probackup的详细用法请参考vb_probackup。

vb_restore

vb_restore是VexDB提供的针对vb_dump导出数据的导入工具。通过此工具可导入由vb_dump生成的自定义归档格式（c）、目录格式（d）、tar格式（t）的备份文件。

vb_restore的主要功能包含：

- **导入到数据库**

  如果连接参数中指定了数据库，则数据将被导入到指定的数据库中。其中，并行导入必须指定连接的密码。导入时生成列会自动更新，并像普通列一样保存。

- **导入到脚本文件**

  如果未指定导入数据库，则必须指定list选项。将会创建包含重建数据库所必须的SQL语句脚本并写入到文件或者标准输出。等效于直接使用vb_dump导出为纯文本格式，多用于TOC文件选择性恢复。

关于vb_restore的详细内容请参考vb_restore。

VexDB

功能描述

vexdb是 VexDB数据库的主线程，也是一个可执行的命令，能够通过其启动一个数据库线程。

客户端应用程序为了访问数据库，将连接（通过网络或本地）到一个正在运行的VexDB线程。然后该线程实例会启动一个独立的线程来处理这个连接。

> **说明**
>
> 通过vexdb启动数据库时，需要再开一个窗口来连接到数据库，也可以使用&符号使程序在后台执行。

一个 vexdb线程总是管理来自同一个数据库的数据。一个系统上可以同时运行多个 vexdb线程，只要使用不同的数据目录和不同的端口号。 vexdb启动时需要知道数据目录的位置，该位置必须通过-D指定。通常，-D直接指向由 vb_initdb 创建的数据库目录。

缺省时 vexdb在前台启动并将日志信息输出到标准错误。但在实际应用中，vexdb应当作为后台进程启动，而且多数是在系统启动时自动启动。

vexdb还能以单用户模式运行，这种用法主要用于vb_initdb 的初始化过程中，有时候也被用于调试灾难性恢复。不过，单用户模式运行的服务器并不适合于调试，因为没有实际的进程间通讯和锁动作发生。

当从shell上以单用户模式调用时，用户可以输入查询，然后结果会在屏幕上以一种更适合开发者阅读（不适合普通用户）的格式显示出来。在单用户模式下，将把会话用户ID设为1并赋予系统管理员权限。该用户不必实际存在，因此单用户模式运行的服务器可以用于对某些意外损坏的系统表进行手工恢复。

数据库正常启动时，不能通过 single 单用户模式登录。

语法格式

```
vexdb [OPTION]...
```

参数描述

通用选项

表1 通用选项

参数	参数说明	取值范围
-B NBUFFERS	设置服务器线程使用的共享内存缓冲区的数量。	-
-b BINARY UPGRADES	binary upgrade 标志。	-
-c NAME=VALUE	给一个正在运行的参数赋值。 可以同时指定多个-c 从而设置多个参数。	-
-C NAME	打印一个命名的运行时的参数的值然后退出。可以用在正在运行的服务器上，从 postgresql.conf 返回值。	postgresql.conf 中的所有参数。
-d 1-5	设置调试级别，1-5记录对应 debug 级别信息。级别越高，写到服务器日志的调试输出越多。	取值范围：1-5 说明：如果取值小于等于0，则只会记录 notice 级别的信息。
-D DATADIR	声明数据目录或者配置文件的文件系统路径。	用户自定义路径。
-e	把缺省日期风格设置为“European”，也就是说用 DMY 规则解释日期输入，并且在一些日期输出格式里日在月份前面打印。	-
-F	关闭 fsync 调用以提高性能，但是要冒系统崩溃时数据毁坏的风险。声明这个选项等效关闭了 fsync 参数。	-
-h HOSTNAME	指定 VexDB 侦听来自前端应用 TCP/IP 连接的主机名或 IP 地址。	前端存在的主机名或 IP 地址。
-i	该选项允许远程客户通过 TCP/IP（网际域套接字）与服务器通讯。	-
-k DIRECTORY	指定 VexDB 侦听来自前端应用连接的 Unix 域套接字的目录。	缺省通常是/tmp ，但是可以在编译的时候修改。
-l	该选项允许远程客户通过 SSL（ 安全套接层）与服务器通讯。	-
-N MAX-CONNECT	设置服务器接受的客户端连接的最大数。缺省时由 vb_initdb 自动选择。声明这个选项等价于声明 max_connections 配置参数。	取值范围：正整数。
-M SERVERMODE	在启动时指定数据库的启动模式。	SERVERMODE 可以取下面四个值： • primary：本端以主机模式启动。 • standby：本端以备机模式启动。 • pending：本端处于等待状态，等待提升为主机或者备机。
-o OPTIONS	向每个服务器进程传递 "OPTIONS"。该选项仅保留语法，没有实际功能。	-
-p PORT	指定 VexDB 侦听客户端连接的 TCP/IP 端口或本地 Unix domain socket 文件的扩展。默认端口号为15400。	正整数，在操作系统支持的端口范围内。
-s	在每条命令结束时打印时间信息和其他统计信息。	-
-S WORK-MEM	声明内部排序和散列在求助于临时磁盘文件之前可以使用的内存大小。	单位为 KB。
-u NUM	指定升级前数据库内核版本号。	-
-V, –version	打印 vexdb 工具的版本信息然后退出。	-
–NAME=VALUE	给一个正在运行的参数赋值。	-

–describe-config	描述配置参数然后退出。	-
–securitymode	以安全模式运行。 **说明** 在安全模式下，不允许执行COPY TO/FROM filename，不允许对public角色授权，不允许创建和修改表空间，用户需要具有USEFT属性，才能创建、修改、删除和使用外表。	-
–single_node	拉起单机数据库。这是默认选项。	-
-?, –help	显示关于VexDB命令行参数的帮助信息，然后退出。	-

开发者选项

下表参数主要是便于开发人员调试使用，有时也用于帮助恢复严重损坏的数据库。在应用程序使用数据库提供服务时，请不要使用这些参数进行调试。

表2 开发者选项

参数	参数说明	取值范围
-f s|i|n|m|h	禁止某种扫描和连接方法的使用。	取值范围： • s：关闭顺序 • i：索引扫描 • n：关闭嵌套循环 • m：融合（merge）连接 • h：Hash连接
-n	主要用于调试导致服务器进程异常崩溃的问题。一般策略是通知所有其他服务器进程终止并重新初始化共享内存和信号灯。该选项指定vexdb不重新初始化共享内存。	-
-O	允许修改系统表的结构。	-
-P	读系统表时忽略系统索引，但在修改表时仍然更新索引。 须知：此选项可能导致系统表损坏，甚至数据库无法启动。	-
-t pa|pl|ex	打印与每个主要系统模块相关的查询记时统计。	-
-T	主要用于调试导致服务器进程异常崩溃的问题。该选项指定vexdb通过发送SIGSTOP信号停止其他所有服务器进程，但是并不让它们退出。这样就允许系统程序员手动从所有服务器进程搜集内核转储。	-
-W NUM	指定一个新的服务器进程开始需要等待的秒数。	单位：秒
–localxid	使用本地事务ID，而不是全局事务ID。 须知：此选项仅用于vb_initdb。使用此选项可能会导致数据库不一致。	已存在的本地事务ID

单用户模式选项

表3 单用户模式选项

参数	参数说明	取值范围
–single	启动单用户模式。必须是命令行中的第一个选项。	-
DBNAME	要访问的数据库的名称。必须是命令行中的最后一个选项。	字符串。默认为用户名。
-d 0-5	重新指定调试级别。	0-5
-E	回显所有命令。	-
-j	禁止使用新行作为语句分隔符。	-
-r FILENAME	将所有服务器标准输出和标准错误保存到文件 filename 中。	-

自启动模式选项

表4 自启动模式选项

参数	参数说明	取值范围
–boot	启动自启动模式。必须是命令行中的第一个选项。该参数是对数据库中的参数变量及相关配置初始化，常用在数据库安装的流程中，直接使用该参数，无明显行为感知。	-
-r FILENAME	将所有服务器标准输出和标准错误保存到文件 filename 中。	-
-x NUM	指定一个新的服务器线程的类型。在初始化数据库时，会用到自启动模式，通过设置这个参数，启动不同线程来执行一些逻辑，正常情况下，不会用到，因为自启动模式很少会用到。	0-5
-G	在 initdb 时将表存储在段页中。	

错误处理

一个提到了 semget 或 shmget 的错误信息可能意味着需要重新配置内核，提供足够的共享内存和信号灯。可以通过降低 shared_buffers 值以减少 VexDB的共享内存的消耗，或者降低 max_connections 值减少 VexDB的信号灯的消耗。

如果发现类似“另外一个服务器正在运行”的错误信息，可以根据系统使用不同的命令：

```
ps ax | grep vexdb
```

或

```
ps -ef | grep vexdb
```

如果确信没有冲突的服务器正在运行，可以删除消息里提到的锁文件然后再次运行。

无法绑定端口的错误信息可能表明该端口已经被其他非 vexdb进程使用。如果终止 vexdb后又马上用同一端口号运行它，也可能得到错误信息。这时，必须多等几秒，等操作系统关闭了该端口再试。最后，如果使用了一个操作系统认为是保留的端口，也可能导致这个错误信息。例如：Unix版本认为低于1024的端口号是“可信任的”，因而只有Unix系统管理员可以使用它们。

> **说明**
>
> - 使用SIGKILL杀死主进程会阻止vexdb前释放它持有的系统资源（例如共享内存和信号灯），会影响新的进程。
> - 可以使用SIGTERM，SIGINT，SIGQUIT信号正常结束服务器进程。第一个信号将等待所有的客户端退出后才退出。第二个将强制断开所有客户端，而第三个将不停止立刻退出，导致在重启时的恢复运行。
> - 信号SIGHUP将会重新加载服务器配置文件。它也可能给单个服务器进程发送SIGHUP信号，但是这通常是不明显的。
> - 要取消一个正在执行的查询，向正在运行的进程发送SIGINT信号。

用法

示例1： 当数据库无法正常启动时，切换进入数据库安装目录下使用单用户模式登录数据库。

1、切换到数据库安装目录下的bin目录下，当安装为标准化安装时，路径如下：

```
cd ~/local/vexdb/bin
```

2、使用单用户模式登录。

```
vexdb --single -D ~/data/vexdb vexdb
```

- 用-D给服务器提供正确的数据库目录的路径。同时还要声名已存在的特定数据库名称。
- 通常，独立运行的服务器把换行符当做命令输入完成字符；要想把一行分成多行写，必需在除最后一个换行符以外的每个换行符前面敲一个反斜杠。
- 如果使用了-j命令行选项，新行将不被当作命令结束符。此时服务器将从标准输入一直读取到EOF标志为止，然后把所有读到的内容当作一个完整的命令字符串看待，并且反斜杠与换行符也被当作普通字符来看待。
- 输入EOF（Control+D）即可退出会话。如果已经使用了-j则必须连续使用两个EOF才行。
- 单用户模式运行的服务器不会提供复杂的行编辑功能（比如没有命令历史）。单用户模式也不会做任何后台处理，像自动检查点。

示例2： 用缺省值在后台启动vexdb。

```
nohup vexdb> logfile 2>&1 < /dev/null &
```

vsql

功能描述

vsql是VexDB提供的在命令行下运行的数据库连接工具，用户可以通过此工具连接服务器并对其进行操作和维护，除了具备操作数据库的基本功能，vsql还提供了若干高级特性，便于用户使用。

基本功能

- 连接数据库： 详情请参考使用vsql连接。
- 执行SQL语句： 支持交互式地键入并执行SQL语句，也可以执行一个文件中指定的SQL语句。
- 执行元命令： 元命令可以帮助管理员查看数据库对象的信息、查询缓存区信息、格式化SQL输出结果、以及连接到新的数据库等。元命令的详细说明请参见元命令参考。

> **说明**
>
> 在vsql里输入的任何以不带引号的反斜杠开头的命令都是vsql元命令，这些命令是将由vsql自己处理的，元命令通常也被称为斜杠或反斜杠命令。

高级特性

vsql支持的高级特性包括：

- 设置变量：vsql提供类似于Linux的shell命令的变量特性，可以使用vsql的元命令 `\set` 设置一个变量。
- SQL代换：利用vsql的变量特性，可以将常用的SQL语句设置为变量，以简化操作。
- 提示符：vsql使用的提示符支持用户自定义。
- 命令自动补齐：vsql支持使用Tab键进行命令的自动补齐，当编译时指定了选项 `--with-readline` ，且客户端连接时指定 `-r` 参数，此功能被打开。

> **说明**
>
> 客户端连接时内置-r参数，无需用户显式指定。

- 客户端操作历史记录：vsql支持客户端操作历史记录，当客户端连接时指定 `-r` 参数，此功能被打开。客户端连接时内置了 `-r` 参数，无需用户显式指定，本功能默认可用。

设置变量

vsql支持使用元命令 `\set` 设置一个变量，格式如下：

```
\set varname value
```

删除变量使用如下语法：

```
\unset varname
```

例如：把变量foo的值设置为bar： `\set foo bar` ，如果想要引用变量的值，在变量前面加冒号。例如查看变量的值： `\echo :foo`

上述变量的引用方法适用于正规的SQL语句和元命令，除此之外vsql还包含一些特殊变量，同时也规划了变量的取值。为了保证和后续版本最大限度地兼容，请避免以其他目的使用这些变量。所有特殊变量见表1. 特殊变量设置。

> **说明**
>
> - 所有特殊变量都由大写字母、数字和下划线组成。
> - 要查看特殊变量的默认值，请使用元命令:varname（例如:DBNAME）。

表1 特殊变量设置

变量	设置方法	变量说明
DBNAME	`\set DBNAME dbname`	当前连接的数据库的名称。每次连接数据库时都会被重新设置。
ECHO	`\set ECHO all | queries`	• 如果设置为 all ，只显示查询信息。设置为 all 等效于使用 vsql 连接数据库时指定 -a 参数。 • 如果设置为 queries ，显示命令行和查询信息。等效于使用 vsql 连接数据库时指定 -e 参数。
ECHO_HIDDEN	`\set ECHO_HIDDEN  on | off | noexec`	当使用元命令查询数据库信息（例如）时，此变量的取值决定了查询的行为： • 设置为 on，先显示元命令实际调用的查询语句，然后显示查询结果。等效于使用 vsql 连接数据库时指定-E 参数。 • 设置为 off，则只显示查询结果。 • 设置为 noexec，则只显示查询信息，不执行查询操作。
ENCODING	`\set ENCODING encoding`	当前客户端的字符集编码。
FETCH_COUNT	`\set FETCH_COUNT variable`	• 如果该变量的值为大于0的整数，假设为 n，则执行 SELECT 语句时每次从结果集中取 n 行到缓存并显示到屏幕。 • 如果不设置此变量，或设置的值小于等于0，则执行 SELECT 语句时一次性把结果都取到缓存。 设置合理的变量值，将减少内存使用量。一般来说，设为100到1000之间的值比较合理。
HOST	`\set HOST hostname`	已连接的数据库主机名称。
IGNOREEOF	`\set IGNOREEOF variable`	• 若设置此变量为数值，假设为10，则在 vsql 中输入的前9次 EOF 字符（通常是 Ctrl+D 组合键）都会被忽略，在第10次按 Ctrl+D 才能退出 vsql 程序。 • 若设置此变量为非数值，则缺省为10。 • 若删除此变量，则向交互的 vsql 会话发送一个 EOF 终止应用。
LASTOID	`\set LASTOID oid`	最后影响的 oid 值，即为从一条 INSERT 或 lo_import 命令返回的值。此变量只保证在下一条 SQL 语句的结果显示之前有效。
ON_ERROR_ROLLBACK	`\set  ON_ERROR_ROLLBACK  on | interactive | off`	• 如果是 on，当一个事务块里的语句产生错误的时候，这个错误将被忽略而事务继续。 • 如果是 interactive，这样的错误只是在交互的会话里忽略。 • 如果是 off（缺省），事务块里一个语句生成的错误将会回滚整个事务。on_error_rollback-on 模式是通过在一个事务块的每个命令前隐含地发出一个 SAVEPOINT 的方式工作的，在发生错误的时候回滚到该事务块。

ON_ERROR_STOP	`\set ON_ERROR_STOP on | off`	• on：命令执行错误时会立即停止，在交互模式下，vsql 会立即返回已执行命令的结果。 • off（缺省）：命令执行错误时将会跳过错误继续执行。
PORT	`\set PORT port`	正连接数据库的端口号。
USER	`\set USER username`	当前用于连接的数据库用户。
VERBOSITY	`\set VERBOSITY terse | default | verbose`	这个选项可以设置为值 terse、default、verbose 之一以控制错误报告的冗余行。 • terse：仅返回严重且主要的错误文本以及文本位置（一般适合于单行错误信息）。 • 如果是 interactive，这样的错误只是在交互的会话里忽略。 • default：返回严重且主要的错误文本及其位置，还包括详细的错误细节、错误提示（可能会跨越多行）。 • verbose：返回所有的错误信息。

SQL代换

vsql变量可以把变量替换成正规的SQL语句，并且vsql还支持为变量更换新的别名或其他标识符等功能。使用SQL代换的方式替换一个变量的值可以在变量前面加上冒号，例如，执行如下命令即可查询系统表PG_PROC。

1.执行元命令设置别名。

```sql
\set foo 'pg_proc'
```

2.通过元命令查询系统表。

```sql
select * from :foo;
```

> **说明**
>
> 变量的值是逐字复制的，甚至可以包含不对称的引号或反斜杠命令。所以必须保证输入的内容有意义。

提示符

通过如下变量可以设置vsql的提示符，这些变量是由字符和特殊的转义字符所组成。

表2 提示符

变量	描述	示例
PROMPT1	vsql 请求一个新命令时使用的正常提示符。PROMPT1 的默认值为： `%/%R%#`	使用变量 PROMPT1切换提示符： • 提示符变为[local]：PROMPT1 %M • 提示符变为 name：PROMPT1 name • 提示符变为=：PROMPT1 %R
PROMPT2	在一个命令输入期待更多输入时（例如，查询没有用一个分号结束或者引号不完整）显示的提示符。	使用变量 PROMPT2显示提示符： `\set PROMPT2 TESTselect * from pg_proc TEST;`
PROMPT3	当执行 COPY 命令，并期望在终端输入数据时（例如，COPY FROM STDIN），显示提示符。	使用变量 PROMPT3显示 COPY 提示符： `\set PROMPT3 '>>>>'copy pg_proc from STDIN;`

提示符变量的值是按实际字符显示的，但是，当设置提示符的命令中出现%时，变量的值根据%后的字符，替换为已定义的内容，已定义的提示符请参见下表：

表3 已定义的提示符

符号	符号说明
%M	主机的全名（包含域名），若连接是通过 Unix 域套接字进行的，则全名为[local]，若 Unix 域套接字不是编译的缺省位置，就是[local:/dir/name]。
%m	主机名删去第一个点后面的部分。若通过 Unix 域套接字连接，则为[local]。
%>	主机正在侦听的端口号。
%n	数据库会话的用户名。
%/	当前数据库名称。
%~	类似%/，如果数据库是缺省数据库时输出的是波浪线~。
%#	如果会话用户是数据库系统管理员，使用#，否则用>。
%R	• 对于 PROMPT1通常是“=”，如果是单行模式则是“^”，如果会话与数据库断开（如果失败可能发生）则是“!”。 • 对于 PROMPT2该序列被“-”、单引号、双引号或“$”（取决于 vsql 是否等待更多的输入：查询没有终止、正在一个 / ... / 注释里、正在引号或者美元符扩展里）代替。
%x	事务状态： • 如果不在事务块里，则是一个空字符串。 • 如果在事务块里，则是“*”。 • 如果在一个失败的事务块里则是“!”。 • 如果无法判断事务状态时为“?”（比如没有连接）。
%digits	指定字节值的字符将被替换到该位置。
%:name	vsql 变量“name”的值。
%command	command 的输出，类似于使用“^”替换。
%[...%]	提示可以包含终端控制字符，这些字符可以改变颜色、背景、提示文本的风格、终端窗口的标题。例如， `\set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%]%#'` ，这个句式的结果是在VT100兼容的可显示彩色的终端上的一个宽体（1;）黑底黄字（33;40）。

命令自动补齐

根据VexDB语法规则，vsql支持使用Tab键进行命令的自动补齐，当编译时指定了选项 `--with-readline`，且客户端连接时指定 `-r` 参数，此功能被打开。例如，crea后键入Tab，vsql会将其补齐为create。

> **说明**

> 客户端连接时内置了-r参数，无需用户显式指定。

- 支持数据库SQL关键字如SELECT 、CREATE、TABLE等。
- 支持表名、视图名等自定义标识符的补齐。
- 元命令选项‘ S’ 、‘ +’不支持自动补齐。
- 对系统表进行补齐时，前缀是“pg_”、 “gs_”、 “vb_” 才会补齐系统表。
- 不支持建表时字段类型的补齐。
- SELECT后不支持任何补齐。
- 不支持常量与宏的自动补齐。
- select * from a,b...不支持第二个开始表的自动补齐, insert into t1 (col1, col2, ...)不支持第二个列的自动补齐。
- 不支持create tablespace语句with以及with后参数的自动补齐。
- 创建索引不支持local、global的自动补齐，修改索引不支持rebuild自动补齐。
- set语句仅支持自动补全USER和SUPERUSER级别的参数。
- 不支持if exists的自动补齐。
- 不支持表名.列名的自动补齐，如 `alter sequence owned by tableName.colName, owned by` 。
- 不支持自定义操作符自动补齐。使用复制粘贴这种方式输入命令，如果粘贴的命令里面有TAB键有可能会使输入命令的格式错乱，无法正常执行。
- `@$><=;|&{()` 这些特殊字符在sql语句中具有固定含义。如果自定义表名中包含这些特殊字符，那么输入的sql语句从这些字符开始不支持自动补齐。

客户端操作历史记录

vsql支持客户端操作历史记录，当客户端连接时指定-r参数，此功能被打开。可以通过，例如， `HISTSIZE 50` ，将记录历史的条数设置为50， `HISTSIZE 0` ，不记录历史。

> **说明**
- 客户端连接时内置了-r参数，无需用户显式指定。
- 客户端操作历史记录条数默认设置为32条，最多支持记录500条。当客户端交互式输入包含中文字符时，只支持UTF-8 的编码环境。
- 出于安全考虑，将包含PASSWORD、IDENTIFIED敏感词的记录识别为敏感信息，不会记录到历史信息中，即不能通过上下翻回显。

环境变量

与vsql相关的环境变量如下表所示：

表4 与vsql相关的环境变量

名称	描述
COLUMNS	如果columns为0，则由此参数控制 wrapped 格式的宽度。这个宽度用于决定在自动扩展的模式下，是否要把宽输出模式变成竖线的格式。
PAGER	如果查询结果无法在一页显示，它们就会被重定向到这个命令。可以用 命令关闭分页器。典型的是用命令 more 或 less 来实现逐页查看。缺省值是平台相关的。说明：less 的文本显示，受系统环境变量 LC_CTYPE 影响。
PSQL_EDITOR	和 命令使用环境变量指定的编辑器。变量是按照列出的先后顺序检查的。在 Unix 系统上默认的编辑工具是 vi。
EDITOR	
VISUAL	
PSQL_EDITOR_LINENUMBER_ARG	当 和 带上一行数字参数使用时，这个变量指定的命令行参数用于向编辑器传递起始行数。像 Emacs 或 vi 这样的编辑器，这只是个加号。如果选项和行号之间需要空白，在变量的值后加一个空格。例如： `PSQL_EDITOR_LINENUMBER_ARG = '+'PSQL_EDITOR_LINENUMBER_ARG='--line 'Unix 系统默认的是+。`
PSQLRC	用户的.vsqlrc 文件的交互位置。
SHELL	使用！命令跟 shell 执行的命令是一样的效果。
TMPDIR	存储临时文件的目录。缺省是/tmp。

获取帮助

连接数据库时可使用如下命令获取帮助信息。

```
vsql--help
```

连接到数据库后，可以使用如下命令获取帮助信息。

```
help;
```

返回结果如下：

```
You are using vsql, the command-line interface to postgres.
Type:  \copyright for distribution terms
    \h for help with SQL commands
    \? for help with vsqlcommands
    \g or terminate with semicolon to execute query
    \q to quit
```

查看VexDB支持的SQL语句帮助

- 查看VexDB支持的所有SQL语句。

```
\h
```

- 查看某条语句的命令帮助，例如查看CREATE DATABASE命令的参数可使用下面的命令：

```
\help CREATE DATABASE
```

语法格式

```
vsql [OPTION]... [DBNAME [USERNAME]]
```

DBNAME：声明要连接的数据库名称。

参数说明

通用参数

- -c, –command=COMMAND

  声明vsql要执行一条字符串命令然后退出。

- -d, –dbname=DBNAME

  指定想要连接的数据库名称。另外vsql允许使用直接在命令行上指定DBNAME。

  此外vsql允许使用扩展的DBNAME，即 `‘postgres[ql]://[user[:password]@][netloc][:port][,…][/dbname][?param1=value1&…]’` 或 `‘[key=value] […]’` 形式的连接串作为DBNAME，vsql将从连接串中解析连接信息。

- -f, –file=FILENAME

  使用文件作为命令源而不是交互式输入。vsql将在处理完文件后结束。如果FILENAME是-（连字符），则从标准输入读取。

  取值范围：绝对路径或相对路径。

- -l, –list

  列出所有可用的数据库，然后退出。

- -v, –set=, –variable=NAME=VALUE

  使用vsql设置变量。类似于使用元命令。变量的示例和详细说明请参见设置变量。

- -X, –no-vsqlrc

  不读取启动文件，启动文件默认为~/.vsqlrc。

- -1（“one”), –single-transaction

  当vsql使用-1选项执行脚本时，会在脚本的开头和结尾分别加上 `START TRANSACTION/COMMIT` 用以把整个脚本当作一个事务执行。这将保证该脚本完全执行成功，或者脚本无效。

- -?, –help

  显示关于vsql命令行参数的帮助然后退出。

输入输出参数

- -a, –echo-all

  在读取行时向标准输出打印所有内容。

  使用此参数可能会暴露部分SQL语句中的敏感信息，如创建用户语句中的password信息等，请谨慎使用。

- -e, –echo-queries

  把所有发送给服务器的查询同时回显到标准输出。

  使用此参数可能会暴露部分SQL语句中的敏感信息，如创建用户语句中的password信息等，请谨慎使用。

- -E, –echo-hidden

  回显由。

- -D, –with-decryption=SWS-KMS-SM4/TAS-KMS-SM4

  解密方式。主要是针对vb_dump生成的文件，使用vsql解析的时候选择的解密方式，包括SWS-KMS-SM4或TAS-KMS-SM4。

- -k, –with-key=KEY

  使用vsql对导入的加密文件进行解密。

> **说明**
>
> - 对于本身就是shell命令中的关键字符如单引号（’ ）或双引号（ “），Linux shell会检测输入的单引号（’ ）或双引号（” ）是否匹配。如果不匹配，shell认为用户没有输入完毕，会一直等待用户输入，从而不会进入到vsql程序。

> - 不支持解密导入存储过程和函数。

- -L, –log-file=FILENAME

  除了正常的输出源之外，把所有查询输出记录到指定的FILENAME文件中。

  取值范围：绝对路径或相对路径。

  > **说明**
  > - 使用此参数可能会暴露部分SQL语句中的敏感信息，如创建用户语句中的password信息等，请谨慎使用。
  > - 此参数只保留查询结果到相应文件中，主要目标是为了查询结果能够更好更准确地被其他调用者（例如自动化运维脚本）解析，而不是保留vsql运行过程中的相关日志信息。

- -m, –maintenance

  允许在两阶段事务恢复期间连接VexDB。

  > **说明**
  > 该选项是一个开发选项，禁止用户使用，只限专业技术人员使用。功能是：使用该选项时，vsql可以连接到备机，用于校验主备机数据的一致性。

- -n, –no-libedit

  关闭命令行编辑。

- -o, –output=FILENAME

  将所有查询输出重定向到文件FILENAME中。

  取值范围：绝对路径或相对路径。

- -q, –quiet

  安静模式，指定该选项在执行vsql命令时不会打印出额外信息。

- -C, –enable-client-encryption

  当使用-C参数连接本地数据库或者远程连接数据库时，可通过该选项打开全密态数据库开关。

- -s, –single-step

  单步模式运行。意味着每个查询在发往服务器之前都要提示用户，用这个选项也可以取消执行。此选项主要用于调试脚本。

- -S, –single-line

单行运行模式，这时每个命令都将由换行符结束，像分号那样。

输入格式参数

- -A, –no-align

  切换为非对齐输出模式。

- -F, –field-separator=STRING

  设置域分隔符，默认为“|”。

- -H, –html

  打开HTML格式输出。

- -P, –pset=VAR[=ARG]

  在命令行上以。

  > **说明**
  >
  > 该选项是一个开发选项，禁止用户使用，只限专业技术人员使用。功能是：使用该选项时，vsql可以连接到备机，用于校验主备机数据的一致性。

- -R, –record-separator=STRING

  设置记录分隔符。

- -r

  开启在客户端操作中可以进行编辑的模式。

  客户端连接时内置了-r参数，无需用户显式指定，但若用户显式指定-r参数也不会报错。

- -t, –tuples-only

  只打印行。

- -T, –table-attr=TEXT

  允许声明放在HTML table标签里的选项。

  该参数使用时需搭配-H, –html参数一起使用，指定输出为HTML格式。

- -x, –expanded

打开扩展表格式模式。

- -z, –field-separator-zero

  设置非对齐输出模式的域分隔符为空。

  该参数使用时需搭配-A, –no-align参数一起使用，指定非对齐输出模式。

- -0, –record-separator-zero

  设置非对齐输出模式的记录分隔符为空。

  该参数使用时需搭配-A, –no-align参数一起使用，指定非对齐输出模式。

- -2, –pipeline

  使用管道传输密码，禁止在终端使用，必须和-c或者-f参数一起使用。

连接选项

- -h, –host=HOSTNAME

  指定正在运行服务器的主机名或者Unix域套接字的路径。

  如果省略主机名，vsql将通过Unix域套接字与本地主机的服务器相连，或者在没有Unix域套接字的机器上，通过TCP/IP与localhost连接。

- -p, –port=PORT

  指定数据库服务器的端口号。可以通过修改配置文件postgresql.conf中port参数修改默认端口号。

  默认值：5432

- -U, –username=USERNAME

  指定连接数据库的用户。

  取值范围：字符串，默认使用与当前操作系统用户同名的用户。

  > **说明**
  > - 通过该参数指定用户连接数据库时，需要同时提供用户密码用以身份验证。您可以通过交换方式输入密码，或者通过-W参数指定密码。
  > - 用户名中包含有字符，*需要在字符*前增加转义字符才可成功连接数据库。

- -W, –password=PASSWORD

当使用-U参数连接数据库时，可通过该选项指定密码。

元命令参考

该部分介绍使用VexDB数据库命令行交互工具登录数据库后，vsql所提供的元命令。所谓元命令就是在vsql里输入的任何以不带引号的反斜杠开头的命令。

用户想要知道VexDB支持的元命令有哪些，可以连接到数据库之后执行如下命令：

```
\?
```

一般元命令

- 显示VexDB的版权信息。
- or ;

  把当前查询缓冲区的内容发送给服务器并且把查询的输出存到文件中，或者将输出重定向到到shell命令。

  单独一个。
-) [NAME]

  给出指定SQL语句的语法帮助。

  如果没有给出NAME，vsql将列出可获得帮助的所有命令。如果NAME是一个星号（*），则显示所有SQL语句的语法帮助。
- |off]

  控制并发执行开关。

  取值范围：on，off

on：打开控制并发执行开关，且最大并发数为num。

off：关闭控制并发执行开关。

默认值：100

> **说明**
>
> - 不支持事务中开启并发执行以及并发中开启事务。
> - 不支持。
> - 不推荐在并发中使用set语句，否则导致结果与预期不一致。

- parallel内部不允许创建临时表。如需使用临时表，需要在开启parallel之前创建好，并在parallel内部使用。
- _timeout，否则可能会导致并发执行过程中断连。
- 在on之后一条或多条命令，会等到off命令执行后才会执行，因而，每个on需对应一个off，否则会导致on之后的命令无法执行。
- 服务器能接受的最大连接数受max_connections及当前已有连接数限制。
- 设置num时请考虑服务器当前可接受的实际连接数合理指定。

- 退出vsql程序。

查询缓存区元命令

- 使用外部编辑器编辑查询缓冲区或者文件。
-]

  使用外部编辑器编辑函数定义。如果指定了LINE（即行号），则光标会指到函数体的指定行。
- 打印当前查询缓冲区到标准输出。
- 置（或清空）查询缓冲区。
- FILE

  将当前查询缓冲区内容输出到指定文件。

输入/输出元命令

- ...

  在任何vsql客户端登录数据库成功后使用该命令可以执行导入导出数据。

  这是一个运行SQL语句COPY命令的操作，与COPY命令不同的是，COPY命令是服务去在读写指定文件，而vsql读写文件是作为本地的文件系统和服务器之间的跳板取出或写入数据。这意味着文件访问性和权限都是本地用户的，而不是服务器的，因此不需要SQL超级用户权限。

  说明

  - ，格式良好的数据导入，不会对非法字符进行预处理，也无容错能力。导入数据应优先选择COPY。
  - ：对导入数据的总行数进行分批，用户可以手动指定每批数据的行数（batch_size）。详情参考使用元命令导入数据。
  - ，从而实现数据文件的并行导入，目前并发数范围为[1, 8]。
  - ：

- 不支持临时表、二进制文件以及事务内的并行导入。
- 数据导入支持AES128加密时不支持并行导入。
- 在这些情况下，即使指定了parallel参数，仍然会走非并行流程。

- 把字符串写到标准输出。
- FILE

  从文件FILE中读取内容，并将其当作输入，执行查询。
- + FILE KEY

  对vb_dump/vb_dumpall生成的文件进行解密。
- + FILE KEY USERPIN

  执解密操作的用户名和密码。
- FILE

  和，只是对相关文件名的解析不同。

  当在交互模式执行时，两个命令的表现相同。不过，当在脚本中调用时，，而不是当前工作目录。
- + FILE KEY

  和+类似，是对相关文件名的解析不同。
- + FILE KEY USERPIN

  执解密操作的用户名和密码。
- [FILE]

  把所有的查询结果发送到文件里。
- 把字符串写到查询结果输出流里。

显示信息元命令

> **说明**
- 选项S表示显示系统对象。
- 选项+表示显示对象附加的描述信息。

> - PATTERN用来指定要被显示的对象名称。以下元命令中如果声明了PATTERN，只显示名称匹配PATTERN的内容。

- S+]

  列出当前search_path中模式下所有的表、视图和序列。

- S+] NAME

  列出指定表、视图和索引的结构。

  假设存在表a，列出指定表a的结构：

  ```
  \dtable+ a
  ```

- [PATTERN]

  列出所有表、视图和索引。支持disable 状态的约束/索引，由vb_enable_show_disabled_constraints参数控制，默认开启，表示支持展示。

  例如，列出所有名称以f开头的表、视图和索引：

  ```
  \d+ f*
  ```

- 列出所有可用的聚集函数以及它们操作的数据类型和返回值类型。

  例如，列出所有名称以f开头可用的聚集函数以及它们操作的数据类型和返回值类型：

  ```
  \da f*
  ```

- 列出所有可用的表空间。

  例如，列出所有名称以p开头的可用表空间：

  ```
  \db p*
  ```

- 列出所有字符集编码之间的转换

  例如，列出所有名称以c开头的字符集编码转换：

  ```
  \dc c*
  ```

- 列出所有类型转换。

例如，列出所有名称以c开头的类型转换：

```
\dC c*
```

- 显示所有匹配PATTERN的描述。如果没有给出参数，则显示所有可视对象。包括：聚集、函数、操作符、类型、关系（表、视图、索引、序列、大对象）、规则。

  例如，列出所有可视对象：

  ```
  \dd
  ```

- 显示所有默认的使用权限。

  例如，列出所有默认的使用权限：

  ```
  \ddp
  ```

- 列出所有可用域。

  例如，列出所有可用域：

  ```
  \dD
  ```

- 列出所有的Data Source对象。

  例如，列出所有的Data Source对象：

  ```
  \ded
  ```

- 列出所有的外部表。
- 列出所有的外部服务器。
- 列出用户映射信息。
- 列出封装的外部数据。
- 列出所有可用函数以及它们的参数和返回的数据类型。
- a：代表聚集函数。
- n：代表普通函数。
- t：代表触发器。

- w：代表窗口函数。
- 列出所有的文本搜索配置信息。

  例如，列出所有的文本搜索配置信息：

  ```
  \dF+
  ```

- 列出所有的文本搜索字典。
- 列出所有的文本搜索分析器。
- 列出所有的文本搜索模板。
- 列出所有数据库角色。因为用户和群组的概念被统一为角色，所以这个命令等价于。
- _list的别名，显示一个大对象的列表。
- 列出可用的程序语言。
- 列出物化视图。
- 列出所有的模式（名称空间）。
- 列出所有可用的操作符以及它们的操作数和返回的数据类型。
- 列出排序规则。
- 列出一列可用的表、视图以及相关的权限信息。

rolename=xxxx/yyyy：赋予一个角色的权限。

=xxxx/yyyy：赋予public的权限。

xxxx表示赋予的权限，yyyy表示授予这个权限的角色。权限的参数说明请参见下表。

表5 权限的参数说明

	参数	说明
r	SELECT：允许对指定的表、视图读取数据。	
w	UPDATE：允许对指定表更新字段。	
a	INSERT：允许对指定表插入数据。	
d	DELETE：允许删除指定表中的数据。	
D	TRUNCATE：允许清理指定表中的数据。	
x	REFERENCES：允许创建外键约束。由于当前不支持外键，所以该参数暂不生效。	
t	TRIGGER：允许在指定表上创建触发器。	
X	EXECUTE：允许使用指定的函数以及利用这些函数实现的操作符。	
U	USAGE： • 对于过程语言，允许用户在创建函数时，指定过程语言。 • 对于模式，允许访问包含在指定模式中的对象。 • 对于序列，允许使用 nextval 函数。	
C	CREATE： • 对于数据库，允许在该数据库里创建新的模式。 • 对于模式，允许在该模式中创建新的对象。 • 对于序列，允许使用 nextval 函数。 • 对于表空间，允许在其中创建表以及允许创建数据库和模式的时候把该表空间指定为其缺省表空间。	
c	CONNECT：允许用户连接到指定的数据库。	
T	TEMPORARY：允许创建临时表。	
A	ALTER：允许用户修改指定对象的属性。	
P	DROP：允许用户删除指定的对象。	
m	COMMENT：允许用户定义或修改指定对象的注释。	
i	INDEX：允许用户在指定表上创建索引。	
v	VACUUM：允许用户对指定的表执行 ANALYZE 和 VACUUM 操作。	
*	给前面权限的授权选项。	

-]

  列出所有修改过的配置参数。这些设置可以是针对角色的、针对数据库的或者同时针对两者的。

  > **说明**
  > - PATTERN1和PATTERN2表示要列出的角色PATTERN和数据库PATTERN。
  > - 如果声明了PATTERN，只列出名称匹配PATTERN的规则。缺省或指定*时，则会列出所有设置。

  例如，列出postgres数据库所有修改过的配置参数。

  ```
  \drds * postgres
  ```

- 列出所有的数据类型。
- 列出所有数据库角色。

  因为用户和群组的概念被统一为角色，所以这个命令等价于。

- , , , ,

  这一组命令，字母E、i、s、t和v分别代表着外部表、索引、序列、表和视图。可以以任意顺序指定其中一个或者它们的组合来列出这些对象。

  例如：。在命令名称后面追加+，则每一个对象的物理尺寸以及相关的描述也会被列出。

  默认情况下只会显示用户创建的对象。通过PATTERN或者S修饰符可以把系统对象包括在内。

- 列出安装数据库的扩展信息。
- [+]

  列出服务器上所有数据库的名称、所有者、字符集编码以及使用权限。

- FUNCNAME

  显示函数的定义。

  对于带圆括号的函数名，需要在函数名两端添加双引号，并且在双引号后面加上参数类型列表。参数类型列表两端添加圆括号。

  假设存在函数名带圆括号的函数func()name，列出函数的定义。

  ```
  \sf "func()name"(argtype1, argtype2)
  ```

- 列出数据库中所有表、视图和序列以及它们相关的访问特权。如果给出任何pattern ，则被当成一个正则表达式，只显示匹配的表、视图、序列。

格式化元命令

- 对齐模式和非对齐模式之间的切换。
- 把正在打印的表的标题设置为一个查询的结果或者取消这样的设置。
- STRING]

  对于不对齐的查询输出，显示或者设置域分隔符。
-

若当前模式为文本格式，则切换为HTML输出格式。

若当前模式为HTML格式，则切换回文本格式。

- NAME [VALUE]

  设置影响查询结果表输出的选项。NAME的取值见下表可调节的打印选项。

  表6 可调节的打印选项

选项	选项说明	取值范围
border	value 必须是一个数字。通常这个数字越大，表的边界就越宽线就越多，但是这个取决于特定的格式。	在 HTML 格式下，取值范围为大于0的整数。 在其他格式下，取值范围：0（无边框），1（内部分隔线），2（台架）。
expanded (或 x)	在正常和扩展格式之间切换。	当打开扩展格式时，查询结果用两列显示，字段名称在左、数据在右。这个模式在数据无法放进通常的“水平”模式的屏幕时很有用。 在正常格式下，当查询输出的格式比屏幕宽时，用扩展格式。正常格式只对 aligned 和 wrapped 格式有用。
fieldsep	声明域分隔符来实现非对齐输出。这样就可以创建其他程序希望的制表符或逗号分隔的输出。要设置制表符域分隔符，键入fieldsep ''。缺省域分隔符是 '|' (竖条符)。	-
fieldsep_zero	声明域分隔符来实现非对齐输出到零字节。	-
footer	用来切换脚注。	-
format	设置输出格式。允许使用唯一缩写（这意味着一个字母就够了）。	取值范围： unaligned：写一行的所有列在一条直线上中，当前活动字段分隔符分隔。 aligned：此格式是标准的，可读性最好的文本输出。 wrapped：类似 aligned，但是包装跨行的宽数据值，使其适应目标字段的宽度输出。 html：把表输出为可用于文档里的对应标记语言。输出不是完整的文档。 latex：把表输出为可用于文档里的对应标记语言。输出不是完整的文档。 troff-ms：把表输出为可用于文档里的对应标记语言。输出不是完整的文档。
null	打印一个字符串，用来代替一个 null 值。	缺省是什么都不打印，这样很容易和空字符串混淆。
numericlocale	切换分隔小数点左边的数值的区域相关的分组符号。	on：显示指定的分隔符。 off：不显示分隔符。

		忽略此参数，显示默认的分隔符。
pager	控制查询和 vsql 帮助输出的分页器。如果设置了环境变量 PAGER，输出将被指向到指定程序，否则使用系统缺省。	on：当输出到终端且不适合屏幕显示时，使用分页器。 off：不使用分页器。 always：当输出到终端无论是否符合屏幕显示时，都使用分页器。
recordsep	声明在非对齐输出格式时的记录分隔符。	-
recordsep_zero	声明在非对齐输出到零字节时的记录分隔符。	-
tableattr（或 T）	声明放在 html 输出格式中 HTML table 标签的属性（例如：cellpadding 或 bgcolor）。注意：这里可能不需要声明 border，因为已经在border 里用过了。如果没有给出 value，则不设置表的属性。	-
title	为随后打印的表设置标题。这个可以用于给输出一个描述性标签。如果没有给出 value，不设置标题。	-
tuples_only （或者 t）	在完全显示和只显示实际的表数据之间切换。完全显示将输出像列头、标题、各种脚注等信息。在 tuples_only 模式下，只显示实际的表数据。	-
feedback	切换是否输出结果行数。	-

连接元命令

- onnect] [DBNAME|- USER|- HOST|- PORT|-]

  连接到一个新的数据库。当数据库名称长度超过127个字节时，默认前127个字节有效，连接到前127个字节对应的数据库，但是vsql的命令提示符中显示的数据库对象名仍为截断前的名称。

  重新建立连接时，如果切换数据库登录用户，将可能会出现交互式输入，要求输入新用户的连接密码。该密码最长长度为999字节，受限于GUC参数password_max_length的最大值。
- 设置客户端字符编码格式。不带参数时，显示当前的编码格式。
- 输出当前连接的数据库的信息。

操作系统元命令

- 切换当前的工作目录。

取值范围：绝对路径或相对路径，且满足操作系统路径命名规则。

- NAME [VALUE]

  设置环境变量NAME为VALUE，如果没有给出VALUE值，则不设置环境变量。

- 以毫秒为单位显示每条SQL语句的执行时间（不包括屏显打印时间）。

  on表示打开显示。

  off表示关闭显示。

- ! [COMMAND]

  返回到一个单独的Unix shell或者执行Unix命令COMMAND。

变量元命令

- NAME

  提示用户用文本格式来指定变量名称。

-]

  设置内部变量NAME为VALUE或者如果给出了多于一个值，设置为所有这些值的连接结果。如果没有给出第二个参数，只设变量不设值。

  有一些常用变量被SQL特殊对待，它们是一些选项设置，通常所有特殊对待的变量都是由大写字母组成(可能还有数字和下划线)，具体请参考下表。

  表7

名称	命令说明	取值范围
VERBOSITY value	控制错误报告的冗余行。	value 取值范围： default，verbose，terse
ON_ERROR_STOP value	如果设置了这个变量，脚本处理将马上停止。如果该脚本是从另外一个脚本调用的，那个脚本也会按同样的方式停止。如果最外层的脚本不是从一次交互的 vsql 会话中调用的而是用-f 选项调用的，vsql 将返回错误代码3，以示这个情况与致命错误条件的区别（错误代码为1）。	value 取值范围为： on/off，true/false，yes/no，1/0
RETRY [retry_times]	用于控制是否开启语句出错场景下的重试功能，参数 retry_times 用来指定最大重试次数，缺省值为5，取值范围为5-10。当重试功能已经开启时，再次执行RETRY可以关闭该功能。使用配置文件 retry_errcodes.conf 列举需要重试的错误码列表，该文件和 vsql 可执行程序位于同一级目录下。该配置文件为系统配置，非用户定义，不允许用户直接修改。支持的错误场景如下所示： YY001：TCP 通信错误，对端连接重置。 YY002：TCP 通信错误，对端流重置。 YY003：锁等待超时。 YY004：TCP 通信错误，连接超时。 YY005：SET 命令发送失败，查询设置错误。 YY006：内存申请失败。 YY007：通信库错误，通信库内存分配错误。 YY008：通信库错误，无通信库缓存数据。 YY009：通信库错误，通信库释放内存关闭。 YY0010：通信库错误，SCTP、TCP 断开。 YY0011：通信库错误，通信库断开。 YY0012：通信库错误，通信库远程关闭。 YY0013：通信库错误，等待未知通信库通信。 YY0014：通信库错误，无效快照。 YY0015：通信库错误，通讯接收信息错误。 53200：内存耗尽。	retry_times 取值范围为：5-10

	08006：连接失败。 08000：连接异常。 57P01：管理员关闭系统。 XX003：关闭远程流接口。 XX009：重复查询编号。 说明 1、不支持事务块中的语句错误重试。 2、不支持通过 ODBC、JDBC 接口查询的出错重试。 3、含有 unlogged 表的 sql 语句，不支持节点故障后的出错重试。 4、vsql 客户端本身出现的错误，不在重跑考虑范围之内。	

- NAME

  不设置（或删除）vsql变量名。

大对象元命令

_list

显示一个目前存储在该数据库里的所有VexDB大对象和提供给它们的注释。

示例

前提条件

vsql连接数据库时使用的用户需要具备访问数据库的权限。

操作步骤

使用vsql连接postgres数据库。

vsql工具使用-d参数指定目标数据库名、-U参数指定数据库用户名、-h参数指定主机名、-p参数指定端口号信息。

若未指定数据库名称，则使用初始化时默认生成的数据库名称。

若未指定数据库用户名，则默认使用当前操作系统用户作为数据库用户名。

当某个值没有前面的参数（-d、-U等）时，若连接的命令中没有指定数据库名（-d）则该参数会被解释成数据库名。

如果已经指定数据库名（-d）而没有指定数据库用户名（-U）时，该参数则会被解释成数据库用户名。

示例1： 使用vexdb用户连接到本机postgres数据库的5432端口。

```
vsql -d postgres -p 5432
```

示例2： 使用jack用户连接到远程主机（10.180.123.163）postgres数据库的5432端口。

本步骤执行之前应配置远程客户端接入认证，更多配置方法参见配置客户端接入认证。

```
vsql -h 10.180.123.163 -d postgres -U jack -p 15400
```

示例3： 参数postgres和vexdb不属于任何选项时，分别被解释成了数据库名和用户名。

1. 连接数据库。

```
vsql postgres vexdb -p 5432
```

等效于：

```
vsql -d postgres -U vexdb -p 5432
```

详细的vsql参数请参见参数说明。

2. 执行SQL语句。

创建数据库human_staff为例，通常，输入的命令行在遇到分号的时候结束。如果输入的命令行没有错误，结果就会输出到屏幕上。

```
CREATE DATABASE human_staff;
```

3. 执行vsql元命令。

例如列出VexDB中所有数据库和描述信息。

```
\l
```

返回结果如下：

```
List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 postgres  | vexdb | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | vexdb | UTF8     | en_US.utf8 | en_US.utf8 | =c/vexdb          +
           |          |          |            |            | vexdb=CTc/vexdb template1 | vexdb | UTF8     | en_US.utf8 | en_US.utf8 | =c/vexdb          +
           |          |          |            |            | vexdb=CTc/vexdb vexdb | vexdb | UTF8     | en_US.utf8 | en_US.utf8 |
(4 rows)
```

删除创建的数据库。

```
DROP DATABASE human_staff;
```

更多vsql元命令请参见元命令参考。

常见问题处理

连接性能问题

- 开启log_hostname参数，但是配置了错误的DNS导致的连接性能问题。

  连接上数据库，通过show log_hostname语句，检查数据库中是否开启了log_hostname参数。

  如果开启了相关参数，那么数据库内核会通过DNS反查客户端所在机器的主机名。这时如果数据库主节点配置了不正确的或者不可达的DNS服务器，那么会导致数据库建立连接过程较慢。log_hostname参数详细请参见log_hostname参数。

- 数据库内核执行初始化语句较慢导致的性能问题。

  此种情况定位较难，可以尝试使用Linux的跟踪命令： `strace` 。

  ```
  strace vsql -U MyUserName -W MyPassWord -d postgres -h 127.0.0.1 -p 23508 -r -c '\q'
  ```

  此时便会在屏幕上打印出数据库的连接过程。比如较长时间停留在下面的操作上：

  ```
  sendto(3, "Q\0\0\0\25SELECT VERSION()\0", 22, MSG_NOSIGNAL, NULL, 0) = 22
  poll([{fd=3, events=POLLIN|POLLERR}], 1, -1) = 1 ([{fd=3, revents=POLLIN}])
  ```

  此时便可以确定是数据库执行SELECT VERSION();语句较慢。

  在连接上数据库后，便可以通过执行explain performance select version();语句来确定初始化语句执行较慢的原因。

另外还有一种场景不太常见：由于数据库主节点所在机器的磁盘满或故障，此时所查询等受影响，无法进行用户认证，导致连接过程挂起，表现为假死。解决此问题清理数据库主节点的数据盘空间便可。

- TCP连接创建较慢问题。

  此问题可以参考上面的初始化语句较慢排查的做法，通过strace跟踪，如果长时间停留在：

  ```
  connect(3, {sa_family=AF_FILE, path="/home/test/tmp/gaussdb_llt1/.s.Pvsql.61052"}, 110) = 0
  ```

  或者

  ```
  connect(3, {sa_family=AF_INET, sin_port=htons(61052), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EINPROGRESS (Operation now in progress)
  ```

  那么说明客户端与数据库端建立物理连接过慢，此时应当检查网络是否存在不稳定、网络吞吐量太大的问题。

创建连接故障

- vsql: could not connect to server: No route to host

  此问题一般是指定了不可达的地址或者端口导致的。请检查-h参数与-p参数是否添加正确。

- vsql: FATAL: Invalid username/password,login denied.

  此问题一般是输入了错误的用户名和密码导致的，请联系数据库管理员，确认用户名和密码的正确性。

- vsql: FATAL: Forbid remote connection with trust method!

  数据库由于安全问题，禁止远程登录时使用trust模式。这时需要修改pg_hba.conf文件里的连接认证信息。

  > **说明**
  >
  > - 请不要修改pg_hba.conf中VexDB主机的相关设置，否则可能导致数据库功能故障。
  > - 建议业务应用部署在VexDB之外，而非VexDB内部。

- 连接数据库，添加-h 127.0.0.1可以连接，去掉后无法连接问题。

  通过执行SQL语句 `show unix_socket_directory;` 检查数据库主节点使用的Unix套接字目录，是否与shell中的环境变量$PGHOST一致。

  如果检查结果不一致，那么修改PGHOST环境变量到GUC参数unix_socket_directory指向的目录便可。

  关于unix_socket_directory的更多信息，详见GUC参数说明unix_socket_directory。

- The “libpq.so” loaded mismatch the version of vsql, please check it.

  此问题是由于环境中使用的libpq.so的版本与vsql的版本不匹配导致的，请通过“ldd vsql”命令确认当前加载的libpq.so的版本，并通过修改LD_LIBRARY_PATH环境变量来加载正确的libpq.so。

- vsql: symbol lookup error: xxx/vsql: undefined symbol: libpqVersionString

  此问题是由于环境中使用的libpq.so的版本与vsql的版本不匹配导致的（也有可能是环境中存在PostgreSQL的libpq.so），请通过“ldd vsql”命令确认当前加载的libpq.so的版本，并通过修改LD_LIBRARY_PATH环境变量来加载正确的libpq.so。

- vsql: connect to server failed: Connection timed out

  Is the server running on host “xx.xxx.xxx.xxx” and accepting TCP/IP connections on port xxxx?

  此问题是由于网络连接故障造成。请检查客户端与数据库服务器间的网络连接。如果发现从客户端无法PING到数据库服务器端，则说明网络连接出现故障。请联系网络管理人员排查解决。

- vsql: FATAL: permission denied for database “postgres”

  DETAIL: User does not have CONNECT privilege.

  此问题是由于用户不具备访问该数据库的权限，可以使用如下方法解决。

  1、使用管理员用户dbadmin连接数据库。

  ```
  vsql -d postgres -U dbadmin -p 5432
  ```

  2、赋予用户user1访问数据库的权限。

  ```
  GRANT CONNECT ON DATABASE postgres TO user1;
  ```

  实际上，常见的许多错误操作也可能产生用户无法连接上数据库的现象。如用户连接的数据库不存在，用户名或密码输入错误等。这些错误操作在客户端工具也有相应的提示信息。

  - vsql: FATAL: sorry, too many clients already，active/non-active: 2/10/3.

  此问题是由于系统连接数量超过了最大连接数量。请联系数据库DBA进行会话连接数管理，释放无用会话。

表8 查看用户会话连接数的方法

描述	命令

查看指定用户的会话连接数上限。	执行如下命令查看连接到指定用户 USER1的会话连接数上限。其中-1表示没有对用户 user1设置连接数的限制。
	`SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='user1';`
	返回结果如下：
	`rolname | rolconnlimit`
	`---------+--------------`
	`user1 | -1`
	`(1 row)`

查看指定数据库的会话连接数上限。	执行如下命令查看连接到指定数据库 postgres 的会话连接数上限。其中-1表示没有对数据库 postgres 设置连接数的限制。
	`SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='postgres';`
	返回结果如下：
	`datname | datconnlimit`
	`---------+--------------`
	`postgres | -1`
	`(1 row)`

查看指定数据库已使用的会话连接数。	执行如下命令查看指定数据库 postgres 上已使用的会话连接数。其中，1表示数据库 postgres 上已使用的会话连接数。
	`SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='postgres';`
	返回结果如下：
	`count`
	`-------`
	`1`
	`(1 row)`

- 会话状态可以在视图PG_STAT_ACTIVITY中查看。无用会话可以使用函数pg_terminate_backend进行释放。

  例如：

  ```
  select datid,pid,state from pg_stat_activity;
  ```

返回结果如下：

```
datid |       pid       | state
-------+-----------------+--------
17694 | 140381980522240 | active
17694 | 140381812676352 |
17694 | 140381846238976 |
17694 | 140381795895040 | active
17694 | 140381745510144 | idle
17694 | 140381615351552 | active
17694 | 140381581788928 |
17694 | 140381598570240 |
17693 | 140381460035328 | active
(9 rows)
```

其中pid的值即为该会话的线程ID。根据线程ID结束空闲会话140381745510144。

```
SELECT PG_TERMINATE_BACKEND(140381745510144);
```

返回结果如下：

```
pg_terminate_backend
----------------------
t
(1 row)
```

- vsql: wait xxx.xxx.xxx.xxx:xxxx timeout expired

  vsql在向数据库发起连接的时候，会有5分钟超时机制，如果在这个超时时间内，数据库未能正常的对客户端请求进行校验和身份认证，那么vsql会退出当前会话的连接过程，并报出如上错误。

  一般来说，此问题是由于连接时使用的-h参数及-p参数指定的连接主机及端口有误（即错误信息中的xxx部分），导致通信故障；极少数情况是网络故障导致。要排除此问题，请检查数据库的主机名及端口是否正确。

- vsql: could not receive data from server: Connection reset by peer.

  同时，检查数据库主节点日志中出现类似如下日志“ FATAL: cipher file “/data/dbnode/server.key.cipher” has group or world access”，一般是由于数据目录或部分关键文件的权限被误操作篡改导致。请参照其他正常实例下的相关文件权限，修改回来便可。

- vsql: FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  目标数据库主节点的pg_hba.conf里配置了当前客户端IP使用“gss”方式来做认证，该认证算法不支持用作客户端的身份认证，请修改到”sha256”后再试。

> **说明**
>
> - 请不要修改pg_hba.conf中VexDB主机的相关设置，否则可能导致数据库功能故障。

- 建议业务应用部署在VexDB之外，而非VexDB内部。

其他故障

出现因“总线错误”（Bus error）导致的core dump或异常退出。

一般情况下出现此种问题，是进程运行过程中加载的共享动态库（在Linux为.so文件）出现变化，或者进程二进制文件本身出现变化，导致操作系统加载机器的执行码或者加载依赖库的入口发生变化，操作系统出于保护目的将进程杀死，产生core dump文件。

解决此问题，重试便可。同时请尽可能避免在升级等运维操作过程中，在VexDB内部运行业务程序，避免升级时因替换文件产生此问题。

> **说明**
>
> 此故障的core dump文件的可能堆栈是dl_main及其子调用，它是操作系统用来初始化进程做共享动态库加载的。如果进程已经初始化，但是共享动态库还未加载完成，严格意义上来说，进程并未完全启动。

备份与恢复

备份是指将当前的数据库系统、数据文件或日志文件复制到一个专门的备份服务器、活动磁盘或者其他能长期存储数据的介质上作为副本。数据库备份记录了在进行备份这一操作时数据库中所有数据的状态。一旦数据库因意外而遭到损坏，这些备份文件可用来恢复数据库。

恢复是与备份相对应的数据库管理工作，系统进行数据库恢复的过程中，系统会根据备份文件创建数据库并且恢复数据库中的数据。

数据备份是保护数据安全的重要手段之一，数据库的备份与恢复是数据库管理员维护数据安全性和完整性的重要操作。VexDB 数据库支持三种备份恢复类型，以及多种备份恢复方案，备份和恢复过程中提供数据的可靠性保障机制。

备份与恢复类型可分为逻辑备份与恢复、物理备份与恢复。

备份类型介绍

- 逻辑备份与恢复：

  逻辑备份是从数据库中导出数据并写入一个输出文件，该文件的格式与原数据库的文件格式不同，只是原数据库中数据内容的一个映像。逻辑备份时可以指定导出的对象，包括表定义、数据、模式等，也可以导出所有数据库相关信息。逻辑备份通过逻辑导出对数据进行备份，只能基于备份时刻进行数据转储，所以恢复时也只能恢复到备份时保存的数据。

- 物理备份与恢复：

  物理备份是通过对数据库的物理文件（如数据、日志文件等）进行拷贝的方式对数据库进行备份。当数据库发生故障时，可以使用这些备份文件进行还原。通过备份的数据文件及归档日志等文件，可以对数据库进行完全恢复。

VexDB 支持的备份方式

表1 VexDB 支持的备份方式

备份类型	应用场景	支持的介质	工具名称	恢复时间	优点	缺点
逻辑备份与恢复	适合于数据量小的场景。 可以备份单表和多表，单database或所有database。备份后的数据需要使用vsql或者vb_restore工具恢复。数据量大时，恢复时间较长。	磁盘SSD	vb_dump	纯文本格式数据恢复时间长。归档格式数据恢复时间中等。自定义归档格式恢复时间比纯文本格式的恢复时间短。	可灵活选择导出信息，支持用户自定义导出一个数据库或其中的对象（模式、表、视图等）。导出的数据库可以是默认数据库postgres，也可以是自定义数据库。支持多种导出格式。	备份格式指定为纯文本格式时，数据只能通过vsql工具进行恢复，恢复时间较长。
			vb_dumpall	数据恢复时间长。	可以导出VexDB数据库的所有数据，包括默认数据库postgres的数据、自定义数据库的数据、以及VexDB所有数据库公共的全局对象。	只能导出纯文本格式的数据，导出的数据只能通过vsql进行恢复，恢复时间较长。
物理备份与恢复	适用于数据量大的场景，主要用于全量数据备份恢复，也可对整个数据库中的WAL归档日志和运行日志进行备份。		vb_basebackup	恢复时可以直接拷贝替换原有的文件， 或者直接在备份的库上启动数据库，恢复时间快。	备份原理是对服务器数据库文件的二进制进行全量拷贝，因此只能对数据库某一个时间点的时间作备份。	不支持增量备份。结合PITR恢复，可以恢复到全量备份时间点后的某一时间点。
			vb_probackup	恢复时可以直接恢复到某个备份点，在备份的库上启动数据库，恢复时间快。	可备份外部目录的内容，如脚本文件、配置文件、日志文件、dump文件等。支持增量备份、定期备份和远程备份。增量备份时间相对于全量备份时间比较短，只需要备份修改的文件。当前默认备份是数据目录，如果表空间不在数据目录，需要手动指定备份的表空间目录进行备份。	若备份时发生错误，备份程序并不会完全结束并退出，导致无法继续执行备份操作。严格依赖归档备份的有效性，如果其中一个备份集失效，则恢复失败。

备份方案的选择

当需要进行备份恢复操作时，主要从以下四个方面考虑数据备份方案：

- 备份对业务的影响在可接受范围。
- 数据库恢复效率：为尽量减小数据库故障的影响，要使恢复时间减到最少，从而使恢复的效率达到最高。
- 数据可恢复程度：当数据库失效后，要尽量减少数据损失。
- 数据库恢复成本：在现网选择备份策略时参考的因素比较多，如备份对象、数据大小、网络配置等，表2.备份策略典型场景列出了可用的备份策略和每个备份策略的适用场景。

表2 备份策略典型场景

备份策略	关键性能因素	典型数据量	性能规格
数据库实例备份	数据大小网络配置	数据：PB 级对象：约100万个	备份：每个主机80 Mbit/s（NBU/EISOO+磁盘）约90%磁盘 I/O 速率（SSD/HDD）
表备份	表所在模式网络配置（NBU）	数据：10 TB 级	备份：基于查询性能速度+I/O 速度多表备份时，备份耗时计算方式： `auto总时间 = 表数量 x 起步时间 + 数据总量 / 数据备份速度` 其中：磁盘起步时间为5s 左右，NBU 起步时间比 DISK 长（取决于 NBU 部署方案）。数据备份速度为单节点50MB/s 左右（基于1GB 大小的表，物理机备份到本地磁盘得出此速率）。表越小，备份性能更低。

逻辑备份与恢复

- 逻辑备份

  逻辑备份是从数据库中导出数据并将其写入一个输出文件，该文件的格式与原数据库的文件格式不同，只是原数据库中数据内容的一个映像。逻辑备份时可以指定导出的对象，包括表定义、数据、模式等，也可以导出所有数据库相关信息。

- 逻辑恢复

  逻辑恢复是指当数据库发生数据损坏或数据丢失后，使用工具将备份文件导入数据库的过程。

逻辑备份

本章节主要介绍了用于逻辑备份的工具以及它们的用法、注意事项等。

VexDB 用于逻辑备份的工具包括：

- vb_dump
- vb_dumpall

vb_dump

工具介绍

vb_dump 是 VexDB用于导出数据库相关信息的一种逻辑备份工具，用户可以自定义导出一个数据库或其中的对象（模式、表、视图等），不支持导出回收站对象。支持导出的数据库可以是默认数据库 postgres，也可以是自定义数据库。

工具运行产生的日志位于 `$GAUSSLOG/bin/vb_dump` 目录下，每个日志最大 16 MB，最多保留 50 个。如果没有配置环境变量 $GAUSSLOG，则不会产生日志文件。

事务性导出

为保证数据一致性和完整性，vb_dump 会对需要转储的表设置共享锁。如果表在别的事务中设置了共享锁，vb_dump 会等待锁释放后锁定表。如果无法在指定时间内锁定某个表，转储会失败。用户可以通过指定 –lock-wait-timeout 选项，自定义等待锁超时时间。

例如，T1 时刻启动 vb_dump 导出 A 数据库，那么导出数据结果将会是 T1 时刻 A 数据库的数据状态，T1 时刻之后对 A 数据库的修改不会被导出。

导出格式

vb_dump 支持将数据库信息导出至纯文本格式的 SQL 脚本文件或其他归档文件中。

vb_dump可以创建四种不同的导出文件格式，通过-F或者–format=选项指定，如表1.导出文件格式所示：

表1 导出文件格式

格式名称	-F的参数值	说明	建议	对应导入工具
纯文本格式	p	纯文本脚本文件包含SQL语句和命令。命令可以由vsql命令行终端程序执行，用于重新创建数据库对象并加载表数据。	小型数据库，一般推荐纯文本格式。	使用vsql工具恢复数据库对象前，可根据需要使用文本编辑器编辑纯文本导出文件
二进制归档格式	c	一种二进制文件。支持从导出文件中恢复所有或所选数据库对象。	中型或大型数据库，推荐二进制归档格式。	使用vb_restore可以选择要从二进制归档/目录归档/tar归档导出文件中导入相应的数据库对象。
目录归档格式	d	该格式会创建一个目录，该目录包含两类文件，一类是目录文件，另一类是每个表和blob对象对应的数据文件。	-	
tar归档格式	t	tar归档文件支持从导出文件中恢复所有或所选数据库对象。tar归档格式不支持压缩且对于单独表大小应小于8GB。	-	

> **说明**
>
> 可以使用vb_dump工具将文件压缩为目录归档或二进制归档导出文件，减少导出文件的大小。生成目录归档或二进制归档导出文件时，默认进行中等级别的压缩。vb_dump程序无法压缩已归档导出文件。

导出对象

vb_dump 会按顺序检查以下信息是否需要导出，并按照实际导出工作量的数目打印：

- 模式

- 用户自建表
- 插件（Extensions）
- 用户自建函数
- 用户自建类型
- 用户自建PL
- 用户自建聚集函数（Aggregate Function）
- 用户自建操作符
- 用户自建操作符类（Operator Class）
- 用户自建操作符族（Operator Family）
- 用户自建全文检索Parser
- 用户自建全文检索模板
- 用户自建全文检索字典
- 用户自建全文检索配置
- 用户自建外部数据封装器（Foreign Data Wrapper）
- 用户自建外部服务器（Foreign Server）
- 默认权限
- 用户自建排序规则
- 用户自建转换规则
- 用户自建强制转换
- 表继承信息
- 索引
- 约束 支持导出无效索引和约束。在恢复数据时先重建索引和约束，再进行无效化处理，最后插入数据。支持的范围包括：
  - 索引：主键索引、UNIQUE索引；
  - 约束：主键约束、UNIQUE约束、CHECK约束、外键约束。
- 外表约束（foreign table constraint）
- 触发器
- 事件
- 重写规则（Rewrite Rules）

- 行级安全策略
- 导出表的安全策略
- 用户自建包
- 逻辑复制发布相关信息
- 逻辑复制订阅相关信息
- 事件触发器

注意事项

- vb_dump 工具由安装 VexDB数据库的操作系统用户执行。
- 支持在备库上执行 vb_dump，在主库和备库中 vb_dump 的用法完全相同。
- 如果 VexDB有任何本地数据要添加到template1数据库，请谨慎将vb_dump的输出恢复到一个真正的空数据库中，否则可能会因为被添加对象的定义被复制，出现错误。要创建一个无本地添加的空数据库，需从 template0 而非 template1 复制，例如：

  ```
  CREATE DATABASE foo WITH TEMPLATE template0;
  ```

- tar 归档形式的文件大小不得超过 8 GB（tar 文件格式的固有限制）。
- 由 vb_dump 生成的转储文件不包含优化程序用来做执行计划决定的统计数据。因此，最好从某转储文件恢复之后运行 ANALYZE | ANALYSE 以确保最佳效果。转储文件不包含任何 ALTER DATABASE...SET 命令，这些设置由 vb_dumpall 转储，还有数据库用户和其他完成安装设置。
- 禁止修改 -F c/d/t 格式导出的文件和内容，否则可能无法恢复成功。对于 -F p 格式导出的文件，如有需要，可根据需要谨慎编辑导出文件。
- 不支持导出临时表。
- 不支持导出物化视图的表数据。
- 不支持加密导出存储过程和函数。
- 不支持导出数据库中的定时任务。
- vb_dump 工具在进行数据导出时生成的列不会被转储。
- 纯文本格式的 SQL 脚本文件：包含将数据库恢复为其保存时的状态所需的SQL语句。通过 vsql运行该 SQL 脚本文件，可以恢复数据库。即使在其他主机和其他数据库产品上，只要对 SQL 脚本文件稍作修改，也可以用来重建数据库。
- 归档格式文件：包含将数据库恢复为其保存时的状态所需的数据，可以是 tar 格式、目录归档格式或二进制归档格式，详见表1.导出文件格式。该导出结果必须与 vb_restore配合使用来恢复数据库，vb_restore 工具在导入时，系统允许用户选择需要导入的内容，甚至可以在导入之前对等待导入的内容进行排序。详见TOC文件选择性恢复。

命令格式

```
vb_dump [OPTION]... [DBNAME]
```

通用参数包括：

```
General options:
  -f, --file=FILENAME                      output file or directory name
  -F, --format=c|d|t|p                     output file format (custom, directory, tar,
                                           plain text (default))
  -j, --jobs=NUM                           use this many parallel jobs to dump
  -v, --verbose                            verbose mode
  -V, --version                            output version information, then exit
  -Z, --compress=0-9                       compression level for compressed formats
  --lock-wait-timeout=TIMEOUT              fail after waiting TIMEOUT for a table lock
  -?, --help                               show this help, then exit
```

控制转储内容的参数包括：

```
Options controlling the output content:
  -a, --data-only                          dump only the data, not the schema
  -b, --blobs                              include large objects in dump
  -c, --clean                              clean (drop) database objects before recreating
  -C, --create                             include commands to create database in dump
  -E, --encoding=ENCODING                  dump the data in encoding ENCODING
  -g, --exclude-guc=GUC_PARAM              do NOT dump the GUC_PARAM set
  -n, --schema=SCHEMA                      dump the named schema(s) only
  -N, --exclude-schema=SCHEMA              do NOT dump the named schema(s)
  -o, --oids                               include OIDs in dump
  -O, --no-owner                           skip restoration of object ownership in
                                           plain-text format
  -s, --schema-only                        dump only the schema, no data
  -q, --target=VERSION                     dump data format can compatible Gaussdb version (v1 or ..)
  -S, --sysadmin=NAME                      system admin user name to use in plain-text format
  -t, --table=TABLE                        dump the named table(s) only
  -T, --exclude-table=TABLE                do NOT dump the named table(s)
  --include-table-file=FileName            dump the named table(s) only
  --exclude-table-file=FileName            do NOT dump the named table(s)
  --pipeline                               use pipeline to pass the password,
                                           forbidden to use in terminal
  -x, --no-privileges/--no-acl             do not dump privileges (grant/revoke)
  --column-inserts/--attribute-inserts     dump data as INSERT commands with column names
  --disable-dollar-quoting                 disable dollar quoting, use SQL standard quoting
  --disable-triggers                       disable triggers during data-only restore
  --exclude-table-data=TABLE               do NOT dump data for the named table(s)
  --exclude-with                           do NOT dump WITH() of table(s)
  --inserts                                dump data as INSERT commands, rather than COPY
  --no-publications                        do not dump publications
  --no-security-labels                     do not dump security label assignments
  --no-synchronized-snapshots              do not use synchronized snapshots in parallel jobs
  --no-subscriptions                       do not dump subscriptions
  --no-tablespaces                         do not dump tablespace assignments
  --no-unlogged-table-data                 do not dump unlogged table data
  --include-alter-table                    dump the table delete column
  --quote-all-identifiers                  quote all identifiers, even if not key words
  --section=SECTION                        dump named section (pre-data, data, or post-data)
  --serializable-deferrable                wait until the dump can run without anomalies
  --dont-overwrite-file                    do not overwrite the existing file in case of plain, tar and custom format
  --use-set-session-authorization
                                           use SET SESSION AUTHORIZATION commands instead of
                                           ALTER OWNER commands to set ownership
  --exclude-function                       do not dump function and procedure
  --with-encryption=AES128/SM4             dump data is encrypted using AES128 or SM4
  --with-key=KEY                           encryption key, AES128 key must be 16 bytes in length, SM4 key must be
                                           greater than 0 in length, the key means index of encryption key
  --with-userpin=username:password         used to connect to seckms server
  --with-salt=RANDVALUES                   used by vb_dumpall, pass rand value array
  --include-extensions                     include extensions in dump
  --binary-upgrade                         for use by upgrade utilities only
  --binary-upgrade-usermap="USER1=USER2"   to be used only by upgrade utility for mapping usernames
  --non-lock-table                         for use by OM tools utilities only
  --include-depend-objs                    dump the object which depends on the input object
  --exclude-self                           do not dump the input object
  --table-conditions="[tablename; whereclause ]limitclause"        add table conditions for exporting data
  --get-src-def                            get the original defination (include the comment) of view or matview
```

```
  --sess-replic-role=replica                Set the guc parameter session_replication_role in the output file (if and only if format = p).
  --enable-dump-sys-level-objs              enable dumping of system-level objects
```

连接参数包括：

```
Connection options:
  -h, --host=HOSTNAME                       database server host or socket directory
  -p, --port=PORT                           database server port number
  -U, --username=NAME                       connect as specified database user
  -w, --no-password                         never prompt for password
  -W, --password=PASSWORD                   the password of specified database user
  --role=ROLENAME                           do SET ROLE before dump
  --rolepassword=ROLEPASSWORD               the password for role
```

参数说明

- DBNAME

  指定要连接的数据库。“DBNAME”前面不需要加短或长选项。

  - 方式一： 不需要-d，直接指定“DBNAME”为vexdb

    ```
    vb_dump -p port_number vexdb -f dump1.sql
    ```

    其中，port_number 表示数据库端口号。

  - 方式二： 环境变量中可定义缺省数据库，以定义为 vexdb为例：

    ```
    export PGDATABASE=vexdb
    vb_dump -p port_number -f dump1.sql
    ```

    其中，PGDATABASE 为环境变量。

通用参数

- -f, –file=FILENAME

  将输出发送至指定文件或目录。如果省略该参数，则使用标准输出。如果输出格式为(-F c/-F d/-F t)时，必须指定 -f 参数。如果 -f 的参数值含有目录，要求当前用户对该目录具有读写权限，并且不能指定已有目录。

  -f 选项指定的路径可以是绝对路径或相对路径。

- -F, –format=c | d | t | p

选择输出格式。格式如下：

- c | custom：输出一个二进制格式的归档，并且以目录形式输出，作为vb_restore输入信息。该格式是最灵活的输出格式，因为能手动选择，而且能在恢复过程中将归档项重新排序。该格式默认状态下会进行中等级别的压缩。
- d | directory：该格式会创建一个目录，该目录包含两类文件，一类是目录文件，另一类是每个表和 blob 对象对应的数据文件。
- t | tar：输出一个 tar 格式的归档形式，作为vb_restore 输入信息。tar 格式与目录格式兼容；tar 格式归档形式在提取过程中会生成一个有效的目录格式归档形式。但是，tar 格式不支持压缩且对于单独表有 8 GB 的大小限制。此外，表数据项的相应排序在恢复过程中不能更改。
- p | plain：输出一个文本SQL脚本文件（默认）。

- -j,–jobs=NUM

  指定备份线程并发数。仅支持目录格式并行备份。

  并发数支持的范围为1~INT_MAX，但实际上无法达到最大值。并发数受以下因素的限制：（详见常见问题的问题1）。

  - open files数量
  - 内存限制
  - 进程最大数

- -v, –verbose

  开启冗长模式。该选项将导致 vb_dump 向转储文件输出详细的对象注解和启动/停止次数，向标准错误流输出处理信息。

- -V, –version

  打印vb_dump版本，然后退出。

- -Z, –compress=0-9

  指定使用的压缩比级别。

  取值范围：0-9

  - 0表示无压缩。
  - 1表示压缩比最小，处理速度最快。
  - 9表示压缩比最大，处理速度最慢。

> **说明**
>
> 针对二进制归档格式，该选项指定单个表数据片段的压缩，默认方式是以中等级别进行压缩。对于 gzip 格式，默认的压缩等级为 6。 tar 归档格式和纯文本格式目前不支持压缩。

- –lock-wait-timeout=TIMEOUT

  请勿在转储刚开始时一直等待以获取共享表锁。如果无法在指定时间内锁定某个表，就选择失败。可以以任何符合 SET statement_timeout 的格式指定超时时间。

- -?, –help

  显示 vb_dump 命令行参数帮助，然后退出。

转储参数

- -a, –data-only

  只输出数据，不输出模式（数据定义）。转储表数据、大对象和序列值。

  - 该选项不能和 –schema-only 同时使用。
  - 该选项不能和 -c,–clean 同时使用。

- -b, –blobs

  在转储中包括大对象。

  该参数为扩展预留接口，不建议使用。

- -c, –clean

  在将创建数据库对象的指令输出到备份文件之前，先将清理（删除）数据库对象的指令输出到备份文件中。（如果目标数据库中没有任何对象，vb_restore 工具可能会输出一些提示性的错误信息。）

  - 该选项只对文本格式有意义。该选项仅文本格式时生效。若需要对归档格式执行前进行 clean，可以在执行 vb_restore 时指定。
  - 该选项不能和 -a,–data-only 同时使用。

- -C, –create

  导入数据库之前会先使用 CREATE DATABASE 创建数据库。（指定该选项后，-d指定的数据库仅用以执行 CREATE DATABASE 命令，所有数据依然会导入到创建的数据库中。）

- -E, –encoding=ENCODING

  以指定的字符集编码创建转储。默认情况下，以数据库编码（GUC 参数 CLIENT_ENCODING）创建转储。（得到相同结果的另一个办法是将环境变量“PGCLIENTENCODING”设置为所需的转储编码。）

- -g, –exclude-guc

  不转储 GUC_PARAM 集。 该参数为扩展预留接口，不建议使用。

- -n, –schema=SCHEMA

只转储与模式名称匹配的模式，此选项包括模式本身和所有它包含的对象。如果该选项没有指定，所有在目标数据库中的非系统模式将会被转储。

写入多个 -n 选项来选择多个模式。此外，根据 vsql的 -d 命令所使用的相同规则，模式参数可被理解成一个 pattern，所以多个模式也可以通过在该 pattern 中写入通配符来选择。使用通配符时，注意给 pattern 打引号，防止 shell 扩展通配符。

- 指定 -n 时，vb_dump 不会转储已选模式所依赖任何其他数据库对象。因此，无法保证某个指定模式的转储结果能够自行成功地储存到一个空数据库中。例如：
- sch1 中的视图 v1 依赖于 sch2 模式的表 t2，若在 -n 选项中指定了 sch1，则 v1 定义所依赖的 t2 并不会被转储。
- 指定 -n 时，非模式对象不会被转储。
- 指定模式中的同义词也会被导出。
- 不支持使用-n显式指定pg_catalog、information_schema等系统级schema，除非同时指定了–enable-dump-sys-level-objs 选项。
- -n 和 -N 的数量加起来有上限，在 34 万到 35 万个之间，略小于 35万个。
- 转储支持多个模式的转储。多次输入 -n schemaname 转储多个模式。

  例如：

  ```
  vb_dump -h host_name -p port_number vexdb -f backup/bkp_shl2.sql -n sch1 -n sch2
  ```

  上例中，sch1 和 sch2 模式会被转储。

- -N, –exclude-schema=SCHEMA

不转储任何与模式 pattern 匹配的模式。pattern 将参照针对 -n 的相同规则来理解。可以通过输入多次 -N，不转储与任何pattern匹配的模式。

- 同时输入 -n 和 -N 时，会转储与至少一个 -n 选项匹配、与 -N 选项不匹配的模式。
- 如果指定了 -N 而未指定 -n，则不转储常规转储中与 -N 匹配的模式。
- 转储过程支持排除多个模式，输入 -N exclude schema name 排除多个模式。

  例如：

  ```
  vb_dump -h host_name -p port_number vexdb -f backup/bkp_shl2.sql -N sch1 -N sch2
  ```

  在上面这个例子中，sch1 和 sch2 在转储过程中会被排除。

- -o, –oids

转储每个表的对象标识符（OIDs），作为表的一部分数据。该选项用于应用以某种方式参照了 OID 列的情况。如果不是以上这种情况，请勿使用该选项。

该选项不能和 –inserts/–column-inserts 同时使用，因为INSERT命令不能设置 OIDS。

- -O, –no-owner

  不输出设置对象的归属这样的命令，以匹配原始数据库。默认情况下，vb_dump会发出 ALTER OWNER 或 SET SESSION AUTHORIZATION 语句设置所创建的数据库对象的归属。在执行脚本过程中，默认不设置属主的行为可能会使某些语句执行不成功，除非是由系统管理员触发（或是拥有脚本中所有对象的同一个用户）。通过指定 -O，该脚本会授予该用户拥有所有对象的权限，使任何用户都能执行。

  该选项仅文本格式时生效。若需要对归档格式执行前进行 clean，可以在执行vb_restore 时指定。

- -s, –schema-only

  只转储对象定义（模式），而非数据。

  - 该选项不能和 -a,–data-only 同时使用。
  - 若在导出时指定该参数，则在使用vb_restore进行恢复时也必须指定仅导入模式（即指定-s参数）。

- -q, –target

  该选项无实际意义。

- -S, –sysadmin=NAME

  该参数为扩展预留接口，不建议使用。

- -t, –table=TABLE

  指定转储的表（或视图、或序列、或外表）对象列表，可以使用多个-t选项来选择多个表，也可以使用通配符指定多个表对象。当使用通配符指定多个表对象时，注意给 pattern 打引号，防止 shell 扩展通配符。当使用 -t 时，-n 和 -N 会失效，这是因为由 -t 选择的表的转储不受那些选项的影响。

  - -t 参数选项个数必须小于等于 100。

  -如果 -t 参数选项个数大于 100，建议使用参数 –include-table-file 来替换。

  - 当 -t 已指定时，vb_dump 不会转储已选表所附着的任何其他数据库对象。因此，无法保证某个指定表的转储结果能够自行成功地储存到一个空数据库中。
  - -t tablename 只转储在默认搜索路径中可见的表。-t *.tablename 转储数据库下所有模式下的 tablename 表。-t schema.table 转储特定模式中的表。
  - 对于表名中包含大写字母的表，在使用 -t 参数指定导出时，需对表名添加英文双引号并使用反斜杠 进行转义。例如，如对于表 “abC”，导出需指定 -t "abC"；对于表 schema.”abC”，导出需指定 -t schema."abC"。
  - 不支持-t显式指定导出系统表和系统视图，除非同时指定了–enable-dump-sys-level-objs 选项。

    例如：

    ```
    vb_dump -h host_name -p port_number vexdb -f backup/bkp_shl2.sql -t schema1.table1 -t schema2.table2
    ```

在上面这个例子中，schema1.table1 和 schema2.table2 会被转储。

- -T, –exclude-table=TABLE

  不转储的表（或视图、或序列、或外表）对象列表，可以使用多个 -T 选项来选择多个表，也可以使用通配符指定多个表对象。当同时输入 -t 和 -T 时，会转储在 -t 列表中，而不在 -T 列表中的表对象。

  例如：

  ```
  vb_dump -h host_name -p port_number vexdb -f backup/bkp_shl2.sql -T table1 -T table2
  ```

  在上面这个例子中，table1 和 table2 在转储过程中会被排除。

- –include-table-file=FILENAME

  指定需要 dump 的表文件。

  不支持文件中带有系统表和系统视图，除非同时指定了–enable-dump-sys-level-objs 选项。

- –exclude-table-file=FILENAME

  指定不需要 dump 的表文件。用法详见示例 5。

  同 –include-table-file，其内容格式如下：schema1.table1 schema2.table2 ...

- –pipeline

  使用管道传输密码，禁止在终端使用。

- -x, –no-privileges|–no-acl

  防止转储访问权限（授权/撤销命令）。

- –column-inserts|–attribute-inserts

  以 INSERT 命令带列名（INSERT INTO表（列、...）值...）方式导出数据。这会导致恢复缓慢。但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。

  该选项不能和 -o,–oids 同时使用，因为 INSERT 命令不能设置 OIDS。

- –disable-dollar-quoting

  该选项将禁止在函数体前使用美元符号$，并强制使用SQL标准字符串语法对其进行引用。

- –disable-triggers

该参数为扩展预留接口，不建议使用。

- –exclude-table-data=TABLE

  指定不转储任何匹配表 pattern 的表这方面的数据。依照针对 -t 的相同规则理解该 pattern。可多次输入 –exclude-table-data 来排除匹配任何 pattern 的表。当用户需要特定表的定义但不需要其中的数据时，这个选项很有帮助。排除数据库中所有表的数据，参见 –schema-only。

- –exclude-with

  导出的表定义，末尾不添加 WITH(orientation=row，compression=on）这样的描述。

- –inserts

  发出 INSERT 命令（而非 COPY 命令）转储数据。这会导致恢复缓慢。但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。注意：如果重排列顺序，可能会导致整个恢复失败。列顺序改变时，–column-inserts 选项不受影响，虽然会更慢。

  该选项不能和 -o,–oids 同时使用，因为 INSERT 命令不能设置 OIDS。

- –no-publications

  不转储发布。

- –no-security-labels

  该参数为扩展预留接口，不建议使用。

- –no-synchronized-snapshots

  并行作业中不使用同步快照。

- –no-subscriptions

  不转储订阅。

- –no-tablespaces

  不输出选择表空间的命令。使用该选项，无论默认表空间是哪个，在恢复过程中所有对象都会被创建。

  该选项只对文本格式有意义。针对归档格式，可以调用 vb_restore 时指定选项。

- –no-unlogged-table-data

  不转储未记录的表数据。

  该参数为扩展预留接口，不建议使用。

- –include-alter-table

转储表删除列。该选项会记录列的删除。

- –quote-all-identifiers

  强制对所有标识符加引号。

  为了向后续版本迁移，且其中可能涉及引入额外关键词，在转储相应数据库时该选项会有帮助。

- –section=SECTION

  指定已转储的名称区段（pre-data、data 和 post-data）。

- –serializable-deferrable

  转储过程中使用可串行化事务，以确保所使用的快照与之后的数据库状态一致；要实现该操作需要在无异常状况的事务流中等待某个点，因为这样才能保证转储成功，避免引起其他事务出现 serialization_failure 要重新再做。但是该选项对于灾难恢复没有益处。对于在原始数据库进行升级的时候，加载一个数据库的拷贝作为报告或其他只读加载共享的转储是有帮助的。没有这个选项，转储会反映一个与任何事务最终提交的序列化执行不一致的状态。如果当 vb_dump 启动时，读写事务仍处于非活动状态，即便使用该选项也不会对其产生影响。如果读写事务处于活动状态，转储的开始时间可能会延迟一段不确定的时间。

- –dont-overwrite-file

  文本、tar 以及二进制格式情况下会重写现有文件。这对目录格式不适用。

  例如：设想这样一种情景，即当前目录下 backup.sql 已存在。如果在输入命令中输入 -f backup.sql 选项时，当前目录恰好也生成 backup.sql，文件就会被重写。

  如果备份文件已存在，且输入 –dont-overwrite-file 选项，则会报告附带“转储文件已经存在”的错误信息。

  ```
  vb_dump -p port_number vexdb -f backup.sql -F plain --dont-overwrite-file
  ```

- –use-set-session-authorization

  输出符合 SQL 标准的 SET SESSION AUTHORIZATION 命令而不是 ALTER OWNER 命令来确定对象所有权。这样令转储更加符合标准，但是如果转储文件中的对象的历史有些问题，那么可能不能正确恢复。并且，使用 SET SESSION AUTHORIZATION 的转储需要数据库系统管理员的权限才能转储成功，而 ALTER OWNER 需要的权限则低得多。

- –exclude-function

  不导出函数和存储过程。

- –with-encryption=AES128/SM4

  指定转储数据使用的加密方式，支持 AES128 和 SM4 算法。加密导出仅支持导出格式为纯文本（-F p）。

  - 支持在使用江南天安加密机进行加密导出时指定纯文本以外的格式。即导出格式可以是目录格式（d），二进制归档格式（c），tar格式（t）或者纯文本格式（p）。
  - 该选项必须与 –with-key 和 –with-salt 同时使用。

- AES128 加密算法使用的是 CBC 模式，SM4 加密算法使用的是 CTR 模式。
- 当使用三未信安的加密算法时，必须使用 –with-userpin 指定创建密钥时的 KMS 用户信息。

- –with-key=KEY

  密钥名称或者密钥索引。

  AES128 密钥规则如下：

  - 钥长度为 8~16 个字符。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。

- –with-userpin=username:password

  连接 seckms 加密服务器的 username:password 信息。

- –with-salt=RANDVALUES

  加密导出时使用此参数传递随机盐值。必须为 16 个字符的字符串。

- –include-extensions

  在转储中包含扩展。

- –binary-upgrade

  该参数为扩展预留接口，不建议使用。

  –binary-upgrade 和 –binary-upgrade-usermap 必须一起使用。

- –binary-upgrade-usermap= “USER1=USER2”

  该参数为扩展预留接口，不建议使用。

- –non-lock-table

  该参数为扩展预留接口，不建议使用。

- –include-depend-objs

  备份结果包含依赖于指定对象的对象信息。该参数需要同 -t/–include-table-file 参数关联使用才会生效。

  该参数需要同 -t/–include-table-file 参数关联使用才会生效。

- –exclude-self

  备份结果不包含指定对象自身的信息。该参数需要同 -t/–include-table-file 参数关联使用才会生效。

–exclude-self 必须同 –include-depend-objs 一起使用。

- –table-conditions= “[tablename; whereclause]limitclause”

  导出格式为纯文本格式时，仅导出部分数据：

  - 导出整个指定库/模式时，使用 limit 子句限制每张表的导出行数。limit n 意为导出的每张表仅包含 n 行 copy（insert）数据。
  - 单表 / 多表导出时，使用 where 子句 /limit 子句限制表的导出条件或导出行数。参考示例 7。
  - whereclause 用于指定导出条件，仅导出符合条件的结果集数据。
  - limitclause 用于限制导出行数，limit n 意为导出（符合条件的）前 n 行。

  > **说明**
  >
  > - 指定tablename时表示单表数据过滤条件，不指定 tablename 时表示多表过滤条件。
  > - 支持的单表条件的筛选关键词包括：where、order、limit、offset、fetch。
  > - 支持的多表条件的筛选关键词包括：limit、offset、fetch。
  > - 支持使用多个–table-conditions分别指定每张表的导出条件。
  > - 当指定了单表 limit 时，不支持再指定多表的 limit 条件。
  > - 当指定了 -o 选项后，对于有 OIDS 的表，将会忽略 –table-conditions 中的条件并导出全部表数据。

- –get-src-def

  导出视图或者物化视图的原始定义（包含注释）。

  未指定该选项时，默认通过反序列化的方式获取视图或物化视图的定义。此方式可以确保在视图或其依赖对象的定义发生修改的情况下，仍能正确获取到当前的有效视图定义。

- –sess-replic-role=replica

  当备份格式为p（文本格式）时，在输出文件中设置guc参数session_replication_role为replica，以避免触发器行为、重写规则对恢复过程的影响。

  由于session_replication_role是SUSET类型参数，只允许数据库管理员用户进行修改，因此恢复时不应使用普通用户，以免因权限不足导致无法修改参数。

- –enable-dump-sys-level-objs

  允许导出系统级对象。

  该选项必须与-n/–schema， -t/–table或者–include-table-file结合使用，单独指定既不生效也不报错：

  - 与-n/–schema结合使用，表示允许导出-n/–schema指定的系统命名空间（若存在）下的内容。
  - 与-t/–table结合使用，表示允许导出-t/–tablez指定的系统表/系统视图（若存在）。

- 与–include-table-file结合使用，表示允许导出文件中指定的系统表/系统视图（若存在）。

连接参数

- -h, –host=HOSTNAME

  指定主机名称。如果数值以斜杠开头，则被用作到 Unix 域套接字的路径。缺省从 PGHOST 环境变量中获取（如果已设置），否则，尝试一个 Unix 域套接字连接。该参数只针对 VexDB外，对VexDB内本机只能用 127.0.0.1。

  例如：主机名

  环境变量：PGHOST

- -p, –port=PORT

  指定主机端口。在开启线程池情况下，建议使用 pooler port，即主机端口+1。

  环境变量：PGPORT

- -U, –username=NAME

  指定所连接主机的用户名。不指定连接主机的用户名时，用户默认系统管理员。

  环境变量：PGUSER

- -w, –no-password

  不出现输入密码提示。如果主机要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。

- -W, –password=PASSWORD

  指定用户连接的密码。如果主机的认证策略是 trust，则不会对系统管理员进行密码验证，即无需输入 -W 选项；如果没有 -W 选项，并且不是系统管理员，工具会提示用户输入密码。

- –role=ROLENAME

  指定创建转储使用的角色名。选择该选项，会使 vb_dump 连接数据库后，发起一个 SET ROLE 角色名命令。当所授权用户（由 -U 指定）没有 vb_dump 要求的权限时，该选项会起到作用，即切换到具备相应权限的角色。某些安装操作规定不允许直接以超系统管理员身份登录，而使用该选项能够在不违反该规定的情况下完成转储。

- –rolepassword=ROLEPASSWORD

  指定角色名的密码。

  该选项必须和 –role 一起使用。

示例

示例1

执行vb_dump，导出 vexdb 数据库全量信息，导出的 backup1 文件为目录格式。备份命令如下：

```
vb_dump -U vexdb -W Vbase@123 -f $PGDATA/backup1 -p 5432 vexdb -F d
```

备份成功，打印信息如下：

```
vb_dump[port='5432'][vexdb][2025-08-21 15:54:11]: The total objects number is 529.
vb_dump[port='5432'][vexdb][2025-08-21 15:54:11]: [100.00%] 529 objects have been dumped.
vb_dump[port='5432'][vexdb][2025-08-21 15:54:11]: dump database vexdb successfully
vb_dump[port='5432'][vexdb][2025-08-21 15:54:11]: total time: 1918  ms
```

示例2

执行 vb_dump，导出 vexdb数据库全量信息，导出的文件为二进制归档格式，并指定压缩等级为 6。备份命令如下：

```
vb_dump -U vexdb -W Vbase@123 -f $PGDATA/backup2.dmp -Z 6 -p 5432 vexdb -F c
```

备份成功，打印信息如下：

```
vb_dump[port='5432'][vexdb][2025-08-21 16:19:45]: The total objects number is 523.
vb_dump[port='5432'][vexdb][2025-08-21 16:19:45]: [100.00%] 523 objects have been dumped.
vb_dump[port='5432'][vexdb][2025-08-21 16:19:45]: object table1 start dumping by pid 29995
vb_dump[port='5432'][vexdb][2025-08-21 16:19:45]: object table1 finish dumping by pid 29995
vb_dump[port='5432'][vexdb][2025-08-21 16:19:45]: dump database test successfully
vb_dump[port='5432'][vexdb][2025-08-21 16:19:45]: total time: 1565  ms
```

示例3

执行 vb_dump，指定 -a 选项，仅导出 vexdb数据库的数据。备份命令如下：

```
vb_dump -U vexdb -W Vbase@123 -f $PGDATA/backup3.tar -a -p 5432 vexdb -F t
```

打印信息如下：

```
vb_dump[port='5432'][vexdb][2025-08-21 17:22:59]: object table1 start dumping by pid 6065
vb_dump[port='5432'][vexdb][2025-08-21 17:22:59]: object table1 finish dumping by pid 6065
vb_dump[port='5432'][vexdb][2025-08-21 17:22:59]: dump databasevexdbsuccessfully
vb_dump[port='5432'][vexdb][2025-08-21 17:22:59]: total time: 1803  ms
```

示例4

使用vb_dump备份，指定不同转储参数。

1. 使用vsql工具连接至vexdb数据库。

```
vsql -d vexdb -p 5432 -r
```

2. 创建测试数据库test4。

```
create database test4;
```

3. 切换至数据库test4中。

```
\c test4;
```

4. 创建测试表并插入数据。

```
create table table1(name text,id varchar(2));
create table table2(name text,id varchar(2));
create table table3(name text,id varchar(2));

insert into table1 values('alice',1);
insert into table2 values('jack',2);
insert into table3 values('bob',3);
```

5. 执行vb_dump备份操作。
   - 备份数据库test4下的全部表，备份为二进制归档格式。

```
vb_dump test4 -p 5432 -t public.* -F c -f $PGDATA/backup4_1.dmp
```

   - 备份整个test4数据库，备份为纯文本格式，备份时指定-C，表示备份文件backup4_2.sql是以创建数据库和连接到创建的数据库的命令开始的。

```
vb_dump test4 -p 5432 -C -F p -f $PGDATA/backup4_2.sql
```

- 备份test4下全部表，备份格式为纯文本格式，备份时指定-c，表示在将创建数据库对象的指令输出到备份文件之前，先将清理（删除）数据库对象的指令输出到备份文件backup4_3.sql中：

```
vb_dump test4 -p 5432 -c -t public.* -F p -f $PGDATA/backup4_3.sql
```

- 备份test4数据库下除了table3以外的表，备份的目录格式为tar包。

```
vb_dump test4 -p 5432 -t public.* -T public.table3 -F t -f $PGDATA/backup4_4.tar
```

- 备份test4所有表，指定-s只转储对象定义（模式）。

```
vb_dump test4 -p 5432 -t public.* -s -F p -f $PGDATA/backup4_5.sql
```

示例5

使用vb_dump导出数据库信息时，使用–exclude-table-file选项指定文件，不导出该文件中指定的表信息。

1. 使用vsql工具连接至vexdb数据库。
2. 创建测试数据库test5。

```
create database test5;
```

3. 切换至数据库test5下。

```
\c test5;
```

4. 创建测试表并插入测试数据。

```
create table table1(name text,id varchar(2));
create table table2(name text,id varchar(2));
create table table3(name text,id varchar(2));

insert into table1 values('alice',1);
insert into table2 values('jack',2);
insert into table3 values('bob',3);
```

5. 退出vsql程序，创建文本文件temp.sql，填写不导出的表的表名，此处以table2为例。

```
echo "public.table2" > $PGDATA/temp.sql
```

6. 使用如下备份命令，备份数据库test5下除了table2以外的表。

```
vb_dump -U vexdb -W Vbase@123 -p 5432 test5 --exclude-table-file=$PGDATA/temp.sql -f $PGDATA/backup5.sql
```

示例6

执行vb_dump，仅导出依赖于指定表testtable的视图信息。然后创建新的testtable表，再恢复依赖其上的视图。

1. 使用vsql工具连接至vexdb数据库，进入数据库创建测试表testtable。

```
create table testtable(id int);
```

2. 退出vsql程序，备份仅依赖于testtable的视图。

```
vb_dump -s -p 5432 vexdb -t PUBLIC.testtable --include-depend-objs --exclude-self -f $PGDATA/backup6.sql -F p
```

3. 使用vsql工具连接至vexdb数据库，修改testtable名称。

```
ALTER TABLE PUBLIC.testtable RENAME TO testtable_bak;
```

4. 创建新的testtable表。

```
CREATE TABLE PUBLIC.testtable(a int, b int, c int);
```

5. 退出vsql程序，还原依赖于testtable的视图。

```
vsql -p 5432 vexdb -r -f $PGDATA/backup6.sql
```

示例7

使用–table conditions限定导出条件。

1. 创建测试用数据库dbtest_07。

```
create database dbtest_07;
```

2. 创建测试表t_backup1。

```
create table t_backup1(id int,col text);
```

3. 插入测试数据。

```
insert into t_backup1 values(1,'张三');
insert into t_backup1 values(2,'李四');
insert into t_backup1 values(3,'王五');
insert into t_backup1 values(4,'赵六');
```

4. 使用–table conditions限定导出条件，导出数据。

```
vb_dump vexdb -t t_backup1 --table-condition="t_backup1; where id > 1 limit 1" -f $PGDATA/backup1.sql  #命令1
vb_dump vexdb -t t_backup1 --table-condition="t_backup1; where id > 1" -f $PGDATA/backup2.sql  #命令2
vb_dump vexdb -t t_backup1 --table-condition="t_backup1; limit 1" -f $PGDATA/backup3.sql  #命令3
```

- 命令1：导出id大于1的行，仅导出第一条符合要求的记录。
- 命令2：导出id大于1的行，不限制记录条数。
- 命令3：仅导出第一条符合要求的记录。

vb_dumpall

工具介绍

vb_dumpall 是 VexDB 用于导出所有数据库相关信息的工具。它可以导出 VexDB 数据库的所有数据，包括默认数据库 postgres 的数据、自定义数据库的数据以及 VexDB 所有数据库公共的全局对象。

vb_dumpall 工具支持导出完整一致的数据。例如，T1 时刻启动 vb_dumpall 导出 VexDB 数据库，那么导出数据的结果将会是 T1 时刻该 VexDB 数据库的数据状态，T1 时刻之后对 VexDB 的修改不会被导出。

vb_dumpall 在导出 VexDB 所有数据库时分为两部分：

- vb_dumpall自身对所有数据库公共的全局对象进行导出，包括有关数据库用户和组、表空间以及属性（例如，适用于数据库整体的访问权限）信息。
- vb_dumpall通过调用vb_dump来完成VexDB中各数据库的SQL脚本文件导出，该脚本文件包含将数据库恢复为其保存时的状态所需要的全部SQL语句。

以上两部分导出的结果为纯文本格式的SQL脚本文件，使用vsql运行该脚本文件即可恢复VexDB数据库。

工具运行产生的日志位于 `$GAUSSLOG/bin/vb_dumpall` 目录下，每个日志最大 16 MB，最多保留 50 个。如果没有配置环境变量 $GAUSSLOG，则不会产生日志文件。

注意事项

- vb_dumpall工具由安装VexDB数据库的操作系统用户执行。
- vb_dumpall工具在进行数据导出时，其他用户可以访问VexDB数据库（读或写）。
- 由于vb_dumpall内部调用vb_dump，所以一些诊断信息可以参考vb_dump。
- vb_dumpall仅支持纯文本格式导出，所以只能使用vsql工具恢复转储内容。
- 成功恢复后，建议在每个数据库上运行ANALYZE | ANALYSE，优化程序提供有用的统计数据。
- vb_dumpall恢复前需要所有必要的表空间目录为空；否则，对于处在非默认位置的数据库，数据库创建会失败。（可以通过vsql工具的。）
- 不支持导出数据库中的定时任务。
- vb_dumpall工具在进行数据导出时生成的列不会被转储。
- 为了保证数据一致性和完整性，vb_dumpall会对需要转储的表设置共享锁。如果某张表在别的事务中设置了共享锁，vb_dumpall会等待此表的锁释放后锁定此表。如果无法在指定时间内锁定某张表，转储会失败。用户可以通过指定–lock-wait-timeout选项，自定义等待锁超时时间。
- 由于vb_dumpall读取所有数据库中的表，因此必须以VexDB管理员身份进行连接，才能导出完整文件。在使用vsql执行脚本文件导入时，同样需要管理员权限，以便添加用户和组以及创建数据库。

命令格式

```
vb_dumpall [OPTION]...
```

常用参数：

```
General options:
  -f, --file=FILENAME                      output file name
  -v, --verbose                            verbose mode
  -V, --version                            output version information, then exit
  --lock-wait-timeout=TIMEOUT              fail after waiting TIMEOUT for a table lock
  -?, --help                               show this help, then exit
```

控制转储内容的参数包括：

```
Options controlling the output content:
  -a, --data-only                          dump only the data, not the schema
  -c, --clean                              clean (drop) databases before recreating
  -g, --globals-only                       dump only global objects, no databases
  -o, --oids                               include OIDs in dump
  -O, --no-owner                           skip restoration of object ownership
  -r, --roles-only                         dump only roles, no databases or tablespaces
  -s, --schema-only                        dump only the schema, no data
  -S, --sysadmin=NAME                      system admin user name to use in the dump
  -t, --tablespaces-only                   dump only tablespaces, no databases or roles
  -x, --no-privileges                      do not dump privileges (grant/revoke)
  --column-inserts/--attribute-inserts     dump data as INSERT commands with column names
  --disable-dollar-quoting                 disable dollar quoting, use SQL standard quoting
  --disable-triggers                       disable triggers during data-only restore
  --inserts                                dump data as INSERT commands, rather than COPY
  --no-publications                        do not dump publications
  --no-security-labels                     do not dump security label assignments
  --no-tablespaces                         do not dump tablespace assignments
  --no-subscriptions                       do not dump subscriptions
  --no-unlogged-table-data                 do not dump unlogged table data
  --include-alter-table                    dump the table delete column
  --quote-all-identifiers                  quote all identifiers, even if not key words
  --dont-overwrite-file                    do not overwrite the existing file
  --use-set-session-authorization          use SET SESSION AUTHORIZATION commands instead of
                                           ALTER OWNER commands to set ownership
  --with-encryption=AES128/SWS-KMS-SM4/TAS-KMS-SM4
                                           dump data is encrypted using AES128 or SWS-KMS-SM4 or TAS-KMS-SM4
  --with-key=KEY                           encryption key, AES128 key must be 16 bytes in length, SWS-KMS-SM4 key must be
                                           greater than 0 in length, if use TAS-KMS-SM4,the key means index of encryption key
  --with-salt=RANDVALUES                   random values for encrypt
  --include-extensions                     include extensions in dumpall
  --include-templatedb                     include dumping of template database also
  --pipeline                               use pipeline to pass the password,forbidden to use in terminal
  --binary-upgrade                         for use by upgrade utilities only
  --binary-upgrade-usermap="USER1=USER2"   to be used only by upgrade utility for mapping usernames
  --non-lock-table                         for use by OM tools utilities only
  --tablespaces-postfix                    to be used only by upgrade utility for adding the postfix name specified for all the tablespaces
  --parallel-jobs                          number of parallel jobs to dump databases
  --skip-initial-user                      skip the initial user when dump role
  --get-src-def                            get the original defination (include the comment) of view or matview
  --sess-replic-role=replica               Set the guc parameter session_replication_role in the output file (if and only if format = p).
```

连接参数包括：

```
Connection options:
  -h, --host=HOSTNAME                      database server host or socket directory
  -l, --database=DBNAME                    alternative default database
  -p, --port=PORT                          database server port number
  -U, --username=NAME                      connect as specified database user
  -w, --no-password                        never prompt for password
  -W, --password=PASSWORD                  the password of specified database user
  --role=ROLENAME                          do SET ROLE before dump
  --rolepassword=ROLEPASSWORD              the password for role
```

参数说明

通用参数

- -f, –filename=FILENAME：

  将输出发送至指定文件。如果这里省略，则使用标准输出。

  –parallel-jobs和-f/–filename必须同时使用。

- -v, –verbose

  开启冗长模式。该选项将导致vb_dumpall向转储文件输出详细的对象注解和启动/停止次数，向标准错误流输出处理信息。

- -V, –version

  打印vb_dumpall版本，然后退出。

- –lock-wait-timeout=TIMEOUT

  请勿在转储刚开始时一直等待以获取共享表锁。如果无法在指定时间内锁定某个表，就选择失败。可以以任何符合SET statement_timeout的格式指定超时时间。

- -?, –help

  显示vb_dumpall命令行参数帮助，然后退出。

转储参数

- -a, –data-only

  只转储数据，不转储模式（数据定义）。

  - 该选项不能和-s/–schema-only同时使用。
  - 该选项不能和-r/–roles-only同时使用。
  - 该选项不能和-t/–tablespaces-only同时使用。
  - 该选项不能和-g/–globals-only同时使用。

- -c, –clean

  在重新创建数据库之前，执行SQL语句清理（删除）这些数据库。针对角色和表空间的转储命令已添加。

- -g, –globals-only

只转储全局对象（角色和表空间），无数据库。

- 该选项和-a/–data-only不能同时使用。
- 该选项和-r/–roles-only不能同时使用。
- 该选项和-t/–tablespaces-only不能同时使用。

- -o, –oids

  转储每个表的对象标识符（OIDs），作为表的一部分数据。该选项用于应用以某种方式参照了OID列的情况。如果不是以上这种情况，请勿使用该选项。

- -O, –no-owner

  不输出设置对象的归属这样的命令，以匹配原始数据库。默认情况下，vb_dumpall会发出ALTER OWNER或SET SESSION AUTHORIZATION语句设置所创建的模式元素的所属。如果脚本正在运行，该语句不会执行成功，除非是由系统管理员触发（或是拥有脚本中所有对象的同一个用户）。通过指定-O，编写一个任何用户都能存储的脚本，且该脚本会授予该用户拥有所有对象的权限。

  - 该选项仅文本格式时生效。若需要对归档格式执行前进行 clean，可以在执行vb_restore 时指定。
  - 该选项对使用FDW相关语法的部分情况无效。涉及的语法包括：

    ```
    CREATE SERVER ... OWNER TO ...
    CREATE USER MAPPING FOR ... SERVER ... OWNER TO ...
    CREATE FOREIGN TABLE ... SERVER ... OWNER TO ...
    ```

- -r, –roles-only

  只转储角色，不转储数据库或表空间。

  - 该选项和-g/–globals-only不能同时使用。
  - 该选项和-t/–tablespaces-only不能同时使用。
  - 该选项和-a/–data-only不能同时使用。

- -s, –schema-only

  只转储对象定义（模式），而非数据。

  该选项和-a/–data-only不能同时使用。

- -S, –sysadmin=NAME

  该参数为扩展预留接口，不建议使用。

- -t, –tablespaces-only

只转储表空间，不转储数据库或角色。

该选项和-a/–data-only不能同时使用。

- -x, –no-privileges

  防止转储访问权限（授权/撤销命令）。

- –column-inserts|–attribute-inserts

  以INSERT命令带列名（INSERT INTO表（列、...）值...）方式导出数据。这会导致恢复缓慢。但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。

- –disable-dollar-quoting

  该选项将禁止在函数体前使用美元符号$，并强制使用SQL标准字符串语法对其进行引用。

- –disable-triggers

  该参数为扩展预留接口，不建议使用。

- –inserts

  发出INSERT命令（而非COPY命令）转储数据。这会导致恢复缓慢。注意：如果重排列顺序，可能会导致恢复整个失败。–column-inserts选项更加安全，虽然可能更慢些。

- –no-publications

  不转储发布。

- –no-security-labels

  该参数为扩展预留接口，不建议使用。

- –no-tablespaces

  请勿输出创建表空间的命令，也请勿针对对象选择表空间。使用该选项，无论默认表空间是哪个，在恢复过程中所有对象都会被创建。

- –no-subscriptions

  不转储订阅。

- –no-unlogged-table-data

  不转储未记录的表数据。

  该参数为扩展预留接口，不建议使用。

- –include-alter-table

导出表中已删除的列信息。

- –quote-all-identifiers

  强制对所有标识符加引号。为了向后续版本迁移，且其中可能涉及引入额外关键词，在转储相应数据库时该选项会有帮助。

- –dont-overwrite-file

  不重写当前文件。

  文本、tar以及自定义格式情况下会重写现有文件。这对目录格式不适用。

  例如：设想这样一种情景，即当前目录下backup.sql已存在。如果再次备份导出backup.sql文件时，文件就会被重写。

  如果备份文件已存在，且输入–dont-overwrite-file选项，则会报告错误信息“转储文件已经存在”。

- –use-set-session-authorization

  输出符合SQL标准的SET SESSION AUTHORIZATION命令而不是ALTER OWNER命令来确定对象所有权。这样令转储更加符合标准，但是如果转储文件中的对象的历史有些问题，那么可能不能正确恢复。并且，使用SET SESSION AUTHORIZATION的转储需要数据库系统管理员的权限才能转储成功，而ALTER OWNER需要的权限则低得多。

- –with-encryption=AES128/SWS-KMS-SM4/TAS-KMS-SM4

  指定转储数据使用的加密方式（算法）。支持AES128/SWS-KMS-SM4/TAS-KMS-SM4。

  该选项必须与–with-key和–with-salt同时使用。

- –with-key=KEY

  密钥名称或者密钥索引。

  AES128密钥规则如下：

  - 密钥长度为8~16个字符。
  - 至少包含大写字母（A-Z），小写字母（a-z），数字（0-9），非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。
  - 加密算法为TAS-KMS-SM4时，–with-key指定的是江南天安KMS上创建的密钥索引，VexDB限制其索引范围必须是1 ~ 2048。
  - 加密算法为SWS-KMS-SM4时，–with-key指定的是三未信安KMS上创建的密钥名称。

- –with-salt=RANDVALUES

  加密导出时使用此参数传递随机盐值。必须为 16 个字符的字符串。

- –include-extensions

  如果include-extensions参数被设置，将备份所有的CREATE EXTENSION语句。

- –include-templatedb

  转储过程中包含模板库。

- –pipeline

  使用管道传输密码，禁止在终端使用。

- –binary-upgrade

  该参数为扩展预留接口，不建议使用。

- –binary-upgrade-usermap和–binary-upgrade必须一起使用。

- –binary-upgrade-usermap= “USER1=USER2”

  该参数为扩展预留接口，不建议使用。

- –non-lock-table

  该参数为扩展预留接口，不建议使用。

- –tablespaces-postfix

  该参数为扩展预留接口，不建议使用。

- –parallel-jobs

  指定备份进程并发数，取值范围为1~1000。

  –parallel-jobs和-f/–file必须同时使用。

- –skip-initial-user

  导出角色时跳过初始用户。

- –get-src-def

  导出视图或者物化视图的原始定义（包含注释）。

  未指定该选项时，默认通过反序列化的方式获取视图或物化视图的定义。此方式可以确保在视图或其依赖对象的定义发生修改的情况下，仍能正确获取到当前的有效视图定义。

连接参数

- -h, –host=HOSTNAME

指定主机的名称。如果取值是以斜线开头，它将用作Unix域套接字的目录。默认值取自PGHOST环境变量；如果没有设置，将启动某个Unix域套接字建立连接。该参数只针对VexDB外，对VexDB内本机只能用127.0.0.1。

环境变量：$PGHOST

- -l, –database=DATABASE

  指定所连接的转储全局对象的数据库名称，并去寻找还有其他哪些数据库需要被转储。如果没有指定，会使用postgres数据库，如果postgres数据库不存在，会使用template1。

- -p, –port=PORT

  指定服务器所侦听的TCP端口或本地Unix域套接字后缀，以确保连接。默认值设置为PGPORT环境变量。在开启线程池情况下，建议使用 pooler port，即侦听端口+1。

  环境变量：$PGPORT

- -U, –username=NAME

  所连接的用户名。

  环境变量：$PGUSER

- -w, –no-password

  不出现输入密码提示。如果服务器要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。

- -W, –password=PASSWORD

  指定用户连接的密码。

  如果主机的认证策略是trust，则不会对系统管理员进行密码验证，即无需输入-W选项；

  如果没有-W选项，并且不是系统管理员，会提示用户输入密码。

- –role=ROLENAME

  指定创建转储使用的角色名。选择该选项，会使vb_dumpall连接数据库后，发起一个SET ROLE角色名命令。当所授权用户（由-U指定）没有vb_dumpall要求的权限时，该选项会起到作用，即切换到具备相应权限的角色。某些安装操作规定不允许直接以系统管理员身份登录，而使用该选项能够在不违反该规定的情况下完成转储。

- –rolepassword=ROLEPASSWORD

  指定具体角色用户的角色密码。

示例

示例1

使用vb_dumpall一次导出VexDB的所有数据库。

备份命令如下：

```
vb_dumpall -f $PGDATA/backup_all.sql -p 5432
```

备份成功，打印信息如下：

```
vb_dump[port='5432'][dbname='postgres'][2025-08-22 12:17:10]: The total objects number is 447.
vb_dump[port='5432'][dbname='postgres'][2025-08-22 12:17:10]: [100.00%] 447 objects have been dumped.
vb_dump[port='5432'][dbname='postgres'][2025-08-22 12:17:10]: dump database dbname='postgres' successfully
vb_dump[port='5432'][dbname='postgres'][2025-08-22 12:17:10]: total time: 1386  ms
vb_dump[port='5432'][dbname='vexdb'][2025-08-22 12:17:11]: The total objects number is 447.
vb_dump[port='5432'][dbname='vexdb'][2025-08-22 12:17:11]: [100.00%] 447 objects have been dumped.
vb_dump[port='5432'][dbname='vexdb'][2025-08-22 12:17:11]: dump database dbname='vexdb' successfully
vb_dump[port='5432'][dbname='vexdb'][2025-08-22 12:17:11]: total time: 1317  ms
vb_dumpall[port='5432'][2025-08-22 12:17:11]: dumpall operation successful
vb_dumpall[port='5432'][2025-08-22 12:17:11]: total time: 2769  ms
```

示例2

使用vb_dumpall备份时指定-c，将清理这些数据库（对象）的语句输出到备份文件的开头。表示在恢复该文件时，先执行SQL语句清理这些数据库（对象），再重新创建数据库。

1. 使用vsql工具连接至vexdb数据库，创建测试表并插入一条数据。

   ```
   create table student(stdno int,student_age char(20),grade float8);
   insert into student values(001,'15.00',81.00);
   ```

2. 退出vsql程序、执行备份命令。

   ```
   vb_dumpall -f $PGDATA/bak_all.sql -p 5432 -c
   ```

   备份成功，打印信息如下：

```
vb_dump[port='5432'][dbname='postgres'][2025-08-20 14:04:18]: The total objects number is 447.
vb_dump[port='5432'][dbname='postgres'][2025-08-20 14:04:18]: [100.00%] 447 objects have been dumped.
vb_dump[port='5432'][dbname='postgres'][2025-08-20 14:04:18]: dump database dbname='postgres' successfully
vb_dump[port='5432'][dbname='postgres'][2025-08-20 14:04:18]: total time: 1339  ms
vb_dump[port='5432'][dbname='test5'][2025-08-20 14:04:19]: The total objects number is 453.
vb_dump[port='5432'][dbname='test5'][2025-08-20 14:04:19]: [100.00%] 453 objects have been dumped.
vb_dump[port='5432'][dbname='test5'][2025-08-20 14:04:19]: dump database dbname='test5' successfully
vb_dump[port='5432'][dbname='test5'][2025-08-20 14:04:19]: total time: 1356  ms
vb_dump[port='5432'][dbname='vexdb'][2025-08-20 14:04:20]: The total objects number is 451.
vb_dump[port='5432'][dbname='vexdb'][2025-08-20 14:04:20]: [100.00%] 451 objects have been dumped.
vb_dump[port='5432'][dbname='vexdb'][2025-08-20 14:04:20]: dump database dbname='vexdb' successfully
vb_dump[port='5432'][dbname='vexdb'][2025-08-20 14:04:20]: total time: 1366  ms
vb_dumpall[port='5432'][2025-08-20 14:04:20]: dumpall operation successful
vb_dumpall[port='5432'][2025-08-20 14:04:20]: total time: 4144  ms
```

逻辑恢复

本章节介绍如何通过由vb_dump、vb_dumpall逻辑备份得到的文件进行恢复。

当用户遇到数据丢失、误删等情况时，可以通过以下方式恢复备份过的数据：

- vb_restore

  vb_restore工具用于恢复由vb_dump备份得到的自定义归档格式（c）、目录格式（d）或者tar归档格式（t）的备份文件。

- 使用vsql工具恢复

  用于恢复由vb_dump、vb_dumpall备份得到的纯文本格式的文件。

vb_restore

背景信息

vb_restore是VexDB提供的针对vb_dump导出数据的导入工具。通过此工具可导入由vb_dump生成的二进制归档格式（c）、目录格式（d）、tar格式（t）的备份文件。

vb_restore的主要功能包含：

- 导入到数据库

  如果连接参数中指定了数据库，则数据将被导入到指定的数据库中。其中，并行导入必须指定连接的密码。导入时生成列会自动更新，并像普通列一样保存。

- 导入到脚本文件

如果未指定导入数据库，则必须指定list选项。将会创建包含重建数据库所必须的SQL语句脚本并写入到文件或者标准输出。等效于直接使用vb_dump导出为纯文本格式。多用于TOC文件选择性恢复。

工具运行产生的日志位于 `$GAUSSLOG/bin/vb_restore` 目录下，每个日志最大 16 MB，最多保留 50 个。如果没有配置环境变量 $GAUSSLOG，则不会产生日志文件。

注意事项

- vb_restore工具由安装VexDB数据库的操作系统用户执行。
- 一旦恢复，最好在每个数据库上运行ANALYZE | ANALYSE，优化程序提供有用的统计数据。
- 如果安装过程中有任何本地数据要添加到template1数据库，请谨慎将vb_restore的输出载入到一个真正的空数据库中；否则可能会因为被添加对象的定义被复制，而出现错误。要创建一个无本地添加的空数据库，需从template0而非template1复制，例如：

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

- vb_restore不能选择性地导入大对象；例如只能导入那些指定表的对象。如果某个归档形式包含大对象，那所有大对象都会被导入或一个都不会被导入。如果此归档对象通过-L、-t或其他选项被排除，那么所有大对象一个都不会被导入。
- vb_restore默认是以追加的方式进行数据导入。为避免多次导入造成数据异常，在进行导入时，建议使用“-c”参数，在重新创建数据库对象前，清理（删除）已存在于将要还原的数据库中的数据库对象。
- 日志打印无开关，若需隐藏日志，请将日志重定向到日志文件。若恢复表数据时，数据量很大，会分批恢复，因此会多次出现“表数据已完成导入”的日志。
- 排除表恢复不支持指定某一子分区，对于分区表可用参数–exclude-table-file或–exclude-table指定分区表。
- –exclude-table和–exclude-table-file不支持和以下参数同时使用：
  - -t, –table=NAME
  - -I, –index=NAME
  - -s, –schema-only

命令格式

```
vb_restore [OPTION]... FILE
```

- FILE没有短选项或长选项。用来指定归档文件所处的位置。
- 作为前提条件，需输入dbname或-l选项。不允许用户同时输入dbname和-l选项。

通用参数：

```
General options:
  -d, --dbname=NAME                    connect to database name
  -f, --file=FILENAME                  output file name
  -F, --format=c|d|t                   backup file format (should be automatic)
  -l, --list                           print summarized TOC of the archive
  -v, --verbose                        verbose mode
  -V, --version                        output version information, then exit
  -?, --help                           show this help, then exit
```

恢复控制参数：

```
Options controlling the restore:
  -a, --data-only                      restore only the data, no schema
  -c, --clean                          clean (drop) database objects before recreating
  -C, --create                         create the target database
  -e, --exit-on-error                  exit on error, default is to continue
  -I, --index=NAME                     restore named index(s)
  -j, --jobs=NUM                       use this many parallel jobs to restore
  -L, --use-list=FILENAME              use table of contents from this file for
                                       selecting/ordering output
  -n, --schema=NAME                    restore only objects in this schema(s)
  -O, --no-owner                       skip restoration of object ownership
  -P, --function=NAME(args)            restore named function(s)
  -r, --remap-schema=SCHEMA1:SCHEMA2   change SCHEMA1 to SCHEMA2 while restore
  -s, --schema-only                    restore only the schema, no data
  -S, --sysadmin=NAME                  system admin user name to use for disabling triggers
  -t, --table=NAME                     restore named table(s)
  -T, --trigger=NAME                   restore named trigger(s)
  -x, --no-privileges/--no-acl         skip restoration of access privileges (grant/revoke)
  -1, --single-transaction             restore as a single transaction
  --disable-triggers                   disable triggers during data-only restore
  --no-data-for-failed-tables          do not restore data of tables
                                       that could not be created
  --sess-replic-role=replica           Set the guc parameter session
                                       _replication_role
  --no-publications                    do not restore publications
  --no-security-labels                 do not restore security labels
  --no-subscriptions                   do not restore subscriptions
  --no-tablespaces                     do not restore tablespace assignments
  --section=SECTION                    restore named section (pre-data, data, or post-data)
  --use-set-session-authorization      use SET SESSION AUTHORIZATION commands instead of
                                       ALTER OWNER commands to set ownership
  --pipeline                           use pipeline to pass the password,
                                       forbidden to use in terminal
  --with-decryption=AES128/SM4         restore data is decrypted using AES128 or SM4 in hardware encryption mode
  --with-key=KEY                       hardware encryption device internal key index
  --with-salt=RANDVALUES               random values for decrypt
  --exclude-table                      exclude single table, input format: SCHEMA.Object
  --exclude-table-file=FileName        exclude one or more tables, input file format: SCHEMA.Object in each line.
```

连接参数：

```
Connection options:
  -h, --host=HOSTNAME                  database server host or socket directory
  -p, --port=PORT                      database server port number
  -U, --username=NAME                  connect as specified database user
  -w, --no-password                    never prompt for password
  -W, --password=PASSWORD              the password of specified database user
  --role=ROLENAME                      do SET ROLE before restore
  --rolepassword=ROLEPASSWORD          the password for role
```

参数说明

通用参数

- -d, –dbname=NAME

  连接数据库dbname并直接导入到该数据库中。

- -f, –file=FILENAME

  指定生成脚本的输出文件，或使用-l时列表的输出文件。

  默认是标准输出。

  -d/–dbname和-f/–file 不能同时使用。

- -F, –format=c|d|t

  指定归档格式。由于vb_restore会自动决定格式，因此不需要指定格式。

  取值范围：

  - c/custom：该归档形式为vb_dump的二进制格式。
  - d/directory：该归档形式是一个目录归档形式。
  - t/tar：该归档形式是一个tar归档形式。

- -l, –list

  列出归档形式内容。这一操作的输出可用作-L选项的输入。注意如果像-n或-t的过滤选项与-l使用，过滤选项将会限制列举的项目（即归档形式内容）。

- -v, –verbose

  开启冗长模式，打印详细信息。

- -V, –version

  打印vb_restore版本，然后退出。

- -?, –help

  显示vb_restore命令行参数帮助，然后退出。

恢复控制参数

- -a, –data-only

  只导入数据，不导入模式（数据定义）。vb_restore的导入是以追加方式进行的。

  -s/–schema-only 和 -a/–data-only不能同时使用。

- -c, –clean

  在重新创建数据库对象前，清理（删除）已存在于将要还原的数据库中的数据库对象。

  -c/–clean 和 -a/–data-only不能同时使用。

- -C, –create

  导入数据库之前会先使用CREATE DATABASE创建导出时连接的数据库。（指定该选项后，-d指定的数据库仅用于数据库连接，所有数据会导入到创建的数据库中。）

  对于VexDB数据库而言，当前此参数暂无任何实际作用，-d参数指定的数据库就是数据恢复或导入的实际目标。

- -e, –exit-on-error

  当发送SQL语句到数据库时如果出现错误，请退出。默认状态下会继续，且在导入后会显示一系列错误信息。

- -I, –index=NAME

  只导入已列举的index的定义。允许导入多个index。如果多次输入-I index导入多个index。

  例如：

  ```
  vb_restore -h host_name -p port_number -d postgres -I Index1 -I Index2 backup/MPPDB_backup.tar
  ```

  在上面这个例子中，Index1和Index2会被导入。

- -j, –jobs=NUM

  运行vb_restore最耗时的部分（如加载数据、创建index或创建约束）使用并发任务。该选项能大幅缩短导入时间，即将一个大型数据库导入到某一多处理器的服务器上。

使用–single-transaction时，-j/–jobs必须为单任务。

> **说明**
>
> - 每个任务可能是一个进程或一个线程，这由操作系统决定。每个任务与服务器进行单独连接。
> - 该选项的最优值取决于服务器的硬件设置、客户端以及网络。还包括这些因素，如CPU核数量、硬盘设置。建议是从增加服务器上的CPU核数量入手，更大的值（服务器上CPU核数量）在很多情况下也能导致数据文件更快的被导入。当然，过高的值会由于超负荷反而导致性能降低。
> - 该选项只支持二进制归档格式（c）和目录格式（d）。输入文件必须是常规文件（不能是像pipe的文件）。如果是通过脚本文件，而非直接连接数据库服务器，该选项可忽略。而且，多任务不能与–single-transaction选项一起使用。

- -L, –use-list=list-file

  只导入列举在list-file中的那些归档形式元素，导入顺序以它们在文件中的顺序为准。注意如果像-n或-t的过滤选项与-L使用，它们将会进一步限制导入的项目。一般情况下，list-file是通过编辑前面提到的某个-l参数的输出创建的。文件行的位置可更改或直接删除，也可使用分号（;）在行的开始注出。

  当语句中指定了-L，而没有指定dbname时，将list-file中过滤后的恢复命令输出到屏幕，而不实际进行恢复操作。用法参考示例5：-L输出恢复语句但不执行恢复。

  将恢复命令输出到屏幕时，会一并输出vb_restore恢复过程中对数据库参数的修改（比如将client_min_messages修改为最小级别来忽略恢复过程中的问题），这些修改在数据库中执行后可能造成非预期的结果。如果恢复过程不想执行这些参数的改动，可以编辑基于备份文件生成的SQL脚本，删除或注释掉脚本中需要跳过的操作，再使用vsql工具恢复。

- -n, –schema=NAME

  只导入已列举的模式中的对象。该选项可与-t选项一起用以导入某个指定的表。多次输入-n _schemaname_可以导入多个模式。

  例如：

  ```
  vb_restore -h host_name -p port_number -d postgres -n sch1 -n sch2 backup/MPPDB_backup.tar
  ```

  在上面这个例子中，sch1和sch2会被导入。

- -O, –no-owner

  不输出设置对象的归属这样的命令，以匹配原始数据库。默认情况下，vb_restore会发出ALTER OWNER或SET SESSION AUTHORIZATION语句设置所创建的模式元素的所属。除非是由系统管理员（或是拥有脚本中所有对象的同一个用户）进行数据库首次连接的操作，否则语句会失败。使用-O选项，任何用户名都可用于首次连接，且该用户拥有所有已创建的对象。

- -P, –function=NAME(args)

  只导入已列举的函数。请按照函数所在转储文件中的目录，准确拼写函数名称和参数。当-P单独使用时，表示导入文件中所有’function-name(args)’函数；当-P同-n一起使用时，表示导入指定模式下的’function-name(args)’函数；多次输入-P，而仅指定一次-n，表示所有导入的函数默认都是位于-n模式下的。可以多次输入-n schema-name -P ’function-name(args)’同时导入多个指定模式下的函数。

例如：

```
vb_restore -h host_name -p port_number -d postgres -n test1 -P 'Func1(integer)' -n test2 -P 'Func2(integer)' backup/MPPDB_backup.tar
```

在上面这个例子中，test1模式下的函数Func1(i integer)和test2模式下的函数Func2(j integer)会被一起导入。

- -r, –remap-schema=SCHEMA1:SCHEMA2

  恢复时将SCHEMA1更改为SCHEMA2进行恢复。不支持与以下参数同时使用：

  - -a, –data-only restore only the data, no schema
  - -c, –clean clean (drop) database objects before recreating
  - -C, –create create the target database
  - -O, –no-owner skip restoration of object ownership

- -s, –schema-only

  只导入模式（数据定义），不导入数据（表内容）。当前的序列值也不会导入。

  -s/–schema-only 和 -a/–data-only不能同时使用。

  若在使用vb_dump导出时指定了只导出模式（即指定了-s参数），则导入时必须指定本参数。

- -S, –sysadmin=NAME

  该参数为扩展预留接口，不建议使用。

- -t, –table=NAME

  只导入已列举的表定义、数据或定义和数据。该选项与-n选项同时使用时，用来指定某个模式下的表对象。-n参数不输入时，默认为PUBLIC模式。多次输入-n -t 可以导入指定模式下的多个表。

  例如：

  导入PUBLIC模式下的table1：

  ```
  vb_restore -h host_name -p port_number -d postgres -t table1 backup/MPPDB_backup.tar
  ```

  导入test1模式下的test1和test2模式下test2：

  ```
  vb_restore -h host_name -p port_number -d postgres -n test1 -t test1 -n test2 -t test2 backup/MPPDB_backup.tar
  ```

导入PUBLIC模式下的table1和test1 模式下test1：

```
vb_restore -h host_name -p port_number -d postgres -n PUBLIC -t table1 -n test1 -t table1 backup/MPPDB_backup.tar
```

-t不支持schema_name.table_name，指定此格式不会报错，但不会生效。

- -T, –trigger=NAME

  该参数为扩展预留接口。

- -x, –no-privileges/–no-acl

  防止导入访问权限（GRANT/REVOKE命令）。

- -1, –single-transaction

  执行导入作为一个单独事务（即把命令包围在BEGIN/COMMIT中）。该选项确保要么所有命令成功完成，要么没有改变应用。该选项意为–exit-on-error。

  使用–single-transaction时，-j/–jobs必须为单任务。

- –disable-triggers

  该参数为扩展预留接口，不建议使用。

- –no-data-for-failed-tables

  默认状态下，即使创建表的命令失败（如表已经存在），表数据仍会被导入。使用该选项，像这种表的数据会被跳过。如果目标数据库已包含想要的表内容，这种行为会有帮助。该选项只有在直接导入到某数据库中时有效，不针对生成SQL脚本文件输出。

- –sess-replic-role=replica

  导入时提前设置guc参数session_replication_role为replica，以避免触发器行为、重写规则对恢复过程的影响。

  由于session_replication_role是SUSET类型参数，只允许数据库管理员用户进行修改，使用普通用户导入时该设置不生效，会报错提示权限不足，无法修改参数。

- –no-publications

  不导入发布。

- –no-security-labels

  该参数为扩展预留接口，不建议使用。

- –no-subscriptions

  不导入订阅。

- –no-tablespaces

  不输出选择表空间的命令。使用该选项，无论默认表空间是哪个，在导入过程中所有对象都会被创建。

- –section=SECTION

  导入已列举的区段（如pre-data、data或post-data）。

- –use-set-session-authorization

  该选项用来进行文本格式的备份。

  输出SET SESSION AUTHORIZATION命令，而非ALTER OWNER命令，用以决定对象归属。该选项使转储更加兼容标准，但通过参考转储中对象的记录，导入过程可能会有问题。使用SET SESSION AUTHORIZATION的转储要求必须是系统管理员，同时在导入前还需参考SET SESSION AUTHORIZATION，手工对导出文件的密码进行修改验证，只有这样才能进行正确的导入操作，相比之下，ALTER OWNER对权限要求较低。

- –pipeline

  使用管道传输密码，禁止在终端使用。

- –with-decryption=AES128/SM4

  还原数据时采用AES128或SM4硬件加密方式解密。

  若导出时使用–with-encryption指定了加密方式，则恢复时需要指定此参数并采用同样的方式进行解密。

  该选项必须与–with-key和–with-salt同时使用。

- –with-key=KEY

  硬件加密设备内部的密钥索引。

- –with-salt=RANDVALUES

  用于解密的随机盐值。

  - 必须为16个字符的字符串。
  - 恢复时指定的盐值必须与加密导出时指定的盐值相同。

- –exclude-table

  排除指定表进行恢复，输入格式为SCHEMA.OBJECT。不支持与以下参数同时使用：

  - -t, –table=NAME restore named table(s)
  - -I, –index=NAME restore named index(s)

- -s, –schema-only restore only the schema, no data

- –exclude-table-file=FileName

  排除指定文件中包含的单表或多表进行恢复，文件格式为每行指定SCHEMA.OBJECT。不支持与以下参数同时使用：

  - -t, –table=NAME restore named table(s)
  - -I, –index=NAME restore named index(s)
  - -s, –schema-only restore only the schema, no data

  –exclude-table-file仅限于排除表，与其他数据库对象无关。

连接参数

- -h, –host=HOSTNAME

  指定的主机名称。如果取值是以斜线开头，将用作Unix域套接字的目录。默认值取自PGHOST环境变量；如果没有设置，将启动某个Unix域套接字建立连接。该参数只针对VexDB外，对VexDB内本机只能用127.0.0.1。

  环境变量：PGHOST

- -p, –port=PORT

  指定服务器所侦听的TCP端口或本地Unix域套接字后缀，以确保连接。默认值设置为PGPORT环境变量。在开启线程池情况下，建议使用 pooler port，即侦听端口+1。

  环境变量：PGPORT

- -U, –username=NAME

  所连接的用户名。

  环境变量：PGUESR

- -w, –no-password

  不出现输入密码提示。如果服务器要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。

- -W, –password=PASSWORD

  指定用户连接的密码。如果主机的认证策略是trust，则不会对系统管理员进行密码验证，即无需输入-W参数；如果没有-W参数，并且不是系统管理员，vb_restore会提示用户输入密码。

- –role=ROLENAME

  指定导入操作使用的角色名。选择该参数，会使vb_restore连接数据库后，发起一个SET ROLE角色名命令。当所授权用户（由-U指定）没有vb_restore要求的权限时，该参数会起到作用，即切换到具备相应权限的角色。某些安装操作规定不允许直接以初始用户身份登录，而使用该参数能够在不违反该规定的情况下完成导入。

- –rolepassword=ROLEPASSWORD

  指定具体角色用户的角色密码。

  –role 和–rolepassword必须一起使用。

TOC文件选择性恢复

功能描述

使用vb_restore还原除了纯文本格式以外的备份时，可以通过生成并编辑TOC控制文件，将文件内的行注释、删除或重新排列，达到定制还原效果的目的。

注意事项

- 用户可以在生成控制文件后，以行前加分号“;”的方式注释掉不用还原的内容。
- 控制文件内，每行开头的数字代表赋给每个项目的内部归档 ID。

语法格式

生成控制文件的语句如下：

```
vb_restore -l -f test.toc backup.dmp
```

或者：

```
vb_restore -l backup.dmp > test.sql
```

上述语句中：

- test.toc、test.sql是生成的控制文件的名称（包括路径）。
- backup.dmp是通过vb_dump导出得到的备份文件名称（包括路径），备份文件也可以是tar格式或者目录格式。

TOC控制文件的使用请参考示例4。

示例

> **说明**
>
> 示例中待恢复的备份文件均来自vb_dump的示例。

示例1

创建新数据库并恢复

恢复vb_dump示例4中的备份文件backup4_1.dmp，恢复时指定其恢复到一个新的数据库。

1. 使用vsql工具连接至数据库vexdb。
2. 创建数据库test5的数据库。

```
CREATE  database test5;
```

3. 退出vsql程序，执行导入操作。

```
vb_restore -d test5 $PGDATA/backup4_1.dmp
```

4. 完成导入操作后，使用vsql工具连接至test5数据库。

```
vsql -d test5 -p 5432 -r
```

5. 使用如下语句查看导入结果。

```
select * from table1;
select * from table2;
select * from table3;
```

返回结果如下，表示已成功导入test5库，数据库中的每张表都有且仅有一条数据。

```
name  | id
-------+----
alice | 1
(1 row)

name | id
------+----
jack | 2
(1 row)

name | id
------+----
bob  | 3
(1 row)
```

示例2

-c清理对象再恢复

在本文示例1的基础上，再次执行恢复语句导入backup4_1.dmp。在执行时指定-c，表示恢复数据前先删除对象，恢复操作不会导致数据重复。

1. 执行导入语句。

```
vb_restore -d test5 $PGDATA/backup4_1.dmp -c
```

2. 完成导入操作后，使用vsql工具连接至test5数据库。

```
vsql -d test5 -p 5432 -r
```

3. 使用如下语句查看导入结果。

```
select * from table1;
select * from table2;
select * from table3;
```

返回结果如下，表示已成功导入test5库，数据库中的每张表都有且仅有一条数据。

```
name  | id
-------+----
alice | 1
(1 row)

name | id
------+----
jack | 2
(1 row)

name | id
------+----
bob  | 3
(1 row)
```

示例3

-s仅恢复定义

在一个新数据库内恢复vb_dump示例4的备份文件backup4_1.dmp。在恢复时指定-s，代表仅恢复定义。

1. 使用vsql工具连接至数据库vexdb。

```
vsql -d vexdb -p 5432 -r
```

2. 创建用于恢复的数据库testdb4。

```
create database testdb4;
```

3. 退出vsql程序，执行恢复语句。

```
vb_restore -d testdb4 $PGDATA/backup4_1.dmp -s
```

4. 恢复成功后，使用vsql工具连接至数据库testdb4。

```
vsql -d testdb4 -p 5432 -r
```

5. 使用如下语句查看恢复情况。

```
select * from table1;
select * from table2;
select * from table3;
```

返回结果如下，表示已成功将table1、table2、table3导入到数据库testdb4中，只包括定义，不包括数据。

```
name | id
------+----
(0 rows)

name | id
------+----
(0 rows)

name | id
------+----
(0 rows)
```

示例4

TOC文件选择性恢复

在一个新数据库内恢复vb_dump示例4的备份文件backup4_1.dmp，并通过TOC控制文件排除表table2。

1. 使用vsql工具连接至数据库vexdb，创建用于恢复的数据库testdb5。

```
create database testdb5;
```

2. 退出vsql程序，为备份文件backup4_1生成相应的TOC控制文件。

```
vb_restore -l -f test.toc $PGDATA/backup4_1.dmp
```

3. 查看TOC文件的内容。

```
vi test.toc
```

在包含table2信息的第17和22行首加上分号；注释掉内容，修改后的示例如下。修改后保存退出。

```
;
; Archive created at Thu Apr 20 16:45:52 2023
;     dbname: test4
;     TOC Entries: 10
;     Compression: -1
;     Dump Version: 1.12-0
;     Format: CUSTOM
;     Integer: 4 bytes
;     Offset: 8 bytes
;     Dumped from database version: 9.2.4
;     Dumped bygs_dump version: 9.2.4
;
;
; Selected TOC Entries:
;
720; 1259 16619 TABLE public table1 vexdb
;721; 1259 16625 TABLE public table2 vexdb
722; 1259 16631 TABLE public table3 vexdb
447; 1259 13336 VIEW public vb_login_info vexdb
5527; 0 0 ACL public vb_login_info vexdb
5519; 0 16619 TABLE DATA public table1 vexdb
;5520; 0 16625 TABLE DATA public table2 vexdb
5521; 0 16631 TABLE DATA public table3 vexdb
```

4. 使用TOC控制文件完成恢复。

```
vb_restore -Fc -C -L test.toc -d testdb5 $PGDATA/backup4_1.dmp
```

5. 完成恢复后，使用vsql工具连接至数据库testdb5。

```
vsql -d testdb5 -p 5432 -r
```

6. 查看当前数据库的表信息。

```
\d
```

返回结果如下，表示成功导入了table1和table3。

```
               List of relations
Schema |     Name      | Type  |  Owner   |                    Storage
--------+---------------+-------+----------+-------------------------------------------------
public | table1        | table | vexdb | {orientation=row,compression=no,fillfactor=80}
public | table3        | table | vexdb | {orientation=row,compression=no,fillfactor=80}
public | vb_login_info | view  | vexdb |
(3 rows)
```

7. 查看table1和table3的数据。

```
select * from table1;
select * from table3;
```

返回结果如下：

```
name | id
------+----
alice | 1
(1 row)

name | id
------+----
bob  | 3
(1 row)
```

示例5

-L输出恢复语句但不执行恢复

1. 在vexdb数据库中创建测试表并插入测试数据，查看测试表数据。

```
CREATE TABLE table_test1 (
col1 NUMBER,
col2 VARCHAR2(50),
col3 VARCHAR2(50),
col4 VARCHAR2(100),
col5 DATE);
INSERT INTO table_test1(col1, col2, col3, col4, col5) VALUES(1, 'value1', 'value2', 'some text', TO_DATE('2022-01-01', 'YYYY-MM-DD'));
SELECT * FROM table_test1;
```

返回结果如下：

```
col1 |  col2  |  col3  |   col4    |    col5
------+--------+--------+-----------+------------
    1 | value1 | value2 | some text | 2022-01-01
(1 row)
```

2. 退出数据库连接，使用安装VexDB的操作系统用户执行vb_dump命令，备份vexdb数据库的内容。

```
vb_dump vexdb -t table_test1 -F c -f /tmp/bak_test.dmp
```

3. 备份成功后，使用vb_restore的-l选项生成一个归档文件list-file。

```
vb_restore -l -f /tmp/testsql.list /tmp/bak_test.dmp
```

4. 使用vb_restore的-L选项，依据list-file输出过滤后的恢复语句。

```
vb_restore -L /tmp/testsql.list /tmp/bak_test.dmp
```

返回结果如下：

```
--
--vexdb dump
--

SET statement_timeout = 0;
SET xmloption = content;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET uppercase_attribute_name = false;
SET check_function_bodies = false;
SET session_replication_role = replica;
SET client_min_messages = warning;
SET enable_dump_trigger_definer = on;
SET behavior_compat_options = '';

SET search_path = public;

SET default_tablespace = '';

SET default_with_oids = false;

--
-- Name: table_test1; Type: TABLE; Schema: public; Owner:vexdb; Tablespace:
--

CREATE TABLE table_test1 (
    col1 numeric,
    col2 varchar(50),
    col3 varchar(50),
    col4 varchar(100),
    col5 date
)
WITH (orientation=row, compression=no, fillfactor=80);


ALTER TABLE public.table_test1 OWNER TO vexdb;

--
-- Data for Name: table_test1; Type: TABLE DATA; Schema: public; Owner:vexdb--

COPY table_test1 (col1, col2, col3, col4, col5) FROM stdin;
1       value1  value2  some text       2022-01-01
\.
;

--
-- reset some params
--

RESET session_replication_role;
--
--vexdb dump complete
--
```

5. 后续可以按需选择使用vb_restore恢复，或者通过编辑基于备份文件生成的SQL脚本来调整恢复流程和效果，再使用vsql工具恢复。后续恢复步骤此处不再演示。

   基于备份文件生成SQL脚本的命令如下：

```
vb_restore -L /tmp/testsql.list /tmp/bak_test.dmp > /tmp/test.sql
```

使用vsql工具恢复

功能描述

当待恢复的备份文件是由 vb_dump 或vb_dumpall 导出的纯文本格式（FILE.sql）的文件时，可以使用以下两种方式进行备份文件的恢复：

- 方式一：通过 vsql工具的-f选项执行文件中的命令进行恢复。
- 方式二：使用 vsql连接到数据库后，使用。

注意事项

- 本节介绍的方式仅用于恢复纯文本格式的备份文件。
- 使用 vsql工具恢复纯文本格式的备份文件时是以追加方式进行导入的。多次导入会造成数据异常。
- 恢复成功后，建议在每个数据库上运行ANALYZE | ANALYSE，优化程序提供有用的统计数据。
- vb_dumpall恢复前，必须确保所有必要的表空间目录为空；否则，对于处在非默认位置的数据库，数据库创建会失败。（可以通过vsql工具的。）
- 方式二中，`\i` 元命令用于从文件中读取内容，并将其当作输入，执行查询。更多vsql下的元命令详见vsql。

命令格式

方式一：

```
vsql -d vexdb -p 5432 -f FILE.sql
```

- -d：连接的数据库名称，默认为vexdb。
- -p：连接的数据库端口号，默认为5432。
- -f：执行文件FILE.sql中的命令，然后退出。

方式二：

```
\i FILE
```

- 使用vsql工具连接到数据库后才能执行-i元命令。
- FILE：待恢复的备份文件名称（包括路径）。

示例

> **说明**
>
> - 示例1~示例3中待恢复的备份文件来自vb_dump的示例4。
> - 示例4中待恢复的备份文件来自vb_dumpall的示例2。

示例1

恢复备份文件backup4_2.sql。备份时指定了-C，因此恢复时会自行创建名为test4的数据库。

1. 使用vsql工具连接至数据库vexdb。

   ```
   vsql -d vexdb -p 5432 -r
   ```

2. 删除名为test4的数据库（如果存在）。

   请注意当前操作对业务的影响，谨慎操作。

   若预先已存在名为test4的数据库，再次导入时会导致数据重复。

   ```
   drop database if exists test4;
   ```

3. 执行导入操作。

   ```
   \i backup4_2.sql
   ```

4. 导入时会切换连接至test4数据库下，在test4数据库中执行以下语句查询导入结果：

```
select * from table1;
select * from table2;
select * from table3;
```

返回结果如下，表示已成功导入test4库，数据库中的每张表都有且仅有一条数据。

```
name  | id
-------+----
alice | 1
(1 row)

name | id
------+----
jack | 2
(1 row)

name | id
------+----
bob  | 3
(1 row)
```

示例2

恢复备份文件backup4_3.sql。备份时指定了-c，表示在导入test4库之前先删除数据库对象。

1. 执行以下语句恢复备份文件backup4_3.sql。

```
vsql -d test4 -p 5432 -f backup4_3.sql
```

2. 连接至数据库test4下。

```
vsql -d test4 -p 5432 -r
```

3. 使用以下语句查看导入结果：

```
select * from table1;
select * from table2;
select * from table3;
```

返回结果如下，表示已成功导入test4数据库，每张表都有且仅有一条数据，并未重复。

```
name  | id
-------+----
alice | 1
(1 row)

name | id
------+----
jack | 2
(1 row)

name | id
------+----
bob  | 3
(1 row)
```

示例3

新建数据库并恢复备份文件backup4_5.sql。备份时仅备份了所有表的定义（模式）。

1. 使用vsql工具连接至数据库vexdb下。

```
vsql -d vexdb -p 5432 -r
```

2. 创建用于导入的数据库db_test。

```
create database db_test;
```

3. 切换至数据库db_test下。

```
\c db_test
```

4. 将备份文件导入到当前数据库中。

```
\i backup4_5.sql
```

5. 导入完成后，查看导入结果。

```
select * from table1;
select * from table2;
select * from table3;
```

返回结果如下，表示已成功将table1、table2、table3导入到数据库db_test中，只包括定义（模式），不包括数据。

```
name  | id
------+----
(0 rows)

name | id
------+----
(0 rows)

name | id
------+----
(0 rows)
```

示例4

恢复vb_dumpall导出的示例2，备份文件为bak_all.sql。备份时指定了-c，在重新创建数据库之前，会执行SQL语句清理（删除）这些数据库。因此导入后的表数据不会重复。

1. 使用vsql工具连接至数据库vexdb。

```
vsql -d vexdb -p 5432 -r
```

2. 使用如下语句进行恢复：

```
\i bak_all.sql
```

3. 恢复完成后，验证恢复效果。

```
select * from student;
```

返回结果如下，表示恢复成功。

测试表student的数据完成了导入，并且数据不重复。

```
stdno |     student_age      | grade
-------+----------------------+-------
    1 | 15.00                |    81
(1 row)
```

物理备份与恢复

物理备份是通过对数据库的物理文件（如数据、日志文件等）进行拷贝的方式对数据库进行备份。当数据库发生故障时，可以使用这些备份文件进行还原。通过备份的数据文件及归档日志等文件，可以对数据库进行完全恢复。

支持的物理备份工具包括：

- vb_basebackup
  - vb_basebackup工具用于进行基础的物理备份。该工具可以将整个数据库实例的数据进行备份。
  - vb_basebackup不支持增量备份。结合PITR恢复，可恢复全量备份时间点后的某一时间点。
  - vb_basebackup是对数据库按二进制进行备份，因此恢复时可以直接拷贝替换原有的文件，或者直接在备份的库上启动数据库。
- vb_probackup
  - vb_probackup是一个用于管理Vastbase数据库物理备份和恢复的工具。可以实现对Vastbase实例的全量备份和增量备份。
  - 支持备份外部目录的内容，如脚本文件、配置文件、日志文件、dump文件等。
  - vb_probackup通过不同子命令实现了多种功能，如：备份和恢复、备份实例的管理、设置备份的留存策略等。

vb_basebackup

功能描述

VexDB部署成功后，在数据库运行的过程中，可能会遇到各种问题及异常状态。VexDB提供了vb_basebackup工具做基础的物理备份。该工具可以将整个实例的数据进行备份。

vb_basebackup的实现目标是对服务器数据库文件的二进制进行拷贝，其实现原理是使用了复制协议。远程执行vb_basebackup时，需要使用系统管理员账户。vb_basebackup当前支持热备份模式和压缩格式备份模式。

> **说明**
>
> - 热备份是指在数据库运行状态下进行备份操作，备份所需的时间短。
> - 冷备份是指停止正在运行的数据库，分离数据库，直接复制数据库文件。

前提条件

- 可以正常连接VexDB数据库。
- 备份过程中用户权限没有被回收。
- pg_hba.conf中需要配置允许复制连接，且该连接必须由一个系统管理员建立。
- 如果xlog传输模式为stream模式，需要配置max_wal_senders的数量，至少有一个可用。
- 如果xlog传输模式为fetch模式，需要把wal_keep_segments参数设置得足够高，这样在备份末尾之前日志不会被移除。

- 在进行还原时，需要保证各节点备份目录中存在备份文件，若备份文件丢失，则需要从其他节点进行拷贝。

注意事项

- 暂不支持备份文件增量恢复。如需使用此工具恢复至全备后的任意时间点，可以采用PITR指定时间点恢复。
- 恢复后需要检查数据库中的链接文件是否链接到正确的文件。
- vb_basebackup仅支持主机和备机的全量备份，不支持增量。
- vb_basebackup当前支持热备份模式和压缩格式备份模式。
- 若打开增量检测点功能且打开双写，vb_basebackup也会备份双写文件。
- vb_basebackup在备份包含绝对路径的表空间时，如果在同一台机器上进行备份，可以通过-T，–tablespace-mapping重定向表空间路径或使用tar格式进行备份。
- 若pg_xlog目录为软链接，备份时将不会建立软链接，会直接将数据备份到目的路径的pg_xlog目录下。
- 备份过程中收回用户备份权限，可能导致备份失败或者备份数据不可用。
- 如果因为网络临时故障等原因导致Server端无法应答，vb_basebackup将在最长等待120秒后退出。
- 远程备份不支持跨CPU架构进行备份。

语法格式

```
vb_basebackup [OPTION]...
```

控制输出的选项:

```
-D, --pgdata=DIRECTORY receive base backup into directory
-F, --format=p|t       output format (plain (default), tar)
-r, --max-rate=RATE    maximum transfer rate to transfer data directory            (in kB/s, or use suffix "K" or "M" or "G")
-T, --tablespace-mapping=OLDDIR=NEWDIR
                       relocate tablespace in OLDDIR to NEWDIR
-x, --xlog             include required WAL files in backup (fetch mode)
-X, --xlog-method=fetch|stream
                       include required WAL files with specified method
-z, --gzip             compress tar output
-Z, --compress=0-9     compress tar output with given compression level
```

常用选项:

```
-c, --checkpoint=fast|spread
                       set fast or spread checkpointing
-l, --label=LABEL      set backup label
-P, --progress         show progress information
-v, --verbose          output verbose messages
-V, --version          output version information, then exit
-?, --help             show this help, then exit
```

连接选项：

```
-h, --host=HOSTNAME    database server host or socket directory
-p, --port=PORT        database server port number
-s, --status-interval=INTERVAL
                       time between status packets sent to server (in seconds)
-t, --rw-timeout=RW_TIMEOUT
                       read-write timeout during idle connection.(in seconds)
-U, --username=NAME    connect as specified database user
-w, --no-password      never prompt for password
-W, --password         force password prompt (should happen automatically)
-K, --wal_directory    set the target directory to save the wal files
```

参数说明

控制输出的选项

- -D, –pgdata=DIRECTORY

  备份文件输出的目录，必选项，备份目录需提前创建。

- -F,–format=p|t

  设置输出格式为plain（默认）或者tar。没有设置该参数的情况下，默认–format=plain。plain格式把输出写成平面文件，使用和当前数据目录和表空间相同的布局。当集簇没有额外表空间时，整个数据库将被放在目标目录中。如果集簇包含额外的表空间，主数据目录将被放置在目标目录中，但是所有其他表空间将被放在它们位于服务器上的相同的绝对路径中。tar模式将输出写成目标目录中的 tar 文件。主数据目录将被写入到一个名为base.tar的文件中，并且其他表空间将被以其 OID 命名。生成的tar包，需要用gs_tar命令解压。

- -r, –max-rate=RATE

  备份文件的最大传输速率，未指定时最大传输速率为90MB/s。

  取值范围：0或者[32,2147483647]，配置为0表示不限速。

  默认单位为千字节每秒，或者通过“K”（千字节）、“M”（兆字节）和“G”（吉字节）显式指定每秒最大传输速率的单位。

- -T，–tablespace-mapping=olddir=newdir

在备份期间将目录olddir中的表空间重定位到newdir中。为使之有效，olddir必须正好匹配表空间所在的路径（但如果备份中没有包含olddir中的表空间也不是错误）。olddir和newdir必须是绝对路径。如果一个路径凑巧包含了一个=符号，可用反斜线对它转义。对于多个表空间可以多次使用这个选项。

VexDB自带两个表空间：pg_default和pg_global，-T选项适用于除两者之外的用户自定义表空间。

- 如果备份格式为plain，且备份实例中包含自定义表空间，那么备份命令中必须指定有效的-T指令。在恢复过程中，程序会自动完成表空间目录的重定向。
- 如果备份格式为tar，那么无论备份实例中是否包含自定义表空间，都无需显式指定-T。存在自定义表空间时，备份结果中会包含以表空间OID命名的若干tar包，恢复时可以通过修改自动生成的map文件来调整表空间映射。如果不修改自定义表空间的映射关系，则它们仍使用原本的目录（location）。具体示例请查阅实践四：备份格式为tar，带自定义表空间。

- -x，–xlog

  在备份中包含所需的WAL文件（fetch模式）。

  使用这个选项等效于在fetch方法中使用-X。

- -X，–xlog-method=fetch|stream

  设置xlog传输方式。没有设置该参数的情况下，默认–xlog-method=stream。在备份中包括所需的预写式日志文件（WAL文件）。这包括所有在备份期间产生的预写式日志。fetch方式在备份末尾收集预写式日志文件。因此，有必要把wal_keep_segments参数设置得足够高，这样在备份末尾之前日志不会被移除。如果在要传输日志时它已经被轮转，备份将失败并且是不可用的。stream方式在备份被创建时流传送预写式日志。这将开启一个到服务器的第二连接并且在运行备份时并行开始流传输预写式日志。因此，它将使用最多两个由max_wal_senders参数配置的连接。只要客户端能保持接收预写式日志，使用这种模式不需要在主控机上保存额外的预写式日志。

  使用tar格式压缩备份时，xlog传输方式不支持设为stream。

- -z, –gzip

  启用对 tar 文件输出的 gzip 压缩，使用默认的压缩级别。只有使用 tar 格式时压缩才可用，并且会在所有tar文件名后面自动加上后缀.gz。

- -Z, –compress=0-9

  启用对 tar 文件输出的 gzip 压缩，并且制定压缩级别（0 到 9，0 是不压缩，9 是最佳压缩）。只有使用 tar 格式时压缩才可用，并且会在所有tar文件名后面自动加上后缀.gz。

常用参数

- -c，–checkpoint=fast|spread

  设置检查点模式为fast或者spread（默认）。

- -l，–label=LABEL

  为备份设置标签。

- -P，–progress

启用进展报告。

- -v，–verbose

  启用冗长模式，打印更多信息。

- -V，–version

  打印版本后退出。

- -?，–help

  显示vb_basebackup命令行参数。

连接参数

- -h，–host=HOSTNAME

  指定正在运行服务器的主机名或者Unix域套接字的路径。

- -p，–port=PORT

  指定数据库服务器的端口号。可以通过port参数修改默认端口号。

- -s，–status-interval=INTERVAL

  发送到服务器的状态包的时间（以秒为单位）。

- -t, –rw-timeout=RW_TIMEOUT

  设置备份期间checkpoint的时间限制，默认限制时间为120s。当数据库全量checkpoint耗时较长时，可以适当增大–rw-timeout限制时间。

- -U，–username=NAME

  指定连接数据库的用户。

- -w，–no-password

  不出现输入密码提示。

- -W，–password

  当使用-U参数连接本地数据库或者连接远端数据库时，可通过指定该选项出现输入密码提示。

- -K, –wal_directory

  设置存放wal文件的目标目录。设置wal文件保存的目标目录，指定预写日志目录的位置。Waldir必须是一个绝对路径。只有在备份为plain模式时才能指定预写日志目录。

恢复流程

当数据库发生故障时需要从备份文件进行恢复。

因为vb_basebackup是对数据库按二进制进行备份，因此恢复时可以直接拷贝替换原有的文件，或者直接在备份的库上启动数据库。

> **说明**
>
> - 若当前数据库实例正在运行，直接从备份文件启动数据库可能会存在端口冲突，这时需要修改配置文件的port参数，或者在启动数据库时指定一下端口。
> - 若当前备份文件为主备数据库，可能需要修改一下主备之间的复制连接。即配置文件中的postgresql.conf中的replconninfo1、replconninfo2等。
> - 若配置文件postgresql.conf的参数data_directory打开且有配置，当使用备份目录启动数据库时候，data_directory和备份目录不同会导致启动失败。可以修改data_directory的值为新的数据目录，或者注释掉该参数。

方法一，在恢复时拷贝替换原有文件，参考步骤如下：

1. 停止数据库服务器。
2. 将原数据库和所有表空间复制到另外一个位置，以备后面需要。
3. 清理原库中的所有或部分文件。

   若不需要留存原数据库的数据文件，步骤2、3也可以直接用备份文件覆盖原数据库目录。
4. 使用数据库系统用户权限从备份中还原需要的数据库文件。
5. 若数据库中存在链接文件，需要修改使其链接到正确的文件。
6. 重启数据库服务器，并检查数据库内容，确保数据库已经恢复到所需的状态。

方法二，在备份的数据库上启动数据库，参考步骤如下：

1. 停止数据库服务器。
2. 若数据库中存在链接文件，需要修改使其链接到正确的文件。
3. 在备份的数据库上启动数据库，启动时用-D指定备份的数据目录。
4. 检查数据库内容，确保数据库已经恢复到所需的状态。

vb_probackup

功能描述

vb_probackup是一个用于管理VexDB数据库物理备份和恢复的工具。它可以对VexDB实例进行定期备份，以便在数据库出现故障时能够恢复服务器。

vb_probackup工具的主要功能如下：

- 备份：vb_probackup工具采用物理备份的方式进行备份，并且支持备份压缩。对数据库进行备份，可以增强数据库的抗风险能力。
  - 可用于备份单机数据库，也可对主机或者主节点数据库备机进行备份。
  - 可通过-E选项备份外部目录的内容，如脚本文件、配置文件、日志文件、dump文件、表空间等。
  - 支持全量备份、增量备份、备份合并、远程备份、通过定时任务脚本实现定期备份。
  - 除了基础的备份恢复操作以外，vb_probackup通过不同子命令实现了备份实例的管理、设置备份的留存策略等功能，全部用法详见命令说明。
  - 在进行全量备份前，若开启归档（即archive_mode设置为on）需要配置归档目录。
- 恢复：vb_probackup工具支持全量恢复与增量恢复，提供将数据库恢复到之前状态的能力。vb_probackup工具提供的恢复能力，是数据库安全可靠的重要保证。
- 验证：vb_probackup提供了验证备份完整性和一致性的功能，确保备份数据的可靠性。

以上介绍的主要功能，用户在使用时可以通过添加不同的参数，实现更准确的控制。

前提条件

- 可以正常连接VexDB数据库。
- 在配置文件postgresql.conf中手动设置参数enable_cbm_tracking为on，或通过vb_guc工具进行设置。
- 为了防止xlog在传输结束前被清理，请适当调高postgresql.conf文件中wal_keep_segments的值。

注意事项

- 备份必须由运行数据库服务器的操作系统用户执行。
- 备份和恢复的数据库服务器的主版本号必须相同。
- 在进行全量备份前，若开启归档（即archive_mode设置为on）需要配置归档目录。
- 如果要通过ssh在远程模式下备份数据库，需要在本地和远程主机安装相同主版本的数据库，并通过ssh-copy-id remote_user@remote_host命令设置本地主机备份用户和远程主机数据库用户的无密码ssh连接。

- 远程模式下只能执行add-instance、backup、restore子命令。
- 使用restore子命令前，应先停止数据库进程。
- 当存在用户自定义表空间时，备份时要加上-E或者–include-all参数，否则，该表空间不会被备份。

  外部表空间文件页面级别的增量备份，有助于减少备份文件占用的空间。
- 当存在用户自定义表空间时，如果该表空间的路径不在$PGDATA目录下，备份时需要将自定义表空间的绝对路径位置用 -E 指定，不然不会备份；恢复时需要使用–tablespace-mapping以及–external-mapping指示恢复映射。
- 当备份的规模比较大或在备份同时执行业务时，为了防止备份过程中timeout发生，请适当调整postgresql.conf文件的参数session_timeout、wal_sender_timeout。并且在备份的命令行参数中适当调整参数–rw-timeout的值。
- vb_probackup备份任务发生错误时，备份进程并不会完全退出，并且用vb_probackup show命令查看备份集状态时对应记录显示为ERROR，此时需要指定backup_id，手动删除状态为ERROR的备份集。
- vb_probackup不支持同时执行多个备份任务。
- 恢复时，使用-T选项把备份中的外部目录重定向到新目录时，请同时指定参数–external-mapping。
- 当使用远程备份时，请确保远程机器和备份机器的时钟同步，以防止使用–recovery-target-time恢复的场合，启动VexDB时有可能会失败。
- 当远程备份有效时(remote-proto=ssh)，请确保-h和–remote-host指定的是同一台机器。当远程备份无效时，如果指定了-h选项，请确保-h指定的是本机地址或本机主机名。
- 当前暂不支持备份逻辑复制槽。
- 增量备份恢复后，之前创建的逻辑复制槽不可用，需删除重建。
- 备份时，请确保服务器用户对备份的目录下所有文件有读写的权限，以防止在恢复时因权限不足的问题而失败。
- 备份将执行checkpoint与xlog switch操作，此行为将产生新的xlog，并提交事务。一主一备或一主多备场景备份时，若配置文件中synchronous_commit设置为on，备机关停可能会导致主机同步提交事务失败，进而导致备份失败。此场景下，请确认各节点状态正常，或将synchronous_commit设置为off以避免备份失败。
- 在开启enable_cbm_tracking后，不能直接执行增量备份，需要先执行全量备份，即使在开启参数之前已经执行过全量备份。
- 不支持跨CPU架构的远程备份。

命令说明

- init： 初始化备份路径backup-path中的备份目录，该目录将存储已备份的内容。backup-path必须为空目录。

```
vb_probackup init -B backup-path [--help]
```

- add_instance： 在备份路径backup-path内初始化一个新的备份实例，并生成pg_probackup.conf配置文件，该文件保存了指定数据目录pgdata-path的vb_probackup设置。

  若未指定 -D 选项，则默认为数据库原始数据目录 $PGDATA。

```
vb_probackup add-instance -B backup-path -D pgdata-path --instance=instance_name
            [-E external-directories-paths]
            [--remote-proto=protocol] [--remote-host=destination]
            [--remote-path=path] [--remote-user=username]
            [--remote-port=port] [--ssh-options=ssh_options]
            [--remote-libpath=libpath]
            [--enable-dss] [--instance-id=instance_id]
            [--vgname="vgdata,vglog"] [--socketpath=socketpath]
            [--help]
```

- del-instance： 在备份路径backup-path内删除指定实例相关的备份内容。

```
vb_probackup del-instance -B backup-path --instance=instance_name
            [--help]
```

- set-config： 将指定的连接、压缩、日志等相关设置添加到pg_probackup.conf配置文件中，或修改已设置的值。不推荐手动编辑pg_probackup.conf配置文件。

```
vbprobackup set-config -B backup-path --instance=instance_name
            [-D pgdata-path] [-E external-directories-paths]
            [--archive-timeout=timeout]
            [--retention-redundancy=retention-redundancy]
            [--retention-window=retention-window]
            [--wal-depth=wal-depth]
            [--compress-algorithm=compress-algorithm]
            [--compress-level=compress-level]
            [-d dbname] [-h host] [-p port] [-U username]
            [--log-level-console=log-level-console]
            [--log-level-file=log-level-file]
            [--log-filename=log-filename]
            [--error-log-filename=error-log-filename]
            [--log-directory=log-directory]
            [--log-rotation-size=log-rotation-size]
            [--log-rotation-age=log-rotation-age]
            [--remote-proto=protocol] [--remote-host=destination]
            [--remote-path=path] [--remote-user=username]
            [--remote-port=port] [--ssh-options=ssh_options]
            [--remote-libpath=libpath]
            [--enable-dss] [--instance-id=instance_id]
            [--vgname="vgdata,vglog"] [--socketpath=socketpath]
            [--help]
```

- set-backup： 将备份相关设置添加到backup.control配置文件中，或修改已设置的值。

```
vb_probackup set-backup -B backup-path --instance=instance_name -i backup-id
            [--note=text] [--ttl=interval] [--expire-time=time]
            [--help]
```

- show-config： 显示位于备份目录中的pg_probackup.conf配置文件的内容。可以通过指定–format=json选项，以json格式显示。默认情况下，显示为纯文本格式。

```
vb_probackup show-config -B backup-path --instance=instance_name
            [--format=plain|json]
            [--help]
```

- show： 显示备份目录的内容。

```
vb_probackup show -B backup-path
            [--instance=instance_name [-i backup-id]]
            [--archive] [--format=plain|json]
            [--do-enc]
            [--key-idx
            [--help]
```

如果指定了instance_name和backup_id，则显示该备份的详细信息。可以通过指定–format=json选项，以json格式显示。默认情况下，备份目录的内容显示为纯文本格式。

如果不指定instance_name和backup_id，则返回当前实例中的所有备份信息，每一条记录代表一个备份集。

- backup： 创建指定实例的备份。

```
vb_probackup backup -B backup-path --instance=instance_name -b backup-mode
            [-D pgdata-path] [-C] [-S slot-name] [--temp-slot]
            [-s session-timeout]
            [--backup-pg-log] [-j threads_num] [--progress]
            [--no-validate] [--skip-block-validation]
            [-E external-directories-paths]
            [--no-sync] [--note=text]
            [--archive-timeout=timeout]
            [--log-level-console=log-level-console]
            [--log-level-file=log-level-file]
            [--log-filename=log-filename]
            [--error-log-filename=error-log-filename]
            [--log-directory=log-directory]
            [--log-rotation-size=log-rotation-size]
            [--log-rotation-age=log-rotation-age]
            [--delete-expired] [--delete-wal] [--merge-expired]
            [--retention-redundancy=retention-redundancy]
            [--retention-window=retention-window]
            [--wal-depth=wal-depth] [--dry-run]
            [--compress-algorithm=compress-algorithm]
            [--compress-level=compress-level]
            [--compress]
            [-d dbname] [-h host] [-p port] [-U username] [-w] [-W password]
            [-t rwtimeout]
            [--remote-proto=protocol] [--remote-host=destination]
            [--remote-path=path] [--remote-user=username]
            [--remote-port=port] [--ssh-options=ssh_options]
            [--remote-libpath=libpath]
            [--stream]
            [--enable-dss] [--instance-id=instance_id]
            [--vgname="vgdata,vglog"] [--socketpath=socketpath]
            [--ttl=interval] [--expire-time=time]
            [--backup-pg-replslot]
            [--do-enc]
            [--key-idx
            [--help]
```

- restore： 从备份目录backup-path中的备份副本恢复指定实例。

  使用 restore 子命令前，应先停止 vexdb进程。

  如果指定了恢复目标相关参数，vb_probackup将查找最近的备份并将其还原到指定的恢复目标。否则，使用最近一次备份。

```
vb_probackup restore -B backup-path --instance=instance_name
            [-D pgdata-path] [-i backup-id] [-j threads_num] [--progress]
            [--force] [--no-sync] [--no-validate] [--skip-block-validation]
            [--external-mapping=OLDDIR=NEWDIR] [-T OLDDIR=NEWDIR]
            [--skip-external-dirs] [-I incremental_mode]
            [--recovery-target-time=time|--recovery-target-xid=xid
            |--recovery-target-lsn=lsn|--recovery-target-name=target-name]
            [--recovery-target-inclusive=boolean]
            [--recovery-target=latest]
            [--remote-proto=protocol] [--remote-host=destination]
            [--remote-path=path] [--remote-user=username]
            [--remote-port=port] [--ssh-options=ssh_options]
            [--remote-libpath=libpath]
            [--enable-dss] [--instance-id=instance_id]
            [--vgname="vgdata,vglog"] [--socketpath=socketpath]
            [--log-level-console=log-level-console]
            [--log-level-file=log-level-file]
            [--log-filename=log-filename]
            [--error-log-filename=error-log-filename]
            [--log-directory=log-directory]
            [--log-rotation-size=log-rotation-size]
            [--log-rotation-age=log-rotation-age]
            [--archive-host=hostname] [--archive-path=path]
            [--archive-port=port] [--archive-user=username]
            [--force-overwrite]
            [-K wal-directory]
            [--do-enc]
            [--key-idx
            [--help]
```

- merge： 将指定的增量备份与其父完全备份之间的所有增量备份合并到父完全备份。父完全备份将接收所有合并的数据，而已合并的增量备份将作为冗余被删除。

```
vb_probackup merge -B backup-path --instance=instance_name -i backup-id
            [-j threads_num] [--progress]
            [--log-level-console=log-level-console]
            [--log-level-file=log-level-file]
            [--log-filename=log-filename]
            [--error-log-filename=error-log-filename]
            [--log-directory=log-directory]
            [--log-rotation-size=log-rotation-size]
            [--log-rotation-age=log-rotation-age]
            [--do-enc]
            [--key-idx
            [--help]
```

- delete： 删除指定备份，或删除不满足当前保留策略的备份。

```
vb_probackup delete -B backup-path --instance=instance_name
            [-i backup-id | --delete-expired | --merge-expired | --status=backup_status]
            [--delete-wal] [-j threads_num] [--progress]
            [--retention-redundancy=retention-redundancy]
            [--retention-window=retention-window]
            [--wal-depth=wal-depth] [--dry-run]
            [--log-level-console=log-level-console]
            [--log-level-file=log-level-file]
            [--log-filename=log-filename]
            [--error-log-filename=error-log-filename]
            [--log-directory=log-directory]
            [--log-rotation-size=log-rotation-size]
            [--log-rotation-age=log-rotation-age]
            [--do-enc]
            [--key-idx
            [--help]
```

- validate： 验证恢复数据库所需的所有文件是否存在且未损坏。如果未指定instance_name，vb_probackup将验证备份目录中的所有可用备份。如果指定instance_name而不指定任何附加选项，vb_probackup将验证此备份实例的所有可用备份。如果指定了instance_name并且指定backup-id或恢复目标相关选项，vb_probackup将检查是否可以使用这些选项恢复数据库。

```
vb_probackup validate -B backup-path
            [--instance=instance_name] [-i backup-id]
            [-j threads-num] [--progress] [--skip-block-validation]
            [--recovery-target-time=time|--recovery-target-xid=xid
            |--recovery-target-lsn=lsn|--recovery-target-name=target-name]
            [--recovery-target-inclusive=boolean]
            [--log-level-console=log-level-console]
            [--log-level-file=log-level-file]
            [--log-filename=log-filename]
            [--error-log-filename=error-log-filename]
            [--log-directory=log-directory]
            [--log-rotation-size=log-rotation-size]
            [--log-rotation-age=log-rotation-age]
```

- archive-push： 将WAL文件复制到备份目录的相应子目录中。

```
vb_probackup archive-push -B backup-path --instance=instance_name
--wal-file-path=wal-file-path
--wal-file-name=wal-file-name
[-j num-threads][--batch-size=batch_size]
[--archive-timeout=timeout]
[--no-ready-rename][--no-sync][--overwrite][--compress]
[--compress-algorithm=compression-algorithm] [--compress-level=compression_level]
[--remote-proto][--remote-host]
[--remote-port][--remote-path][--remote-user]
[--ssh-options]
[--help]
```

- archive-get： 将WAL文件从备份目录对应的子目录复制到数据库的预写日志位置。

进行restore时，vb_probackup会自动设置该命令作为restore_command的一部分，不需要手动设置；远程模式时，自动设置的archive-get命令不一定能直接使用，需要核对recovery.conf中的archive-get命令是否正确，可手动修改。

```
vb_probackup archive-get -B backup-path --instance=instance_name
            --wal-file-path=wal-file-path
            --wal-file-name=wal-file-name
            [-j num-threads] [--batch-size=batch_size]
            [--no-validate-wal]
            [--remote-proto] [--remote-host]
            [--remote-port] [--remote-path] [--remote-user]
            [--ssh-options]
            [--do-enc]
            [--key-idx
            [--help]
```

- 其它功能

  打印vb_probackup版本。

```
vb_probackup -V|--version
vb_probackup version
```

  显示vb_probackup命令的摘要信息。如果指定了vb_probackup的子命令，则显示可用于此子命令的参数的详细信息。

```
vb_probackup -?|--help
vb_probackup help [command]
```

参数说明

通用参数

- command

  vb_probackup除version和help以外的子命令：init、add-instance、del-instance、set-config、set-backup、show-config、show、backup、restore、merge、delete、validate、archive-push、archive-get。

- -?, –help

  显示vb_probackup命令行参数的帮助信息，然后退出。子命令中只能使用–help，不能使用-?。

- -V, –version

  打印vb_probackup版本，然后退出。

- -B backup-path, –backup-path=backup-path

  备份的路径。

  系统环境变量：$BACKUP_PATH

- -D pgdata-path, –pgdata=pgdata-path

  数据目录的路径。

  系统环境变量：$PGDATA

- –instance=instance_name

  自定义的备份实例名。通过指定实例名，备份工具会根据元数据查到该实例的信息。

- -i backup-id, –backup-id=backup-id

  备份的唯一标识。

- –format=format

  指定显示备份信息的格式，支持plain和json格式。

  默认值：plain

- –status=backup_status

  删除指定状态的所有备份，包含以下状态：

  - OK：备份已完成且有效。
  - DONE：备份已完成但未经过验证。
  - RUNNING：备份正在进行中。
  - MERGING：备份正在合并中。
  - DELETING：备份正在删除中。
  - CORRUPT：部分备份文件已损坏。
  - ERROR：由于意外错误，备份失败。
  - ORPHAN：由于其父备份之一已损坏或丢失，备份无效。

- -j threads_num, –threads=threads_num

  设置备份、还原、合并进程的并行线程数。

- –archive

  显示WAL归档信息。

- –progress

  显示进度。

- –note=text

  给备份添加note。

备份相关参数

- -b backup-mode, –backup-mode=backup-mode

  指定备份模式，支持FULL、PTRACK和PTRACK_FROM_FULL。

备份模式	备份内容	说明
FULL	全量备份	全量备份包含所有数据文件。
PTRACK	增量备份	仅备份自上次备份（可以是全量备份或者增量备份）以来发生变化的页/块。 PTRACK 模式下可以动态跟踪页面的更改。每次更新数据页时，都会在位图中进行相应的标记。
PTRACK_FROM_FULL	差异备份	差异备份表示自最近一次全量备份以来的所有变化的数据页/块。

- -C, –smooth-checkpoint

  将检查点在一段时间内展开。默认情况下，vb_probackup会尝试尽快完成检查点。

- –stream

  从数据库服务器通过stream流处理形式传输文件，生成包括所有必需的WAL文件的STREAM备份。

  当前仅支持 stream 流处理形式，因此该参数可忽略。

- -S slot-name, –slot=slot-name

  指定WAL流处理的复制slot。

- –temp-slot

  在备份的实例中为WAL流处理创建一个临时物理复制slot，它确保在备份过程中，所有所需的WAL段仍然是可用的。

  默认的slot名为pg_probackup_slot，可通过选项–slot/-S更改。

- –backup-pg-log

  将日志目录包含到备份中。此目录通常包含日志消息。默认情况下包含日志目录，但不包含日志文件。如果修改了默认的日志路径，需要备份日志文件时可使用-E参数进行备份，使用方法见下文。

- -E external-directories-paths, –external-dirs=external-directories-paths

  该参数用于将用户指定的特定目录纳入到备份操作的范围之中。在实际应用场景中，当表空间、脚本文件、sql 转储文件以及配置文件存储于默认数据目录之外时，通过使用此选项，能够有效地将这些文件所在的目录备份。

  如果用户需要备份多个外部目录，可采用冒号（:）作为分隔符，依次列出各个目录的路径。例如，若有三个外部目录，其路径分别为 “/home/user/scripts”、“/var/backups/sql_dumps” 和 “/etc/config_files”，则在使用该参数时，应输入 “/home/user/scripts:/var/backups/sql_dumps:/etc/config_files”。

- –include-all

  备份包含所有外部表空间的数据。

  该选项可以与-E同时使用，程序会自动去除重复的外部表空间目录。

- –skip-block-validation

  关闭块级校验，加快备份速度。

- –no-validate

  在完成备份后跳过自动验证。

- –no-sync

  不将备份文件同步到磁盘。

- –note

  给备份添加note。

- –archive-timeout=timeout

  以秒为单位设置流式处理的超时时间。

  默认值：300

- -t rw-timeout

  以秒为单位的连接的超时时间。

  默认值: 120

- –backup-to-stdout

  将备份以流形式发送到STDOUT（标准输出），而不是直接将文件复制到备份目录。

  使用时需同时指定标准输出的位置（绝对路径或相对路径）。

  > **说明**
  > - 标准输入输出文件支持.bak、.sql、.txt、.tar.gz格式。
  > - 流式备份不支持从备库备份。
  > - 流式备份不支持通过MERGE命令合并全量和增量备份集。
  > - 使用此模式进行增量备份时，其依赖的父备份需要通过标准输入传递。

恢复相关参数

- -I, –incremental-mode=none|checksum|lsn

  若PGDATA中可用的有效页没有修改，则重新使用它们，在增量恢复时，需手动指定该参数为checksum或lsn。

  默认值：none

- –external-mapping=OLDDIR=NEWDIR

  在恢复时，将包含在备份中的外部目录从_OLDDIR_重新定位到_NEWDIR_目录。_OLDDIR_和_NEWDIR_都必须是绝对路径。如果路径中包含“=”，则使用反斜杠转义。此选项可为多个目录多次指定。目前只支持指定文件系统中的目录，不支持指定共享存储中的目录。

- -T OLDDIR=NEWDIR, –tablespace-mapping=OLDDIR=NEWDIR

  在恢复时，将表空间从_OLDDIR_重新定位到_NEWDIR_目录。_OLDDIR_和_NEWDIR_必须都是绝对路径。如果路径中包含“=”，则使用反斜杠转义。多个表空间可以多次指定此选项。当-T参数值为外部表空间目录时，必须搭配–external-mapping参数使用。目前只支持指定文件系统中的目录，不支持指定共享存储中的目录。

- –skip-external-dirs

  跳过备份中包含的使用-external-dirs选项指定的外部目录。这些目录的内容将不会被恢复。

- –skip-block-validation

  跳过块级校验，以加快验证速度。在恢复之前的自动验证期间，将仅做文件级别的校验。

- –no-validate

  跳过备份验证。

- –force

  允许忽略备份的无效状态。如果出于某种原因需要从损坏的或无效的备份中恢复数据，可以使用此标志。请谨慎使用。

- –force-overwrite

  指定非空目录进行恢复。

- –restore-from-stdin

  从标准输入（STDIN）以流形式恢复数据。

  流式备份仅支持单线程恢复。

  恢复的目标目录必须为空或不存在。（此参数不能与–force-overwrite同时使用。）

- –reuse-old-dirs

  在恢复时重用外部表空间目录，无需通过–tablespace-mapping设置外部目录映射。

  - 未指定该选项时，备份文件中包含的所有外部表空间目录，都必须通过–external-mapping和–tablespace-mapping选项明确指定目录映射关系。如果存在未指定映射关系的外部表空间，或者备份集中指定的外部表空间不完整时，恢复过程会立即报错并终止。
  - 指定该选项时，恢复过程会忽略所有–external-mapping未提及的外部表空间，并重用原有的外部目录。

    只有确保备份文件创建后与恢复之间没有发生数据变更的外部表空间，才可以使用–reuse-old-dirs重用，否则会产生数据丢失，请谨慎使用。

恢复目标相关参数

> **说明**
>
> 当前不支持配置连续的WAL归档的PITR，因而使用这些参数会有一定限制，具体如下描述。 如果需要使用持续归档的WAL日志进行PITR恢复，请按照下面描述的步骤： 1. 将物理备份的文件替换目标数据库目录。 2. 删除数据库目录下pg_xlog中的所有文件。 3. 将归档的WAL日志文件复制到pg_xlog文件中（此步骤可以省略，通过配置recovery.conf恢复命令文件中的restore_command项替代）。 4. 在数据库目录下创建恢复命令文件recovery.conf，指定数据库恢复的程度。 5. 启动数据库。 6. 连接数据库，查看是否恢复到希望预期的状态。若已经恢复到预期状态，通过pg_xlog_replay_resume()指令使主节点对外提供服务。

- –recovery-target-lsn=lsn

  指定要恢复到的lsn，当前只能指定备份的stop lsn。

- –recovery-target-name=target-name

  指定要将数据恢复到的已命名的保存点，保存点可以通过查看备份中recovery-name字段得到。

当指定的恢复点不存在或者无效时，默认回放到最新日志的位置。

- –recovery-target-time=time

  指定要恢复到的时间，当前只能指定备份中的recovery-time。

- –recovery-target-xid=xid

  指定要恢复到的事务ID，当前只能指定备份中的recovery-xid。

- –recovery-target-inclusive=overwriteboolean

  当该参数指定为true时，恢复目标将包括指定的内容。当该参数指定为false时，恢复目标将不包括指定的内容。该参数必须和–recovery-target-name、–recovery-target-time、–recovery-target-lsn或–recovery-target-xid一起使用。

- –recovery-target=latest

  恢复未归档xlog。当数据库进程异常杀掉后导致xlog未归档时，指定latest自动恢复未归档的xlog。

留存相关参数

可以和backup/delete命令一起使用这些参数。

- –retention-redundancy=retention-redundancy

  指定在数据目录中留存的完整备份的数量。该参数必须设置为大于或等于 0 的整数。当设置为 0 时，表示禁用此留存策略，即不会依据该参数来限制留存的完整备份数量；若设置为 N（N 为大于 0 的整数），则意味着数据目录中会留存最近的 N 个完整备份。

  默认值：0

- –retention-window=retention-window

  指定留存的天数。必须为正整数。0表示禁用此设置。

  默认值：0

- –wal-depth=wal-depth

  用于控制 WAL 日志保留策略。该参数必须设置为大于或等于 0 的整数。当设置为 0 时，意味着不会额外保留备份链上的 WAL 日志；若设置为 N（N 为大于 0 的整数），则表示会保留每个备份链上，距离当前时间最近的 N 个有效备份所关联的 WAL 日志。

  此参数的作用在于，当执行删除旧备份的操作时，会确保保留与最近 N 个备份相关联的 WAL 日志。这些保留下来的 WAL 日志能够保证最近的这 N 个备份仍然可以用于时间点恢复（PITR，Point-In-Time Recovery）操作，即可以将数据库恢复到过去某个特定时间点的状态。

  默认值：0

- –delete-wal

  从任何现有的备份中删除不需要的WAL文件。

- –delete-expired

  删除不符合pg_probackup.conf配置文件中定义的留存策略的备份。

- –merge-expired

  将满足留存策略要求的最旧的增量备份与其已过期的父备份合并。

- –dry-run

  显示所有可用备份的当前状态，不删除或合并过期备份。

固定备份相关参数

- –ttl=interval

  指定从恢复时间开始计算，备份要固定的时间量。必须为正整数。0表示取消备份固定。支持的单位：ms、 s、 min、 h、 d（默认为s）。

  例如：–ttl=30d。

- –expire-time=time

  指定备份固定失效的时间戳。必须是ISO-8601标准的时间戳。需要和留存相关参数配合使用。

  例如：–expire-time= ‘2020-01-01 00:00:00+03’

  如果要将某些备份从已建立的留存策略中排除，可以和backup/set-backup命令一起使用这些参数。

日志相关参数

日志级别：verbose、log、info、warning、error和off。

- –log-level-console=log-level-console

  设置要发送到控制台的日志级别。每个级别都包含其后的所有级别。级别越高，发送的消息越少。指定off级别表示禁用控制台日志记录。

  默认值：info

- –log-level-file=log-level-file

  设置要发送到日志文件的日志级别。每个级别都包含其后的所有级别。级别越高，发送的消息越少。指定off级别表示禁用日志文件记录。

默认值：off

- –log-filename=log-filename

  指定要创建的日志文件的文件名。文件名可以使用strftime模式，因此可以使用%-escapes指定随时间变化的文件名。

  例如，如果指定了“pg_probackup-%u.log”模式，则pg_probackup为每周的每一天生成单独的日志文件，其中%u替换为相应的十进制数字，即pg_probackup-1.log表示星期一；pg_probackup-2.log表示星期二，以此类推。如果指定了–log-level-file参数启用日志文件记录，则该参数有效。

  默认值：“pg_probackup.log”

- –error-log-filename=error-log-filename

  指定仅用于error日志的日志文件名。指定方式与–log-filename参数相同。此参数用于故障排除和监视。

- –log-directory=log-directory

  指定创建日志文件的目录。必须是绝对路径。此目录会在写入第一条日志时创建。

  默认值：$BACKUP_PATH/log

- –log-rotation-size=log-rotation-size

  指定单个日志文件的最大大小。如果达到此值，则启动vb_probackup命令后，日志文件将循环，但help和version命令除外。0表示禁用基于文件大小的循环。支持的单位：KB、MB、GB、TB（默认为KB）。

  默认值：0

- –log-rotation-age=log-rotation-age

  单个日志文件的最大生命周期。如果达到此值，则启动vb_probackup命令后，日志文件将循环，但help和version命令$BACKUP_PATH/log/log_rotation目录下保存最后一次创建日志文件的时间。0表示禁用基于时间的循环。支持的单位：ms、 s、 min、 h、 d（默认为min）。

  默认值：0

连接相关参数

可以和backup命令一起使用这些参数。

- -d dbname, –pgdatabase=dbname

  指定要连接的数据库名称。该连接仅用于管理备份进程，因此您可以连接到任何现有的数据库。如果命令行、PGDATABASE环境变量或pg_probackup.conf配置文件中没有指定此参数，则vb_probackup会尝试从PGUSER环境变量中获取该值。如果未设置PGUSER变量，则从当前用户名获取。

  系统环境变量：$PGDATABASE

- -h hostname, –pghost=hostname

  指定运行服务器的系统的主机名。如果该值以斜杠开头，则被用作到Unix域套接字的路径。

  系统环境变量：$PGHOST

  默认值：local socket

- -p port, –pgport=_port

  指定服务器正在侦听连接的TCP端口或本地Unix域套接字文件扩展名。

  系统环境变量：$PGPORT

  默认值：5432

- -U username, –pguser=username

  指定所连接主机的用户名。

  系统环境变量：$PGUSER

- -w, –no-password

  不出现输入密码提示。如果主机要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。

- -W, –password=password

  指定用户连接的密码。如果主机的认证策略是trust，则不会对系统管理员进行密码验证，即无需输入-W选项；如果没有-W选项，并且不是系统管理员，则会提示用户输入密码。

压缩相关参数

可以和backup命令一起使用这些参数。

- –compress-algorithm=compress-algorithm

  指定用于压缩数据文件的算法。

  取值包括zlib、pglz和none。如果设置为zlib或pglz，此选项将启用压缩。默认情况下，压缩功能处于关闭状态。

  当–compress-algorithm=zlib时，–compress-level参数取值不能为0。

  默认值：none

- –compress-level=compress-level

  指定压缩级别。该参数必须搭配–compress-algorithm一起使用，否则会报错。

- 0表示无压缩。
- 1表示压缩比最小，处理速度最快。
- 9表示压缩比最大，处理速度最慢。

可与–compress-algorithm选项一起使用。

取值范围：0~9

默认值：1

- –compress

以–compress-algorithm=zlib和–compress-level=1进行压缩。

远程模式相关参数

- –remote-proto=protocol

指定用于远程操作的协议。目前只支持SSH协议。取值包括：

- ssh：通过SSH启用远程备份模式。这是默认值。
- none：显式禁用远程模式。
- 如果指定了–remote-host参数，可以省略此参数。

- –remote-host=destination

指定要连接的远程主机的IP地址或主机名。

- –remote-port=port

指定要连接的远程主机的端口号。

默认值：22

- –remote-user=username

指定SSH连接的远程主机用户。如果省略此参数，则使用当前发起SSH连接的用户。

默认值：当前用户

- –remote-path=path

指定vb_probackup在远程系统的安装目录。

默认值：当前路径

- –remote-libpath=libpath

  指定vb_probackup在远程系统安装的lib库目录。

- –ssh-options=ssh_options

  指定SSH命令行参数的字符串。

  例如：–ssh-options= ‘-c cipher_spec -F configfile’

归档相关参数

- –overwrite

  覆盖已归档的WAL文件。

- –batch-size=batch_size

  需要复制的文件总数。

- –prefetch-dir=path

  预取 WAL文件的存储目录。

备份流程

1. 初始化备份目录。在指定的目录下创建backups/和wal/子目录，分别用于存放备份文件和WAL文件。

   ```
   vb_probackup init -B backup-path
   ```

   虽然指定目录下会创建wal子目录，但WAL文件（即pg_xlog文件）会存放在backups目录下。

2. 添加一个新的备份实例。vb_probackup可以在同一个备份目录下存放多个数据库实例的备份。

   ```
   vb_probackup add-instance -B backup-path -D pgdata-path --instance instance_name
   ```

3. 编辑配置文档postgresql.conf，进行归档设置。（如果需要使用持续归档的WAL日志进行PITR恢复，则必须开启归档。）

```
archive_mode = on
archive_command='vb_probackup archive-push -B backup-path --instance instance-name --wal-file-path %p --wal-file-name %f [remote_options]'
hot_standby = on
wal_level = hot_standby # 必须高于minimal
```

4. 创建指定实例的备份。在进行增量备份之前，必须至少创建一次全量备份。

```
vb_probackup backup -B backup-path --instance instance_name -b backup_mode
```

5. 从指定实例的备份中恢复数据。在恢复备份时，必须先将原始实例目录（参数-D后面的目录）下面的内容清空。

```
vb_probackup restore -B backup-path --instance instance_name -D pgdata-path -i backup_id
```

> **说明**
>
> - 在恢复备份前，必须先停止 vexdb进程，并且将原始实例目录（参数 -D 后面的目录）下面的内容清空。
> - 使用 restore 子命令前，应先停止 vexdb进程。
> - 如果因为网络临时故障等原因导致 server 端无应答，vb_probackup 将在等待 archive->timeout（默认300秒）后退出。

故障处理

问题描述	原因和解决方案
ERROR: query failed: ERROR: canceling statement due to conflict with recovery（错误：查询失败：由于与恢复操作冲突，正在取消语句命令）	原因：在备机上执行的操作正在访问存储行，主机上更改或者删除了对应的行，并将 xlog 在备机上重放，迫使备机上操作取消。解决方案：适当增加如下配置参数的值max_standby_archive_delaymax_standby_streaming_delay增加如下配置hot_standby_feedback = on

安全策略

本章节介绍了用于 VexDB 数据保护的安全策略，指导数据库管理员和用户更好的保证数据库的安全。

设置账户安全策略

背景信息

VexDB 为账户提供了自动锁定和解锁账户、手动锁定和解锁异常账户和删除不再使用的账户等一系列的安全措施，保证数据安全。

自动锁定和解锁账户

为了保证账户安全，如果用户输入密码次数超过一定次数（failed_login_attempts），系统将自动锁定该账户，默认值为5。次数设置越小越安全，但是在使用过程中会带来不便。

当账户被锁定时间超过设定值（password_lock_time），则当前账户自动解锁，默认值为1440分钟。时间设置越长越安全，但是在使用过程中会带来不便。

- 参数password_lock_time的取值范围：整型，最小值为1，最大值为525600，单位为分钟，默认值为1440分钟。
- 参数failed_login_attempts的取值范围：整型，最小值为1，最大值为1000，默认值为5。

参数failed_login_attempts表示在任意时候，如果输入密码错误的次数达到设定值则当前账户被锁定，password_lock_time分钟后被自动解锁。因此，只有两个配置参数都为正数时，才可以进行常规的密码失败检查、账户锁定和解锁操作。

这两个参数的默认值都符合安全标准，用户可以根据需要重新设置参数，提高安全等级。建议用户使用默认值。

配置 failed_login_attempts

1. 以安装 VexDB 的操作系统用户（vexdb为例）登录数据库主节点。
2. 使用如下命令连接数据库。

   ```
   vsql -d postgres
   ```

3. 查看已配置的参数。

   ```
   SHOW failed_login_attempts;
   ```

   返回结果如下：

   ```
   failed_login_attempts
   ----------------------
   5
   (1 row)
   ```

   如果显示结果不为 5，执行 `\q` 命令退出数据库。

4. 执行如下命令设置成默认值 5。

   ```
   vb_guc reload -D $PGDATA -c "failed_login_attempts=5"
   ```

配置 password_lock_time 参数

1. 以安装 VexDB 的操作系统用户（vexdb为例）登录数据库主节点。
2. 使用如下命令连接数据库。

```
vsql -d postgres
```

3. 查看已配置的参数。

```
SHOW password_lock_time;
```

返回结果如下所示：

```
password_lock_time
----------------------
    1440
(1 row)
```

如果显示结果不为1440，则修改为1440，执行。

4. 执行如下命令设置成默认值1440。

```
vb_guc reload -D $PGDATA -c "password_lock_time=1440"
```

手动锁定和解锁账户

若管理员发现某账户被盗、非法访问等异常情况，可手动锁定该账户。

当管理员认为账户恢复正常后，可手动解锁该账户。

以手动锁定和解锁用户 joe（joe 为已经存在且需要锁定和解锁的用户）为例：

1. 创建用户。

```
CREATE USER joe PASSWORD 'Vbase@123';
```

2. 手动锁定。

```
ALTER USER joe ACCOUNT LOCK;
```

3. 手动解锁。

```
ALTER USER joe ACCOUNT UNLOCK;
```

删除不再使用的账户

当确认账户不再使用，管理员可以删除账户。该操作不可恢复。

当删除的用户正处于活动状态时，此会话状态不会立马断开，用户在会话状态断开后才会被完全删除。

以删除账户joe为例，命令格式如下：

```
DROP USER joe CASCADE;
```

设置账号有效期

注意事项

- 创建新用户时，需要限制用户的操作期限（有效开始时间和有效结束时间）。
- 不在有效操作期内的用户需要重新设定账号的有效操作期。

操作步骤

1. 以操作系统用户 vexdb 登录数据库主节点。
2. 使用如下命令连接数据库。

   ```
   vsql -d postgres
   ```

3. 创建用户并制定用户的有效开始时间和有效结束时间。

   ```
   CREATE USER joe WITH PASSWORD 'Vbase@123' VALID BEGIN '2015-10-10 08:00:00' VALID UNTIL '2036-10-10 08:00:00';
   ```

   显示如下信息表示创建用户成功。

   ```
   CREATE ROLE
   ```

4. 用户已不在有效使用期内，需要重新设定账号的有效期，这包括有效开始时间和有效结束时间。

```
ALTER USER joe WITH VALID BEGIN '2016-11-10 08:00:00' VALID UNTIL '2017-11-10 08:00:00';
```

显示如下信息表示重新设定成功。

```
ALTER ROLE
```

> **说明**
>
> 在CREATE ROLE或ALTER ROLE语法中： - 若不指定VALID BEGIN，表示不对用户的开始操作时间做限定。 - 若不指定“VALID UNTIL”，表示不对用户的结束操作时间做限定。 - 若两者均不指定，表示该用户一直有效。

设置密码安全策略

加密存储密码

用户密码存储在系统表PG_AUTHID中，为防止用户密码泄露，VexDB 对用户密码进行加密存储，所采用的加密算法由配置参数password_encryption_type决定。

- 当参数设置为0时，表示采用md5方式对密码加密。MD5加密算法安全性低，存在安全风险，不建议使用。
- 当参数设置为1时，表示采用sha256和md5方式对密码加密。MD5加密算法安全性低，存在安全风险，不建议使用。
- 当参数设置为2时，表示采用sha256方式对密码加密，为默认配置。
- 当参数设置为3时，表示采用SM3方式对密码加密。

示例： 采用sha256和md5方式对密码加密。

1. 以安装 VexDB 的操作系统用户（以 vexdb 为例）登录数据库主节点。
2. 使用如下命令连接数据库，其中 vexdb 为需要连接的数据库名称，5432为数据库主节点的端口号。

```
vsql -d postgres
```

3. 查看已配置的加密算法。

```
SHOW password_encryption_type;
```

返回结果如下：

```
password_encryption_type
--------------------------
2
(1 row)
```

如果显示结果为 0 或 1，执行 `\q` 命令退出数据库。

4. 执行如下命令将其设置为安全的加密算法：采用 sha256 和 md5 方式对密码加密。

```
vb_guc reload -D $PGDATA  -c "password_encryption_type=2"
```

> **说明**
>
> 为防止用户密码泄露，在执行CREATE USER或者CREATE ROLE命令创建数据库用户时，不能指定UNENCRYPTED属性，即新创建的用户的密码只能是加密存储的。

密码复杂度

初始化数据库、创建用户、修改用户时需要指定密码，密码必须要符合复杂度的要求，否则会提示用户重新输入密码。

- 参数 password_policy控制了密码至少包含的有效字符类别数，其中有效字符包括大写字母（A-Z）、小写字母（a-z）、数字（0-9）、特殊字符（如表3.特殊字符所示）。取值及其对应的有效复杂度如下表所示：

  表1 password_policy 取值说明

参数取值	说明
0	关闭密码复杂度校验，但密码不能为空并且只包含有效字符。 说明 设置为 0 会存在安全风险，不建议设置为0，即使需要设置也要将所有 VexDB 节点中的 password_policy 都设置为 0 才能生效。
1	启用密码复杂度校验，密码至少需满足四类有效字符中的三类，为参数默认取值。
2	启用密码复杂度校验，且密码需满足四类有效字符中的四类。
3	启用密码复杂度校验，且密码至少需满足四类有效字符中的两类。

- 同时，其他参数与password_policy 协同控制密码的复杂度校验策略，包括如下参数：

  表2 密码复杂度协同参数

参数名称	说明
password_min_uppercase	密码包含的大写字母（A-Z）的最少个数。
password_min_lowercase	密码包含的小写字母（a-z）的最少个数。
password_min_digital	密码包含的数字（0-9）的最少个数。
password_min_special	密码包含的特殊字符的最少个数。特殊字符参见表3.特殊字符。
password_min_length	密码的最小长度。
password_max_length	密码的最大长度。

- 密码除了必须要符合参数password_policy 和表2.密码复杂度协同参数对复杂度的要求以外，还应满足如下要求：
  - 不能和用户名、用户名倒写相同，本要求为非大小写敏感。
  - 不能和当前密码、当前密码的倒写相同。
  - 不能是弱口令，弱口令指的是强度较低，容易被破解的密码，对于不同的用户或群体，弱口令的定义可能会有所区别，用户需自己添加定制化的弱口令。
  - 使用CREATE WEAK PASSWORD DICTIONARY创建弱口令，存放于GS_GLOBAL_CONFIG系统表中，弱口令默认为空。
  - 当创建用户、修改用户需要设置密码时，系统将会把用户设置的密码和弱口令字典中存放的口令进行对比，如果符合，则会提示用户该口令为弱口令，设置密码失败。

示例： 启用密码复杂度校验。

1. 以安装 VexDB 的操作系统用户（以 vexdb 为例）登录数据库主节点。
2. 使用如下命令连接数据库，其中 vexdb 为需要连接的数据库名称，5432 为数据库主节点的端口号。

```
vsql -d postgres
```

3. 查看已配置的参数。

```
SHOW password_policy;
```

返回结果如下：

```
password_policy
---------------------
1
(1 row)
```

如果显示结果不为 1，执行 `\q` 命令退出数据库。

4. 执行如下命令设置成默认值1。

```
vb_guc reload  -D $PGDATA -c "password_policy=1"
```

账户密码的复杂度要求如下：

- 包含大写字母（A-Z）的最少个数（password_min_uppercase）。
- 包含小写字母（a-z）的最少个数（password_min_lowercase）。
- 包含数字（0-9）的最少个数（password_min_digital）。
- 包含特殊字符的最少个数（password_min_special）（特殊字符的列表请参见表1. 特殊字符）。
- 密码的最小长度（password_min_length）。
- 密码的最大长度（password_max_length）。
- 至少包含上述四类字符中的三类。
- 不能和用户名、用户名倒写相同，本要求为非大小写敏感。
- 不能和当前密码、当前密码的倒写相同。
- 不能是弱口令。
  - 弱口令指的是强度较低，容易被破解的密码，对于不同的用户或群体，弱口令的定义可能会有所区别，用户需自己添加定制化的弱口令。
  - 弱口令字典中的口令存放在GS_GLOBAL_CONFIG系统表中，当创建用户、修改用户需要设置密码时，系统将会把用户设置口令和弱口令字典中存放的口令进行对比，如果符合，则会提示用户该口令为弱口令，设置密码失败。
  - 弱口令字典默认为空，用户通过以下语法可以对弱口令字典进行增加和删除，示例如下：

    ```
    CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1'), ('password2'); DROP WEAK PASSWORD DICTIONARY;
    ```

密码重用

用户修改密码时，只有超过不可重用天数（password_reuse_time）或不可重用次数（password_reuse_max）的密码才可以使用。参数配置说明如表2. 不可重用天数和不可重用次数参数说明所示。

示例： 配置 password_reuse_time 参数。

1. 以安装 VexDB 的操作系统用户（以 vexdb 为例）登录数据库主节点。
2. 使用如下命令连接数据库。

```
vsql -d postgres
```

3. 查看已配置的参数。

```
SHOW password_reuse_time;
```

返回结果为：

```
password_reuse_time
---------------------
90
(1 row)
```

如果显示结果不为 90，执行 `\q` 命令退出数据库。

4. 执行如下命令设置成默认值90。

```
vb_guc reload -D $PGDATA -c "password_reuse_time=90"
```

示例： 配置 password_reuse_max 参数

1. 以安装 VexDB 的操作系统用户（以 vexdb 为例）登录数据库主节点。
2. 使用如下命令连接数据库。

```
vsql -d vexdb -p 5432
```

3. 查看已配置的参数。

```
SHOW password_reuse_max;
```

返回结果为：

```
password_reuse_max
--------------------
3
(1 row)
```

如果显示结果不为 3，执行 `\q` 命令退出数据库。

4. 执行如下命令设置成默认值 3。

```
vb_guc reload -D $PGDATA -c "password_reuse_max = 3"
```

密码有效期限

数据库用户的密码都有密码有效期（password_effect_time），当达到密码到期提醒天数（password_notify_time）时，系统会在用户登录数据库时提示用户修改密码。（密码有效期到期后，只有普通用户会收到提示。）

示例： 配置 password_effect_time 参数。

1. 以安装 VexDB 的操作系统用户（以 vexdb 为例）登录数据库主节点。
2. 使用如下命令连接数据库，其中 vexdb 为需要连接的数据库名称，5432 为数据库主节点的端口号。

```
vsql -d vexdb -p 5432
```

3. 查看已配置的参数。

```
SHOW password_effect_time;
```

返回结果为：

```
password_effect_time
----------------------
90
(1 row)
```

如果显示结果不为90，执行 `\q` 命令退出数据库。

4. 执行如下命令设置成默认值90（不建议设置为0）。

```
vb_guc reload -D $PGDATA -c "password_effect_time = 90"
```

示例： 配置 password_notify_time 参数。

1. 以安装 VexDB 的操作系统用户（以 vexdb 为例）登录数据库主节点。
2. 使用如下命令连接数据库。

```
vsql-d postgres
```

3. 查看已配置的参数。

```
SHOW password_notify_time;
```

返回结果为：

```
password_notify_time
----------------------
7
(1 row)
```

如果显示结果不为 7，执行 命令退出数据库。

4. 执行如下命令设置成默认值 7（不建议设置为 0）。

```
vb_guc reload -D $PGDATA -c "password_notify_time = 7"
```

密码修改

在安装数据库时，会新建一个和初始化用户重名的操作系统用户，为了保证账户安全，请定期修改操作系统用户的密码。

以修改用户 user1 密码为例，命令格式如下：

```
passwd user1
```

建议系统管理员和普通用户都要定期修改自己的账户密码，避免账户密码被非法窃取。

示例： 修改用户 user1 的密码。

1. 以系统管理员用户连接数据库并执行如下命令：

```
ALTER USER user1 IDENTIFIED BY "Vbase@123" REPLACE "5678@def";
```

Vbase@123、5678@def 分别代表用户user1的新密码和原始密码，这些密码要符合密码复杂度规则，否则会执行失败。

2. 管理员可以修改自己的或者其他账户的密码。通过修改其他账户的密码，解决用户密码遗失所造成无法登录的问题。 以修改用户joe账户密码为例，命令格式如下：

```
ALTER USER joe IDENTIFIED BY "Vbase@123";
```

- 系统管理员可以修改普通用户密码且不需要用户原密码。
- 系统管理员修改自己密码但需要管理员原密码。
- 系统管理员之间不允许互相修改对方密码。

密码验证

设置当前会话的用户和角色时，需要验证密码。如果输入密码与用户的存储密码不一致，则会报错。

以设置用户 joe 为例，验证过程如下：

```
SET ROLE joe PASSWORD "Vbase@123";
```

返回结果为：

```
ERROR:  Invalid username/password,set role denied.
```

表3 特殊字符

编号	字符	编号	字符	编号	字符	编号	字符
1	~	9	*	17	|	25	<
2	!	10	(	18	[]	29	?
6	%	14	=	22	;	-	-
7	^	15	+	23	:	-	-
8	&	16	</td>	24	,	-	-

表2 不可重用天数和不可重用次数参数说明

参数	取值范围	配置说明
不可重用天数（password_reuse_time）	正数，其中整数部分表示天数，小数部分可以换算成时，分，秒。默认值为90。	如果参数变小，则后续修改密码按新的参数进行检查。如果参数变大（比如由a变大为b），因为b天之前的历史密码可能已经删除，所以b天之前的密码仍有可能被重用。则后续修改密码按新的参数进行检查。 说明： 时间以绝对时间为准，历史密码记录的都是当时的时间，不识别时间的修改。
不可重用次数（password_reuse_max）	1-100之间的正整数。默认值为3	如果参数变小，则后续修改密码按新的参数进行检查。如果参数变大（比如由a变大为b），因为b次之前的历史密码可能已经删除，所以b次之前的密码仍有可能被重用。则后续修改密码按新的参数进行检查。

设置用户密码失效

具有CREATEROLE权限的用户在创建用户时可以强制用户密码失效，新用户首次登陆数据库后需要修改密码才允许执行其他查询操作，命令格式如下：

```
CREATE USER joe PASSWORD "Vbase@123" EXPIRED;
```

具有CREATEROLE权限的用户可以强制用户密码失效或者强制修改密码且失效，命令格式如下：

```
ALTER USER joe PASSWORD EXPIRED;
```

或者：

```
ALTER USER joe PASSWORD "abc@2345" EXPIRED;
```

密码失效的用户登录数据库后，当执行简单查询或者扩展查询时，会提示用户修改密码。修改密码后可以正常执行语句。

只有初始用户、系统管理员（sysadmin）或拥有创建用户（CREATEROLE）权限的用户才可以设置用户密码失效，其中系统管理员也可以设置自己或其他系统管理员密码失效。不允许设置初始用户密码失效。

设置密码有效期

功能描述

参数password_effect_time用于设置帐户密码的有效时间。

设置方法

该参数属于SIGHUP类型参数，请参考重设参数中对应设置方法进行设置。

```
alter system set password_effect_time = n;
```

参数说明

n：账户密码的有效时间，临近或超过有效期系统会提示用户修改密码。

- 可通过GUC参数password_notify_time设置密码到期前多少天进行提醒。
- 可通过GUC参数user_password_warning_log控制是否将密码即将过期的提醒信息写入日志pg_log。如该参数为off，则仅在客户端登录时进行提醒。

示例

1. 查看当前账户密码的有效期。

```
SHOW password_effect_time;
```

返回结果如下：

```
password_effect_time
----------------------
36500
(1 row)
```

2. 更改密码有效期为100天并查询结果。

```
ALTER SYSTEM SET password_effect_time = 100;
SHOW password_effect_time;
```

返回结果如下：

```
password_effect_time
----------------------
100
(1 row)
```

3. 恢复环境。

```
ALTER SYSTEM SET password_effect_time = DEFAULT;
```

高可用集群

为了保证故障的可恢复，需要将数据写多份，设置主备多个副本，通过日志进行数据同步，以避免出现节点故障、停止后重启等情况的数据丢失。**数据库高可用集群**是一组计算机，它们作为一个整体向用户提供数据库服务，当主服务器出现故障时，可以将该服务器中的服务、资源、IP 等转移到另一台服务器上，从而满足业务的持续性。

VexDB 提供了基于HAS（High Availability Service）实现的高可用方案。

概述

HAS for VexDB 是一款功能丰富的集群资源管理软件，可用于高可用集群的管理。

特性介绍

- 支持使用配置文件自动化部署 VexDB 高可用集群。
- 支持集群内主备关系切换时自动切换 VIP 绑定关系。
- 支持集群内部署级联备机。
- 支持日志管理：

  HAS支持对$GAUSSLOG下的日志进行压缩和删除。
- HAS支持解耦化的安装和卸载。
- 支持一键式暂停/恢复HAS服务：

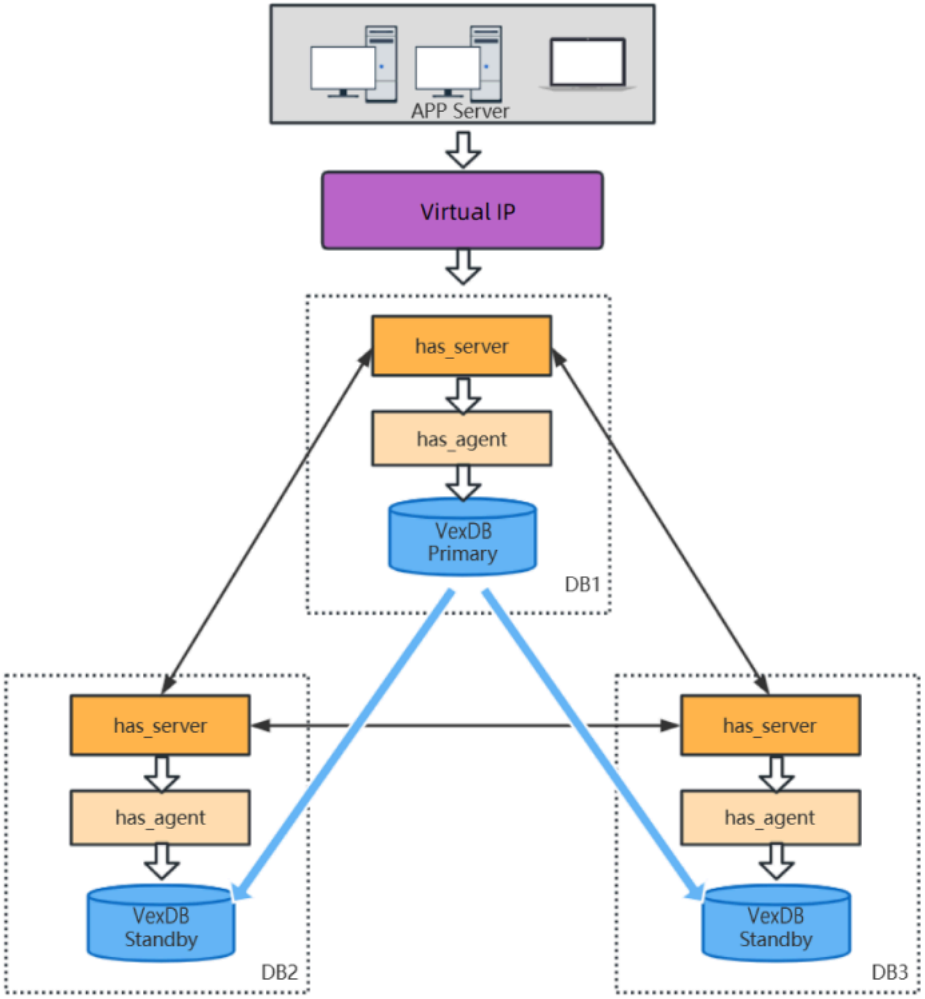
  支持一键暂停HAS自动故障处理服务，避免运维人员在运维过程中的操作受到HAS影响，运维完成之后可以一键恢复HAS服务。
- 支持丰富的集群内运维管理OM工具。
- 支持自定义资源监控：

  根据配置信息，支持用户自定义组件的监控和管理。
- 支持多 AZ 和跨 Region 的异地容灾部署模式。
- HAS涉及的节点间通讯端口（流复制、状态监控）和对外服务VIP 支持 IPV4/IPV6。

集群架构

一主两备是典型的数据库集群部署方式，其中一个备节点为异步数据复制，另外一个备节点为同步数据复制，其部署架构图如下图所示：

图1 一主两备部署架构图



组件	说明
has_server	用来进行数据库实例管理和实例仲裁的组件。主要功能有： 接收各个节点上 has_agent 发送的数据库各实例状态。 提供数据库实例整体状态的查询功能。 监控实例的状态变化并进行仲裁命令的下发。
has_agent	部署在数据库每个主机上，用来启停和监控各个数据库实例进程的数据库管理组件。主要功能有： 数据库实例启动和停止时负责拉起和停止本主机上部署的实例进程。 自动监测 VexDB 服务主备状态，并上报给 has_server。 执行 switchover、failover 等仲裁命令。 管理 VIP。
has_ctl	集群管理命令行客户端工具，提供集群管理操作。
has_monitor	has_agent 的守护进程，注册为系统内的定时任务。
VexDB	数据库服务，数据存储、读写，负责主备机间的数据同步。
VIP	对应用提供数据库连接服务的虚拟 IP。

部署

本章介绍了HAS V3 for VexDB 高可用集群的安装部署相关内容。

准备工作

本文介绍了搭建HAS V3 for VexDB高可用集群需要完成的准备工作，包括如下内容：

资源规划

本文以搭建一主两备高可用集群为例，则数据库机器需准备三台物理或虚拟主机，主机IP地址规划示例如下（IP地址以实际为准）：

系统类型	类型	主机名	IP地址
生产库集群	主库	VexDB1	172.16.0.1
	从库	VexDB2	172.16.0.2
	从库	VexDB3	172.16.0.3
	VIP	-	172.16.0.4

环境配置

在完成了系统和环境配置后，请关注一下配置：

- **编码格式配置**

  应在 utf8 场景下使用gs_install工具安装数据库，执行安装前应按照如下语句进行设置：

  > **说明**
  >
  > 安装完成后如需使用其他的编码格式，可再按需修改成对应的编码格式。

  ```
  echo "export LC_ALL="zh_CN.UTF-8"" >> /etc/profile
  echo "export LANG="zh_CN.UTF-8"" >> /etc/profile
  source /etc/profile
  ```

- **SSH服务端口配置（可选）**

  SSH互信使得在多个Linux服务器之间做操作时，可以免密登录，而不需要输入任何密码。

  通常情况下，HAS集群搭建过程会使用SSH协议的默认端口号（22），如需更改使用的SSH端口，可参考如下方式对集群内所有节点进行操作：

  1、配置SSH Server配置文件/etc/ssh/sshd_config。

  ```
  vi /etc/ssh/sshd_config
  Port 期望端口
  ```

  2、修改SSH Client配置文件/etc/ssh/ssh_config。

  > **说明**
  >
  > 应始终确保两个配置文件的期望端口一致；集群内各节点需要具有相同的SSH服务端口。

```
vi /etc/ssh/ssh_config
Port 期望端口
```

3、重启各节点的SSH服务，使修改生效。

```
systemctl restart sshd
```

- **创建数据库安装用户**

  执行预安装脚本时可以在指定数据库安装用户名和用户组后自动创建用户和组，所以本步骤可以选择跳过。

  1、以root用户登录操作系统。创建数据库安装用户/用户组（可自定义），此处以创建vexdb用户为例。

  - **方式一：** 创建用户vexdb以及同名用户组。（执行useradd命令创建用户时会同步创建同名用户组。）

    ```
    useradd -m vexdb
    ```

  - **方式二：** 使用groupadd命令创建用户组，使用useradd命令创建用户并指定其用户组。

    ```
    groupadd dbgrp
    useradd -m vexdb -g dbgrp
    ```

  2、设定初始密码（需要重复输入2次且完全一致）。

  ```
  passwd vexdb
  ```

安装部署

本章以安装一主两备数据库为例，介绍HAS V3 for VexDB高可用集群的部署流程，从而完成各搭建步骤。

> **说明**
>
> 文档中出现的节点名、IP和集群名以实际为准，图片仅供操作参考。

安装流程

1. 上传文件：根据准备安装集群的CPU和操作系统平台，下载对应的HAS集群管理工具的独立安装包和VexDB数据库安装包，将安装包上传至任意一个用于部署集群的节点上。
2. 解压安装包：以root用户执行tar命令解压搭建HAS V3 集群所需的压缩包。

若用户获取的是HAS集群管理软件的独立安装包，此步骤应分别解压HAS和VexDB数据库的安装包。

3. 配置XML文件：安装前需要先创建 cluster_config.xml 文件。可以参考后文给出的样例自行创建，或使用解压后的安装包中提供的 xml 模板进行修改。

   此xml文件包含部署VexDB集群的服务器信息、安装路径、IP地址以及端口号等配置信息。用户需要根据实际场景配置对应的 XML 文件。

4. 执行预安装脚本：以root用户执行gs_preinstall预安装脚本，为下一步安装准备好需要的环境和用户。
5. 执行安装脚本：切换到预安装时创建的集群安装用户，执行gs_install脚本安装数据库集群。
6. 按需配置许可文件：使用HAS独立安装包完成安装流程后，主、备数据库都将自动生成为期90天的临时License，同时也支持用户手动上传License文件并配置许可路径。
7. 进行VIP配置，通过回调脚本方式在主备切换场景下自动切换VIP。has_agent在本地数据库角色发生切换为主库时，可以通过回调脚本或配置参数的形式，绑定VIP地址；当本节点数据库角色切换为备机时，主动解除VIP地址绑定。

上传文件

上传安装HAS V3 for VexDB 集群所需的软件包至集群的任意一个节点上，此处以上传至/opt/路径下为例。所需的安装软件包如下：

名称格式为HAS_{版本号}_{平台信息}_{datetime}.tar.gz的包属于HAS集群管理工具的独立安装包。

解压安装包

用户可根据获取的安装包搭建HAS V3 的VexDB高可用集群：

解压VexDB安装包

1. 切换至安装包目录。

```
cd /opt
```

2. 使用root用户解压VexDB的安装包至/opt/software/路径下，得到VexDB-installer文件夹后再次解压文件夹内的压缩包（以下命令中的安装包名称以实际为准）。

```
mkdir /opt/software
tar -zxvf VexDB-installer-xxx-xx-xx.tar.gz -C /opt/software
tar -zxvf VexDB-installer/VexDB-2.2-Buildxxx-xxx-xx.tar.gz -C /opt/software
```

解压HAS独立安装包

1. 使用root用户对HAS安装包及其解压后得到的包进行解压，得到om和cm压缩包。（本步骤以centos7安装包为例）

```
tar -zxvf HAS_3@rxxx_centos7_x86-64_20230111.tar.gz
tar -zxvf VexDB-HAS-3-centos7.6_x86_64bit.tar.gz
```

2. 解压om压缩包，会在当前目录下生成script子目录。

```
tar -zxvf VexDB-HAS-3-xxx-om.tar.gz
```

配置XML文件

根据部署需求配置cluster_config.xml文件。可以参考下文给出的示例自行创建xml，也可以使用上一步解压安装包后得到的模板进行修改。

- 配置文件的模板位于解压目录下的/script/gspylib/etc/conf/cluster_config_template.xml，如按本文示例步骤完成解压，即为/opt/software/script/gspylib/etc/conf/cluster_config_template.xml
- xml配置文件中包含了VexDB集群的服务器信息、安装路径、IP地址以及端口号等部署信息。

1. 创建xml文件：

```
vi cluster_config.xml
```

2. 填写以下配置内容并保存。下文样例以一主两备部署方案为例。下文样例仅供参考，每部分内容都包含说明信息，详细参数说明详见配置说明一节，xml文件中各主机的名称与IP映射应配置正确。

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
    <!--整体信息 -->
    <CLUSTER>
        <PARAM name="clusterName" value="Cluster_template" />
        <PARAM name="nodeNames" value="VexDB1,VexDB2,VexDB3" />
        <PARAM name="gaussdbAppPath" value="/opt/VexDB/install/app" />
        <PARAM name="gaussdbLogPath" value="/var/log/omm" />
        <PARAM name="tmpMppdbPath" value="/opt/vexdb/tmp"/>
        <PARAM name="gaussdbToolPath" value="/opt/vexdb/install/om" />
        <PARAM name="corePath" value="/opt/vexdb/corefile"/>
        <PARAM name="backIp1s" value="172.16.105.54,172.16.105.58,172.16.105.60"/>

    </CLUSTER>
    <!-- 每台服务器上的节点部署信息 -->
    <DEVICELIST>
        <!-- node1上的节点部署信息 -->
        <DEVICE sn="VexDB1">
            <PARAM name="name" value="VexDB1"/>
            <PARAM name="azName" value="AZ1"/>
            <PARAM name="azPriority" value="1"/>
            <!-- 如果服务器只有一个网卡可用，将backIP1和sshIP1配置成同一个IP -->
            <PARAM name="backIp1" value="172.16.105.54"/>
            <PARAM name="sshIp1" value="172.16.105.54"/>
            <!--CM节点部署信息-->
            <PARAM name="cmsNum" value="1"/>
            <PARAM name="cmServerPortBase" value="15000"/>
            <PARAM name="cmServerListenIp1" value="172.16.105.54,172.16.105.58,172.16.105.60"/>
            <PARAM name="cmServerHaIp1" value="172.16.105.54,172.16.105.58,172.16.105.60"/>
            <PARAM name="cmServerlevel" value="1"/>
            <PARAM name="cmServerRelation" value="VexDB1,VexDB2,VexDB3"/>
            <PARAM name="cmDir" value="/opt/vexdb/data/cmserver"/>
            <!--dn-->
            <PARAM name="dataNum" value="1"/>
                        <PARAM name="dataPortBase" value="15400"/>
                        <PARAM name="dataNode1" value="/opt/vexdb/install/data/dn,vexdb2,/opt/vexdb/install/data/dn,vexdb3,/opt/vexdb/install/data/dn"/>
            <PARAM name="dataNode1_syncNum" value="1"/>
        </DEVICE>

        <!-- node2上的节点部署信息，其中“name”的值配置为主机名称 -->
        <DEVICE sn="VexDB2">
            <PARAM name="name" value="VexDB2"/>
            <PARAM name="azName" value="AZ1"/>
            <PARAM name="azPriority" value="1"/>
            <!-- 如果服务器只有一个网卡可用，将backIP1和sshIP1配置成同一个IP -->
            <PARAM name="backIp1" value="172.16.105.58"/>
            <PARAM name="sshIp1" value="172.16.105.58"/>
            <!-- cm -->
            <PARAM name="cmServerPortStandby" value="15000"/>
            <PARAM name="cmDir" value="/opt/VexDB/data/cmserver"/>
        </DEVICE>

        <!-- node3上的节点部署信息，其中“name”的值配置为主机名称 -->
        <DEVICE sn="VexDB3">
            <PARAM name="name" value="VexDB3"/>
            <PARAM name="azName" value="AZ1"/>
```

```
            <PARAM name="azPriority" value="1"/>
            <!-- 如果服务器只有一个网卡可用，将backIP1和sshIP1配置成同一个IP -->
            <PARAM name="backIp1" value="172.16.105.60"/>
            <PARAM name="sshIp1" value="172.16.105.60"/>
            <!-- cm -->
            <PARAM name="cmServerPortStandby" value="15000"/>
            <PARAM name="cmDir" value="/opt/vexdb/data/cmserver"/>
        </DEVICE>
    </DEVICELIST>
</ROOT>
```

执行预安装脚本

执行预安装脚本gs_preinstall可以协助自动完成如下的安装环境准备工作：

1. 自动设置Linux内核参数以达到提高服务器负载能力的目的。这些参数直接影响数据库系统的运行状态，请仅在确认必要时调整。
2. 自动将XML配置文件、安装包拷贝到其他主机的相同目录下，安装用户和用户组不存在时，自动创建安装用户以及用户组。
3. 读取XML配置文件中的目录信息并创建，将目录权限授予安装用户。

操作步骤

1. 用户需要检查上层目录权限，保证安装用户对安装包和配置文件目录读写执行的权限。
2. 以root用户进入script工具脚本目录。

```
cd /opt/software/script
```

3. 使用root用户执行执行gs_preinstall预安装脚本，在执行过程中，用户根据提示选择是否创建互信，并输入操作系统root用户和vexdb用户的密码。

```
./gs_preinstall -U vexdb -G dbgrp -X ../cluster_config.xml --sep-env-file=/home/vexdb/.vexdb
```

参数名	说明
-U	为数据库管理员（也是运行VexDB的操作系统用户），-U指定的操作系统用户名系统用户可以在预安装时自动指定创建。 **注意：** 建议安装数据库的操作系统用户名中包含的字母均使用小写。 否则在执行SQL时，指定含有大写字母的操作系统同名数据库初始化用户时，需要被双引号包裹才能被识别。
-G	指定操作系统用户的群组名称，可以在预安装时自动指定创建。。
-X	指定的是XML配置文件的路径。
-sep-env-file	指定的是环境变量文件。若不指定此配置项，则环境变量信息将写入.bashrc文件中，在部署多套集群时可能造成冲突。

4. 预安装过程示例如下：

```
Parsing the configuration file.
Successfully parsed the configuration file.
Installing the tools on the local node.
Successfully installed the tools on the local node.
Are you sure you want to create trust for root (yes/no)?yes
Please enter password for root
Password:
Successfully created SSH trust for the root permission user.
Setting host ip env
Successfully set host ip env.
Distributing package.
Begin to distribute package to tool path.
Successfully distribute package to tool path.
Begin to distribute package to package path.
Successfully distribute package to package path.
Successfully distributed package.
Are you sure you want to create the user[vexdb] and create trust for it (yes/no)? yes
Preparing SSH service.
Successfully prepared SSH service.
Installing the tools in the cluster.
Successfully installed the tools in the cluster.
Checking hostname mapping.
Successfully checked hostname mapping.
Creating SSH trust for [vexdb] user.
Please enter password for current user[vexdb].
Password:
Checking network information.
All nodes in the network are Normal.
Successfully checked network information.
Creating SSH trust.
Creating the local key file.
Successfully created the local key files.
Appending local ID to authorized_keys.
Successfully appended local ID to authorized_keys.
Updating the known_hosts file.
Successfully updated the known_hosts file.
Appending authorized_key on the remote node.
Successfully appended authorized_key on all remote node.
Checking common authentication file content.
Successfully checked common authentication content.
Distributing SSH trust file to all node.
Distributing trust keys file to all node successfully.
Successfully distributed SSH trust file to all node.
Verifying SSH trust on all hosts.
Successfully verified SSH trust on all hosts.
Successfully created SSH trust.
Successfully created SSH trust for [vexdb] user.
Checking OS software.
Successfully check os software.
Checking OS version.
Successfully checked OS version.
Creating cluster's path.
Successfully created cluster's path.
Set and check OS parameter.
Setting OS parameters.
Successfully set OS parameters.
```

```
Warning: Installation environment contains some warning messages.
Please get more details by "/opt/software/script/gs_checkos -i A -h VexDB1,VexDB2,VexDB3 --detail".
Set and check OS parameter completed.
Preparing CRON service.
Successfully prepared CRON service.
Setting user environmental variables.
Successfully set user environmental variables.
Setting the dynamic link library.
Successfully set the dynamic link library.
Setting Core file
Successfully set core path.
Setting pssh path
Successfully set pssh path.
Setting Cgroup.
Successfully set Cgroup.
Set ARM Optimization.
No need to set ARM Optimization.
Fixing server package owner.
Setting finish flag.
Successfully set finish flag.
Preinstallation succeeded.
```

执行安装脚本

1. 运行安装脚本。切换到vexdb用户（该用户为预安装脚本gs_preinstall中-U参数指定的用户），并执行source命令使环境变量生效。

```
su - vexdb
source ~/.VexDB
```

2. 执行gs_install安装脚本，进行集群部署安装，-X指定的是XML配置文件的路径，与预安装时指定的脚本一致。

```
gs_install -X /opt/software/cluster_config.xml --dn-guc="max_process_memory=10GB"
```

参数名	说明
-X	指定XML配置文件的路径。
-dn-guc	指定初始化数据库GUC参数取值。（可选）。 max_process_memory：保证数据库正常启动，max_process_memory参数必须配置为合适大小。 OM工具安装完成默认数据库内核至少需要7G内存；也可以通过修改其他内存相关GUC参数，以减小内核所需大小。</td>

3. 输入数据库密码，在脚本执行过程中，用户需根据提示输入数据库的密码，密码具有一定的复杂度，为保证用户正常使用该数据库，请记住输入的数据库密码。设置的密码要符合复杂度要求：
    - 最少包含8个字符。

- 不能和用户名、当前密码（ALTER）、或当前密码反序相同。
- 至少包含大写字母（A-Z）、小写字母（a-z）、数字、非字母数字字符（限定为~!@#$%^&*()-_=+|[{}];:,<.>/?）四类字符中的三类字符。

4. 安装过程如下所示，安装过程中会生成ssl证书，证书存放路径为${XML配置的程序安装目录}/share/sslcert/om。

```
Parsing the configuration file.
Check preinstall on every node.
Successfully checked preinstall on every node.
Creating the backup directory.
Successfully created the backup directory.
begin deploy..
Installing the cluster.
begin prepare Install Cluster..
Checking the installation environment on all nodes.
begin install Cluster..
Installing applications on all nodes.
Successfully installed APP.
begin init Instance..
encrypt ciper and rand files for database.
Please enter password for database:
Please repeat for database:
begin to create CA cert files
The sslcert will be generated in /opt/gaussdb/cluster/app/share/sslcert/om
Cluster installation is completed.
Configuring.
Deleting instances from all nodes.
Successfully deleted instances from all nodes.
Checking node configuration on all nodes.
Initializing instances on all nodes.
Updating instance configuration on all nodes.
Check consistence of memCheck and coresCheck on DN nodes.
Successful check consistence of memCheck and coresCheck on all nodes.
Configuring pg_hba on all nodes.
Configuration is completed.
Successfully started cluster.
Successfully installed application.
```

> **说明**
>
> root用户互信可能会存在安全隐患，因此建议用户在执行完安装后，立即删除各主机上root用户的互信：rm -rf ~/.ssh

5. 安装完成后，使用如下命令查询集群状态

```
has_ctl query -Cvidp
```

配置许可文件

使用HAS独立安装包完成安装流程后，主、备数据库都将自动生成为期90天的临时License，同时也支持用户手动上传License文件并配置许可路径。

如需使用正式的有效许可文件，可参考以下步骤为集群指定license文件。

步骤1： 上传许可文件至集群的每个节点，保证各节点中许可文件的绝对路径相同。

步骤2： 在任意节点上执行vb_guc命令，-N all表示为集群的所有节点设置许可。

```
vb_guc set -N all -I all -c "license_path = '/home/vexdb/license'"
```

- 上述命令中，license_path为许可文件的绝对路径，请根据实际情况替换引号内的路径。
- license_path在集群的每个节点中的路径应一致。

步骤3： 重启集群，使配置生效。在任意节点执行如下重启命令：

```
has_ctl stop && has_ctl start
```

VIP配置

VIP即对应用提供数据库连接服务的虚拟IP。为高可用集群配置VIP的作用是当主服务器发生故障无法对外提供服务时，动态将虚拟IP切换到备服务器，继续对外提供服务。

当前，HAS提供了两种方案来为集群配置VIP：

上述两个方案在应用效果上也存在差异，前者仅支持绑定单个VIP地址，而后者支持在集群中同时存在多个不同网段的VIP，包括IPV4和IPV6的VIP类型，但要求VIP类型和节点IP类型一致。

- IPV6仅支持通过回调脚本绑定VIP地址，IPV4则可以通过回调脚本绑定VIP地址或通过XML配置文件指定VIP地址（推荐方式）。
- 如果在虚拟机使用IPV6配置VIP，需要先申请可以ping通的IPV6地址，物理机上则可以直接配置需要的IPV6地址为IPV6。

通过回调脚本绑定VIP地址

集群管理支持通过回调脚本方式在主备切换场景下自动切换VIP的功能。has_agent在本地数据库角色发生切换为主库时，可以通过回调脚本并配置参数的形式，绑定VIP地址；当本节点数据库角色切换为备机时，主动解除VIP地址绑定。

创建VIP绑定回调脚本

使用数据库安装用户，在集群内所有节点的has_agent的配置文件目录下创建VIP绑定功能的回调shell脚本has_callback.sh文件。

步骤1 切换至数据库安装用户（本文以用户vexdb为例）。

步骤2 在has_agent目录下创建回调脚本has_callback.sh。

```
su - vexdb
cd /opt/vexdb/data/cmserver/cm_agent
vi has_callback.sh
```

脚本内容如下，注意根据实际情况调整要绑定的VIP网卡名称、广播地址、掩码等参数值：

```
#!/bin/bash
# ----------------------------------------------------------------------------
# Filename:    has_callback.sh
# Revision:    1.0
# Date:        2019/10/09
# Description:
# Notes:
#      callback 仅实现 vip 的 添加和移除
#
# ----------------------------------------------------------------------------
#
# ----------------------------------------------------------------------------
readonly cb_name=$1
readonly role=$2
readonly scope=$3


VIP=172.16.105.107  # vip 地址
#VIP=2001::e1:172:16:102:5
VIPBRD=172.16.107.255 # 广播地址
VIPNETMASKBIT=22   # 掩码
VIPDEV=eth0     # 网络接口名称,vip 会绑定到该接口
VIPLABEL=1   # 接口标签,默认为1

PING_TIMEOUT=2 # 设置为不高于vexdb.yml 配置文件里 loop_wait 的40%, 如果 loop_wait 设置为10, PING_TIMEOUT 建议设置为3, 如果 loop_wait 设置为5,
建议设置为2

function usage() {
    echo "Usage: $0 <on_start|on_stop|on_role_change> <role> <scope>";
    exit 1;
}

function addvip(){
    echo "`date +%Y-%m-%d\ %H:%M:%S,%3N` INFO: /sbin/ip addr add ${VIP}/${VIPNETMASKBIT} brd ${VIPBRD} dev ${VIPDEV} label ${VIPDEV}:${VIPLABEL}"
    sudo /sbin/ip addr add ${VIP}/${VIPNETMASKBIT} brd ${VIPBRD} dev ${VIPDEV} label ${VIPDEV}:${VIPLABEL}
    sudo /usr/sbin/arping -q -A -c 1 -I ${VIPDEV} ${VIP}
    #sudo /sbin/iptables -F

}

function delvip(){
    echo "`date +%Y-%m-%d\ %H:%M:%S,%3N` INFO: sudo /sbin/ip addr del ${VIP}/${VIPNETMASKBIT} dev ${VIPDEV} label ${VIPDEV}:${VIPLABEL}"
    sudo /sbin/ip addr del ${VIP}/${VIPNETMASKBIT} dev ${VIPDEV} label ${VIPDEV}:${VIPLABEL}
    #sudo /usr/sbin/arping -q -A -c 1 -I ${VIPDEV} ${VIP}
    #sudo /sbin/iptables -F

}

#if [[ $cb_name != 'on_master_check'  ]]; then
#    echo "`date +%Y-%m-%d\ %H:%M:%S,%3N` WARNING: has callback $cb_name $role $scope"
#fi

#if [[ $cb_name != 'on_master_check'  ]]; then
#    echo "`date +%Y-%m-%d\ %H:%M:%S,%3N` WARNING: has callback $cb_name $role $scope"
#fi
```

```
case $cb_name in
    on_stop)
        delvip
        ;;
    on_start)
        if [[ $role == 'master' ]]; then
            addvip
        fi
        ;;
    on_role_change)
        if [[ $role == 'master' ]]; then
            addvip
        elif [[ $role == 'slave' ]]||[[ $role == 'replica' ]]||[[ $role == 'logical' ]]; then
            delvip
        fi
        ;;
    on_master_check)
        if [[ $role == 'master' ]]; then
            vip_status=`ip addr|grep ${VIP}`
            if [[ ${vip_status} == '' ]]; then
                echo "`date +%Y-%m-%d\ %H:%M:%S,%3N` WARNING: has callback $cb_name $role $scope"
                echo "`date +%Y-%m-%d\ %H:%M:%S,%3N` WARNING: The vip was lost,bind vip because i am leader"
                /usr/bin/ping -I ${VIPDEV} -q -c 3 -W ${PING_TIMEOUT} ${VIP}
                is_vip_used=$?
                if [[ $is_vip_used -eq 0 ]]; then
                    echo "`date +%Y-%m-%d\ %H:%M:%S,%3N` WARNING: The vip bind failed, ${VIP} is in use"
                else
                    addvip
                fi
            fi
        fi
        ;;
    on_slave_check)
        if [[ $role == 'slave' ]]; then
            vip_status=`ip addr|grep ${VIP}`
            if [[ ${vip_status} != '' ]]; then
                delvip
            fi
        fi
        ;;
    *)
        usage
        ;;
esac
```

步骤3 给脚本文件赋执行权限，用于在本节点数据库主备角色发生变化时，自动由集群管理服务执行绑定和释放VIP的操作。

```
chmod u+x has_callback.sh
```

步骤4 切换至root用户。

步骤5 has_callback.sh中执行的命令需要管理员权限，因此需要设置sudo权限（使用root用户执行visudo命令），在最后边添加行，如(vexdb为用户名，可替换成实际的用户名，ip与arping命令路径也需要根据实际情况配置)。

1. 使用root用户执行visudo命令。

```
visudo
```

2. 在文件中找到root ALL=(ALL) ALL，在该行下方添加以下内容：

```
vexdb ALL=(ALL:ALL) NOPASSWD:/sbin/ip,/usr/sbin/arping,/sbin/iptables,/sbin/ifconfig
```

步骤6 赋予/usr/bin/sudo命令执行权限。

```
chmod u+s /usr/bin/sudo
```

步骤7 赋予各命令执行权限。

```
chmod +x /usr/sbin/ip
chmod +x /usr/sbin/arping
chmod +x /usr/sbin/iptables
chmod +x /usr/sbin/ifconfig
```

修改HAS相关配置文件

用数据库安装用户，修改集群中每个节点的has_agent配置文件cm_agent.conf中的callback_bin_path参数，将建的shell脚本路径/opt/vexdb/data/cmserver/cm_agent/has_callback.sh写入。

步骤1 切换至数据库安装目录，执行如下命令：

```
has_ctl set --param --agent -k callback_bin_path = "/opt/vexdb/data/cmserver/cm_agent/has_callback.sh"
```

修改完成 has_agent.conf 显示如下：

```
dilatation_shard_count_for_disk_capacity_alarm = 1             #check disk capacity,
mber
enable_dcf = off
callback_bin_path=/opt/vastbase/data/cmserver/cm_agent/has_callback.sh
disaster_recovery_type = 0
agent_backup_open = 0
############## End ##################
```

步骤2 修改alarmConfig.conf配置文件，也同样将has_callback.sh脚本路径写入参数callback_bin_path。

```
vi $GAUSSHOME/bin/alarmConfig.conf
```

修改内容：

```
callback_bin_path = /opt/vexdb/data/cmserver/cm_agent/has_callback.sh
```

修改完成显示如下：

```
#used for om_monitor
alarm_component = /opt/huawei/snas/bin/snas_cm_cmd
alarm_report_interval = 10
alarm_report_max_count = 1
#alarm scope should be json string:["A","B","C",""]
alarm_scope = ["newsql"]
callback_bin_path=/opt/vexdb/data/cmserver/cm_agent/has_callback.sh
```

重启集群VIP绑定功能生效

使用数据库安装用户vexdb用户重启集群使VIP功能的配置生效。

```
has_ctl stop
has_ctl start
```

查看IP，下图表示VIP已经成功绑定至172.16.105.54为例的节点上。

```
[redacted] cm_agent]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 52:54:00:98:39:93 brd ff:ff:ff:ff:ff:ff
    inet 172.16.105.54/22 brd 172.16.107.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet 172.16.105.107/22 brd 172.16.107.255 scope global secondary eth0:1
       valid_lft forever preferred_lft forever
    inet6 fe80::9976:b4ba:58a6:ea17/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

通过XML配置文件指定VIP地址

通过XML配置文件指定VIP时，支持多网段VIP能力。

根据本手册指导的HAS V3集群部署流程，部署前需要创建cluster_config.xml文件。此文件中包含部署HAS集群的各类配置信息，用于决定部署方式。若选用此方式进行VIP配置，需要用户关注XML文件中的以下参数：

配置项	说明
floatIp	虚拟IP地址。
dataListenIp	可选参数，各节点监听ip，若不配置，则默认使用backIp1s的值。
floatMap	可选参数，配置VIP时需要配置，表示floatIp与dataListenIp的对应关系，其顺序与dataListenIp1的顺序要相互对应。

关于XML配置文件的详细解释请参考配置说明。

配置说明

XML文件模板位于软件解压目录下的script/gspylib/etc/conf/cluster_config_template.xml。

在XML文件中，配置信息可以被分为如下模块：

- CLUSTER信息：配置数据库名称及各项目录。
- DEVICE LIST信息，此模块包含了CLUSTER信息中提及的各节点，各节点的信息通过标签来区分。
  - 数据库集群主节点，应该包含配置host基本信息、配置数据库主节点信息、配置HAS_SERVER（主、非主）信息。
  - 数据库集群备节点，应该包含配置host基本信息、配置HAS_SERVER（主、非主）信息。

> **说明**
>
> HAS_SERVER的主/非主状态与数据库节点的主备状态没有直接关联，用户可按照实际情况在各节点配置HAS_SERVER信息，保证在DEVICE LIST中，每个DEVICE上都配置了一个HAS_SERVER信息即可。

配置数据库名称及各项目录

> **说明**
>
> 以下配置内容仅为示例，用户可根据每行信息的注释说明自行替换。

配置项

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
    <!-- 整体信息 -->
    <CLUSTER>
    <!-- 数据库名称 -->
    <PARAM name="clusterName" value="Cluster_template" />
    <!-- 数据库节点名称(hostname) -->
    <PARAM name="nodeNames" value="VexDB1,VexDB2,VexDB3" />
    <!-- 数据库安装目录-->
    <PARAM name="gaussdbAppPath" value="/opt/VexDB/install/app" />
    <!-- 日志目录-->
    <PARAM name="gaussdbLogPath" value="/var/log/VexDB" />
    <!-- 临时文件目录-->
    <PARAM name="tmpMppdbPath" value="/opt/VexDB/tmp" />
    <!--数据库工具目录-->
    <PARAM name="gaussdbToolPath" value="/opt/VexDB/install/om" />
    <!--数据库 core 文件目录-->
    <PARAM name="corePath" value="/opt/VexDB/corefile"/>
    <!-- 节点 IP，与数据库节点名称列表一一对应 -->
    <PARAM name="backIp1s" value="172.16.105.54,172.16.105.58,172.16.105.60"/>
    </CLUSTER>
```

注意事项

- /opt/VexDB/install/om路径存放互信等工具，为了避免权限问题，不要把实例数据目录放在此目录下。
- 安装目录和数据目录需为空或者不存在，否则可能导致安装失败。
- 在对数据库节点的实例进行具体配置时，需确保配置的目录之间不相互耦合。即各个配置目录不关联，删除其中任意一个目录，不会级联删除其它目录。如当设置gaussdbAppPath为/opt/VexDB/install/app，gaussdbLogPath为/opt/VexDB/install/app/omm。若gaussdbAppPath目录被删除时，则会级联删除gaussdbLogPath目录，从而引起其它问题。
- 若需要预安装脚本自动创建安装用户时，各配置的目录需保证不与系统创建的默认用户目录耦合关联。
- 配置路径和实例路径时，路径中不能包含“|，;，&，$，<，>，`，，‘，“，{，}，(，)，[，]，~，*，?”特殊字符。
- 配置数据库节点名称时，请通过hostname命令获取各数据库节点的主机名称。

参数说明

整体信息	clusterName	VexDB名称。
nodeNames	VexDB中主机名称。	
backIp1s	主机在后端存储网络中的IP地址（内网IP）。所有VexDB主机使用后端存储网络通讯。	
gaussdbAppPath	VexDB程序安装目录。此目录应满足如下要求： • 磁盘空间>1GB。 • 与数据库所需其它路径相互独立，没有包含关系。	
gaussdbLogPath	VexDB运行日志和操作日志存储目录。此目录应满足如下要求： • 磁盘空间建议根据主机上的数据库节点数规划。数据库节点预留1GB空间的基础上，再适当预留冗余空间。 • 与VexDB所需其它路径相互独立，没有包含关系。 此路径可选。不指定的情况下，VexDB安装时会默认指定“$GAUSSLOG/安装用户名”作为日志目录。	
tmpMppdbPath	数据库临时文件存放目录。 若不配置tmpMppdbPath，默认存放在/opt/VexDB/wisequery/安装用户名_mppdb目录下，其中”opt/VexDB/wisequery”是默认指定的数据库系统工具目录。	
gaussdbToolPath	VexDB系统工具目录，主要用于存放互信工具等。此目录应满足如下要求： • 磁盘空间>100MB。 • 固定目录，与数据库所需其它目录相互独立，没有包含关系。 此目录可选。不指定的情况下，VexDB安装时会默认指定“/opt/VexDB/wisequery”作为数据库系统工具目录。	
corePath	VexDB core文件的指定目录。	

配置host基本信息

> **说明**
>
> 以下配置内容仅为示例，用户可根据每行信息的注释说明自行替换。

配置项

每台host服务器都必须有如下配置信息，示例以node1为例：

以下内容为示例，可自行替换。每行信息均有注释进行说明。

```
<!-- 每台服务器上的节点部署信息 -->
<DEVICELIST>
<!-- 节点1上的部署信息 -->
<DEVICE sn="VexDB1">
<!-- 节点1的主机名称 -->
<PARAM name="name" value="VexDB1"/>
<!-- 节点1所在的 AZ 及 AZ 优先级 -->
<PARAM name="azName" value="AZ1"/>
<PARAM name="azPriority" value="1"/>
<!-- 节点1的 IP，如果服务器只有一个网卡可用，将 backIP1和 sshIP1配置成同一个 IP -->
<PARAM name="backIp1" value="172.16.105.54"/>
<PARAM name="sshIp1" value="172.16.105.54"/>
<!-- node1是否为级联备，on 表示该实例为级联备，另外级联备机在相同的 AZ 里需要配有备机 -->
<PARAM name="cascadeRole" value="on"/>
```

参数说明

参数	说明
name	主机名称。
azName	指定azName（Available Zone Name），字符串（不能含有特殊字符），例如AZ1、AZ2、AZ3。
azPriority	指定azPriority的优先级。
backIp1	主机在后端存储网络中的IP地址（内网IP）。所有VexDB主机使用后端存储网络通讯。
sshIp1	设置SSH可信通道IP地址（外网IP）。 若无外网，则可以不设置该选项或者同backIp1设置相同IP。
cascadeRole	value为on表示该实例为级联备库。

> **说明**
>
> 配置文件中所有IP参数（包含backIp、sshIp、listenIp等）均只支持配置一个IP。如果配置第二个IP参数，则不会读取第二个参数的配置值。
>
> 示例：xml配置文件中同时配置backIp1和backIp2参数： 在解析配置文件时仅读取backIp1参数的配置值，不会读取backIp2参数的配置值。
>
> ```
> <PARAM name="backIp1" value="172.16.105.54"/>
> <PARAM name="backIp2" value="172.16.105.58"/>
> ```

配置数据库主节点信息

> **说明**

> 以下配置内容仅为示例，用户可根据每行信息的注释说明自行替换。

配置项

数据库主节点配置以下信息，内容为示例，可自行替换。每行信息均有注释进行说明。

```
<!--DBnode-->
<PARAM name="dataNum" value="1"/>
<!--DBnode 端口号-->
<PARAM name="dataPortBase" value="15400"/>
<!--DBnode 侦听 IP-->
 <PARAM name="dataListenIp1" value="172.16.105.54,172.16.105.58,172.16.105.60"/>
<!--DBnode 主节点上数据目录，及备机数据目录-->
<PARAM name="dataNode1" value="/opt/VexDB/install/data/dn,VexDB2,/opt/VexDB/install/data/dn,VexDB3,/opt/VexDB/install/data/dn"/>
<!--数据库主节点上的 xlog 目录，及备机 xlog 目录-->
<PARAM name="dataNodeXlogPath1" value="/home/omm/gauss_xlog,/home/omm/gauss_xlog "/>
<!--DBnode 节点上设定同步模式的节点数-->
<PARAM name="dataNode1_syncNum" value="0"/>
<!--数据库节点上自定义设定同步模式的节点方式、数量及顺序,hostname 根据主机名自行替换-->
<PARAM name="syncNode_hostname" value="ANY 1(node1_hostname, node2_hostname)"/>
<!--floatIp1、floatIp2为各节点或各 az 对应的 VIP 地址，用户需要保证 VIP 地址为可用 VIP，即需要可对外提供服务的 VIP（跟现有网卡处于同一网段中）dataListenIp1为各节点监听 ip，该选项可以不配置，若不配置，则默认使用 backIp1s 的值，floatI
<PARAM name="floatIpMap1" value="floatIp1,floatIp2,floatIp3"/>
```

参数说明

参数	说明
dataNum	当前主机上需要部署的数据库节点个数。
dataPortBase	数据库节点的基础端口号。
dataListenIp1	侦听的IP地址。未设置时，使用对应主机上的backIp1生成。 第一个IP是主节点所在主机IP，第二个IP是备节点所在主机IP。
dataNode1	用于指定当前主机上的数据库节点的数据存储目录。此目录为数据库的数据存储目录。应规划到数据盘上。
dataNodeXlogPath1	可选参数，用于指定当前数据库中xlog存储路径。此目录为数据库xlog日志存储目录，只支持绝对路径。 如不指定，则默认存放在数据目录的pg_xlog目录下。
dataNode1_syncNum	可选参数，与下面syncNode_hostname参数二选一。如需配置，仅在主机节点下配置。用于指定当前数据库中同步模式的节点数目。取值范围为0~数据库备机节点数。
syncNode_hostname	可选参数，与前面的dataNode1_syncNum参数二选一。用于一主四备集群时自定义利用FIRST、ANY设置同步备机，即指定当前数据库中同步模式的备机节点方式、数量及顺序。 如需配置，需要在所有的节点下同时配置。 syncNode_hostname中的hostname根据主机名自行替换。参数中指定同步备主机名存在且正确，同步备数量不能超过备选同步备主机个数。参数中FIRST与ANY不可以同时存在，ANY支持组合配置，FIRST不支持组合配置。
floatIpMap1	可选参数，表示floatIp与dataListenIp的对应关系，其顺序与dataListenIp1的顺序要相互对应。

配置HAS_SERVER（主、非主）信息

> **说明**
>
> 以下配置内容仅为示例，用户可根据每行信息的注释说明自行替换。

配置项

配置HAS_SERVER主信息

```
<!--HAS 节点部署信息-->
<PARAM name="cmsNum" value="1"/>
<PARAM name="cmServerPortBase" value="15000"/>
<PARAM name="cmServerListenIp1" value="172.16.105.54,172.16.105.58,172.16.105.60"/>
<PARAM name="cmServerHaIp1" value="172.16.105.54,172.16.105.58,172.16.105.60"/>
<!--cmServerlevel 目前只支持1-->
<PARAM name="cmServerlevel" value="1"/>
<!--cms 主及所有 hostname-->
<PARAM name="cmServerRelation" value="VexDB1,VexDB2,VexDB3"/>
<PARAM name="cmDir" value="/opt/VexDB/data/cmserver"/>
```

配置HAS_SERVER 非主信息

```
<!-- cm -->
<PARAM name="cmServerPortStandby" value="15000"/>
<PARAM name="cmDir" value="/opt/VexDB/data/cmserver"/>
```

参数说明

级联备库

为满足 VexDB 数据库一主多备集群更加灵活的部署需求，HAS V3 支持级联备在集群内的安装部署，实现典型的一主一备和一主多备+级联备的部署能力。

主机通过发送日志给备机实现同步，备机通过发送日志给级联备实现同步，其中主备之间日志同步可配置为同步或异步，备机与级联备之间只能以异步方式。

级联备库同样支持配置文件的自动化部署及状态的监控和查询，支持手动切换级联备机和备机，同时级联备机在故障后可自动拉起。

级联备集群的部署请参考安装部署章节。其中xml文件中在级联备库的节点信息中增加如下代码。

```
<PARAM name="cascadeRole" value="on"/>
```

> **说明**
>
> 级联备机只能与集群中的备机进行异步复制，不能与主机直接同步，部署级联备时必须要求有对应的备机。

例如：在部署级联备库时指定node3为级联备库，在xml脚本中node3的节点部署信息中添加如下内容即可部署级联备：

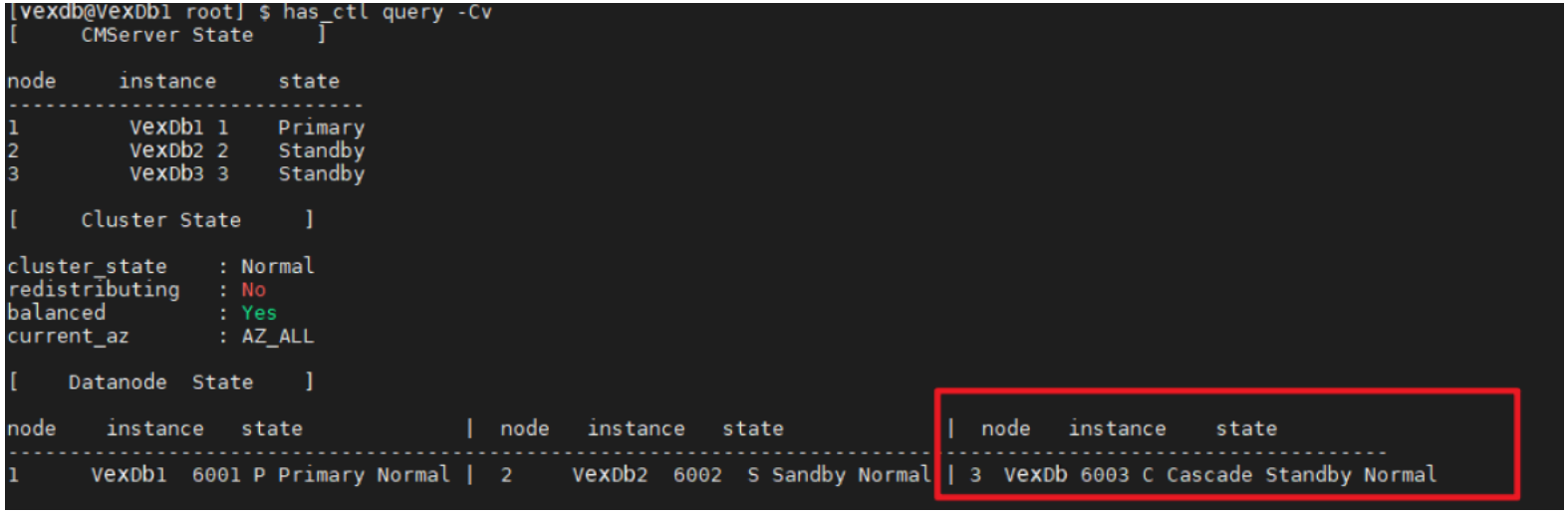
图1 指定级联备

```
<!-- node3上的节点部署信息，其中“name”的值配置为主机名称 -->
<DEVICE sn="node3_hostname">
    <PARAM name="name" value="node3_hostname"/>
    <PARAM name="azName" value="AZ1"/>
    <PARAM name="azPriority" value="1"/>
    <!-- 如果服务器只有一个网卡可用，将backIP1和sshIP1配置成同一个IP -->
    <PARAM name="backIp1" value="192.168.0.3"/>
    <PARAM name="sshIp1" value="192.168.0.3"/>
    <PARAM name="cascadeRole" value="on"/>
    <!-- cm -->
    <PARAM name="cmDir" value="/opt/vexdb/data/cmserver"/>"/>
    <PARAM name="cmServerPortStandby" value="15300"/>
</DEVICE>
```

使用如下语句查看集群状态时，显示其中节点3为级联备库。

```
has_ctl query -Cv
```

图2 查看集群状态



使用约束

- 级联备只能从备机同步数据，不能直接从主机同步。
- 级联备不支持直接从备机build数据，目前只能从主机build数据。如果备机需要全量build，会导致级联备也要重新全量build。
- 级联备为异步同步。
- 级联备不支持notify。
- 目前不提供查询“主 - 备 - 级联备”集群总体架构，需要根据主找到备，再通过备找到级联备。
- 级联备不能有再次级联备。

集群使用

本节介绍的命令均使用安装数据库的系统用户vexdb进行操作。

本节仅介绍集群的启停与其它基础使用命令，关于has_ctl的详细用法及参数说明请参考has_ctl。

集群启停

启动集群

使用数据库安装用户vexdb，在集群内任一服务器上执行以下命令启动集群。

```
has_ctl start
```

图1 启动集群

```
[vexdb@vexdb1 root]$ has_ctl start
has_ctl: checking cluster status.
has_ctl: checking cluster status.
has_ctl: checking cluster status.
has_ctl: start cluster.
has_ctl: start nodeid: 1
has_ctl: start nodeid: 2
has_ctl: start nodeid:3
.................
has_ctl: start cluster successfully.
```

停止集群

使用数据库安装用户vexdb，在集群内任一服务器上执行以下命令停止集群。

```
has_ctl stop
```

图2 停止集群

```
[vexdb@vexdb1 root]$ has_ctl stop
has_ctl: stop cluster.
has_ctl: stop nodeid: 1
has_ctl: stop nodeid: 2
has_ctl: stop nodeid:3
.................
has_ctl: stop cluster successfully.
```

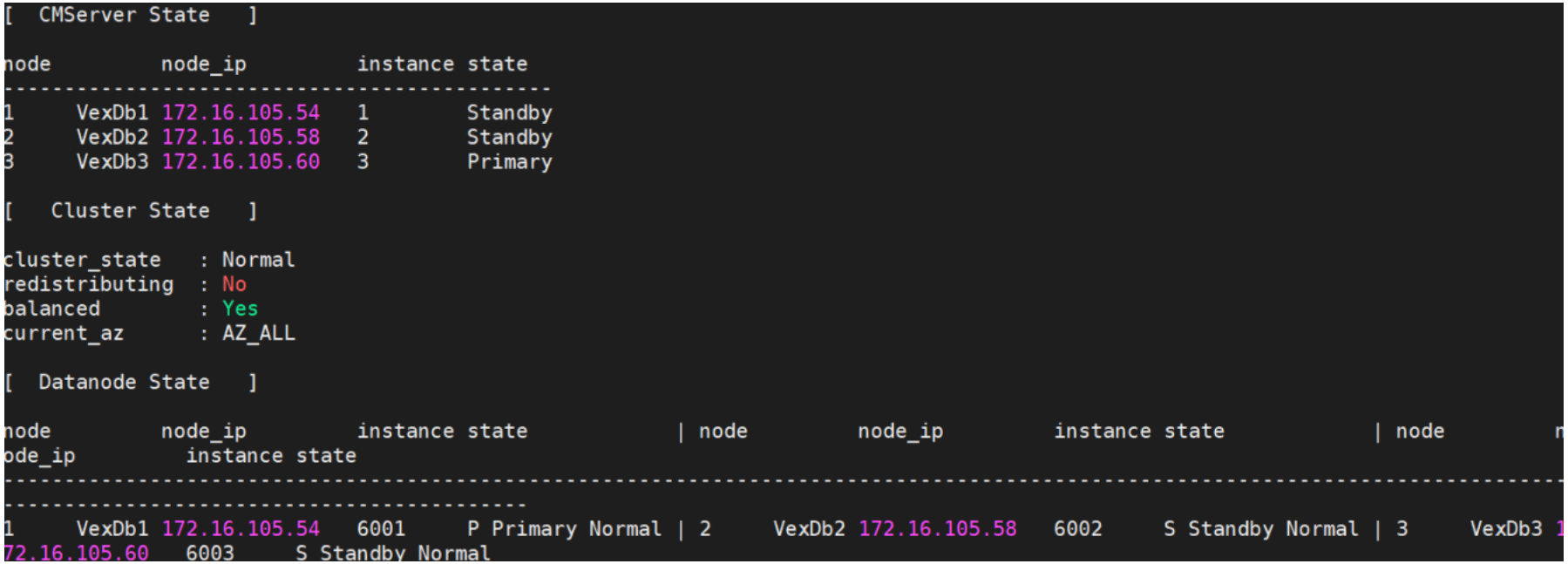
查看集群状态

使用数据库安装用户vexdb，在集群内任一服务器上执行以下命令查看集群状态：

```
has_ctl query -C -v -i
```

执行成功后会显示主备库各节点的信息，下图为一主两备的情况：

图3 查看集群状态



主备切换

使用数据库安装用户vexdb，在集群内任一服务器上执行以下命令，指定要切换为新主机的节点编号NODEID和数据目录DATA_DIR，命令如下：

```
has_ctl switchover -n NODEID -D DATA_DIR
```

例如：

```
has_ctl switchover -n 2 -D /opt/vexdb/install/data/dn
```

上述命令将节点编号为2的节点切换为主节点。执行成功后，使用如下语句查看切换后的集群状态信息：

```
has_ctl query -C -v -i
```

如下所示，node id为2的节点vexdb切换为新主：

图4 主备切换

```
node         node_ip         instance state
-------------------------------------------
1     VexDb1 172.16.105.54   1        Standby
2     VexDb2 172.16.105.58   2        Standby
3     VexDb3 172.16.105.60   3        Primary

[   Cluster State   ]

cluster_state   : Normal
redistributing  : No
balanced        : No
current_az      : AZ_ALL

[  Datanode State   ]

node         node_ip         instance state            node         node_ip         instance state            | node         node_ip         instance st
ate
-------------------------------------------------------- -------------------------------------------------------- ----------------------------------------
----------------
1     VexDb1 172.16.105.54   6001     P Standby Normal  2     VexDb2 172.16.105.58   6002     S Primary Normal | 3     VexDb3 172.16.105.60   6003     S
Standby Normal
```

switchover是维护操作，会变更集群内主机运行的节点，在确保数据库实例状态正常和业务结束情况下，无主备追赶后再进行switchover操作。

修改集群配置

可以使用vb_guc工具，修改VexDB数据库配置文件中的参数：

```
vb_guc set -N all [-D DATADIR] -c "parameter = value"
```

例如：

```
vb_guc set -Z datanode -N all -I all -c "max_process_memory=10GB"
```

故障切换

在主库机器故障时，集群管理组件会自动从备机中选出新主并提升为主库。

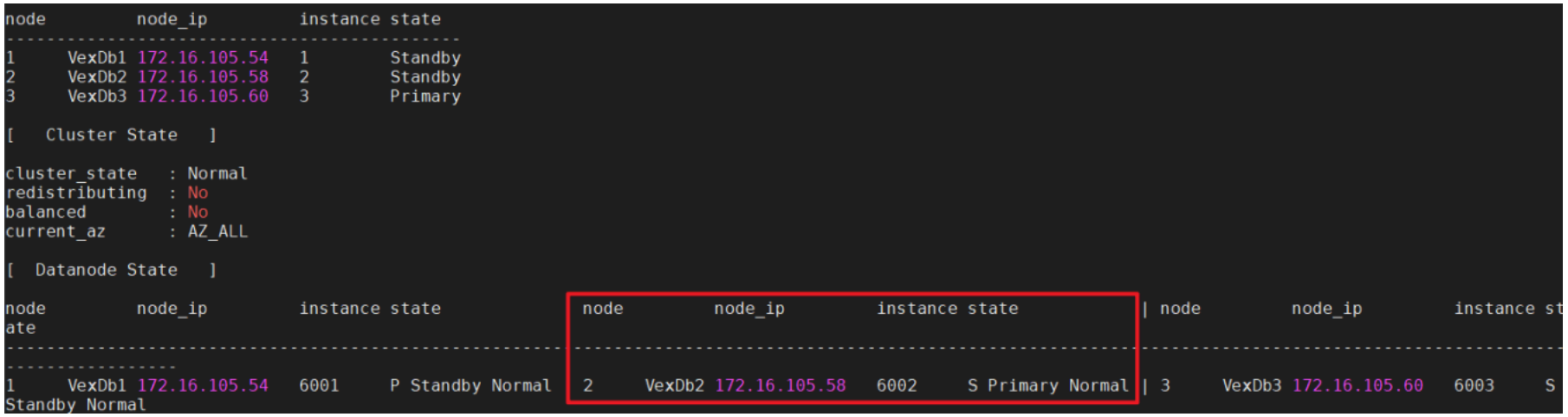
集群启动后，各节点部署运行的has_agent会持续监控并上报本节点数据库当前状态给has_server，has_server 通过心跳机制主动发现节点异常或通过has_agent主动上报，发现当前集群内主库故障，在当前运行的备机中选出新主机，下发failover命令给新主的has_agent,使备机自动升主，继续由主机提供数据库读写服务。

以下为使用has_ctl stop -n 主节点ID的方式停掉主库，模拟故障切换场景。

集群初始阶段，查看状态：

```
has_ctl query -C -v -i
```

图5 集群初始阶段，查看状态



模拟故障，停止主库：

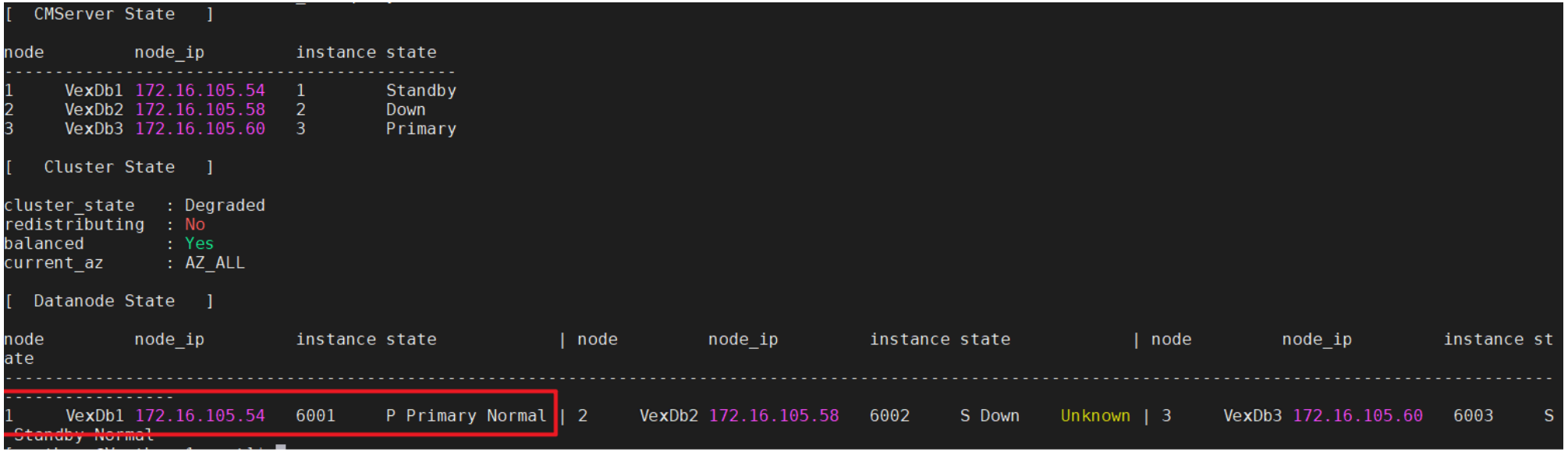
```
has_ctl stop -n 2
```

查看集群状态：

```
has_ctl query -C -v -i
```

返回结果如下，表示node 1节点的VexDB已经升为新主：

图6 模拟故障，停止主库



卸载集群

方法一：

以数据库安装用户登录主节点（本文以 vexdb 用户为例），使用 gs_uninstall 进行卸载。

```
su - vexdb
gs_uninstall --delete-data
```

方法二：

以数据库安装用户登录各个节点，使用 gs_uninstall 指定-L在每个节点执行本地卸载操作。

```
gs_uninstall --delete-data -L
```

清理集群

使用gs_uninstall进行集群卸载后，需要使用gs_postuninstall对环境进行清理。若未执行gs_postuninstall进行清理，再次安装HAS集群时会导致失败。

基于HAS Server分离部署的一主一备架构

功能描述

部署一主一备高可用集群时，支持通过在第三台服务器上额外部署一个 has_server，实现 HAS Server 的高可用能力。

每个节点上必有 om_monitor 和 has_agent，has_server 和 DN 是可选的。

部署方式

配置 XML 文件时，用于指定数据库节点数据存储目录的参数 dataNode1 应参考一主一备形式进行配置：

```
<PARAM name="dataNode1" value="/opt/vexdb/install/data/dn,node2_hostname,/opt/vexdb/install/data/dn"/>
```

XML配置文件中的其余信息应按照一主两备模板进行配置，完整部署流程可参考HAS V3 集群部署。

API 参考

本章介绍如何使用以下程序语言调用 VexDB 数据库。

- Java
- Python

Java

本节介绍通过驱动连接并通过 Java 程序操作 VexDB 的方式，并提供了示例代码以供参考。

Java 数据库连接（Java Database Connectivity，简称 JDBC）是 Java 语言中用来规范客户端程序如何来访问数据库的应用程序接口，提供了诸如查询和更新数据库中数据的方法。

vexdb-jdbc 数据库驱动程序是一个能够支持基本 SQL 功能的通用应用程序编程接口，支持一般的 SQL 数据库访问。通过驱动程序，用户可以在应用程序中实现对 VexDB 数据库的连接与访问。

配置 JDBC 驱动

在使用 JDBC 连接 VexDB 数据库前，需要配置 JDBC 驱动。

设置类路径

1. 通过 JDBC 进行连接之前，请先获取 JDBC 驱动包。 要使用驱动， 必须将驱动 jar 包含在类路径里，可以将jar包路径添加到CLASSPATH 环境变量中，或者使用 java 命令行标记的方式将驱动jar包引入。比如，有一个使用 VexDB JDBC 驱动的应用安装在 /usr/local/lib/myapp.jar，而 VexDB JDBC驱动安装在 /usr/local/vexdb/java/。可以用以下方式运行应用：

   ```
   export
   CLASSPATH=/usr/local/lib/MyApp.jar:/usr/local/vexdb/java/VexDB$shape_jdbc_$version<v|p>_$buildtime.jar.java MyApp
   ```

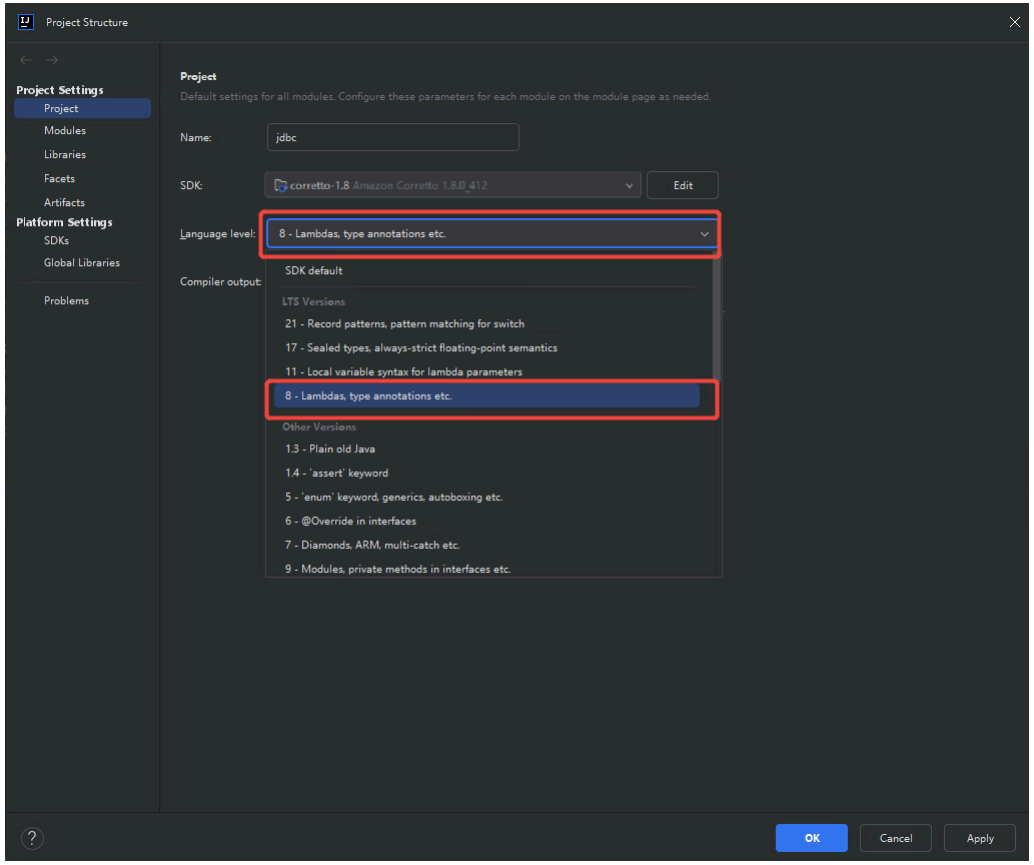
   不同版本的 jar 包名称可能不同，请以实际情况为准。

2. 在集成开发环境中配置 JDBC 驱动：

   以在 IDEAJ Community 中配置为例。

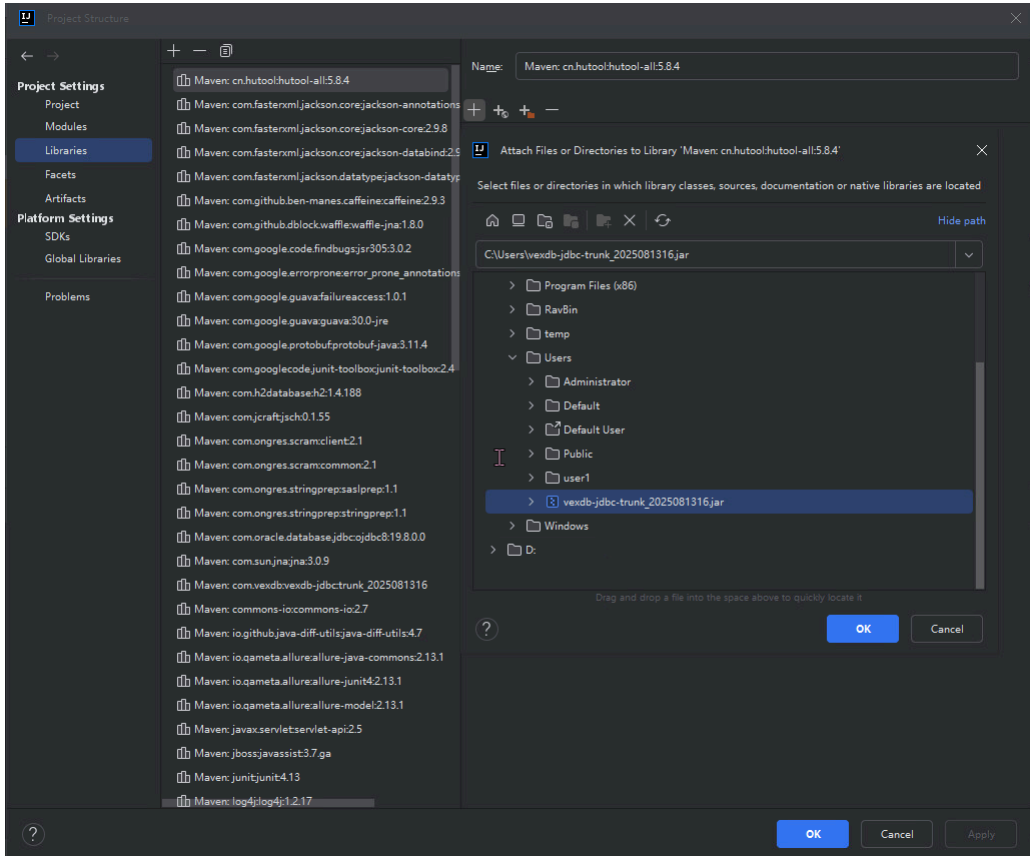
   1）配置 SDK。

   在 IDEAJ 中配置工程的 JDK 1.8，如下图所示。



2）导入 VexDB JDBC 包。

在 IDEAJ 中导入 jar 包，如下图所示。



配置为可接受 TCP/IP连接

由于 java 只支持使用 TCP/IP 连接，所以 VexDB 必须配置为可接受 TCP/IP 连接。可以通过修改 listen_addresses 来进行配置。

```
vi postgresql.conf
```

修改 listen_address 参数。

```
listen_address = '*'
```

配置认证文件

需要配置 VexDB 的主机认证文件，保证客户端程序可以访问数据库。

```
vi pg_hba.conf
```

添加以下内容：

```
host all all 0.0.0.0/0 md5
```

连接数据库

1. 导入 java.sql 包。

   任何使用 JDBC 的程序都需要导入java.sql包：

   ```
   import java.sql.*
   ```

2. 加载驱动。

   在试图与数据库建立连接之前，首先需要加载驱动，加载驱动有两种方法：

   在代码中，用Class.forName(Driver)方法加载驱动，例如：

   ```
   Class.forName("com.vexdb.Driver");
   ```

   在 JVM 启动时作为参数传递，例如：

   ```
   java -Djdbc.drivers=com.vexdb.Driver
   ```

3. 连接 URL 格式

   使用 VexDB JDBC驱动访问 VexDB 的URL格式如下：

   - jdbc:vexdb://host:port/database
   - jdbc:vexdb://host:port/
   - jdbc:vexdb://host/database
   - jdbc:vexdb://host/
   - jdbc:vexdb:/
   - jdbc:vexdb://host:port/database?param1=value1&param2=value2

   URL中各参数的含义如下：

- host

  服务端的主机名，默认为 localhost。如果要指定 IPV6 的地址，必须用方括号括起主机参数，例如：jdbc:vexdb://[::1]:5432/postgres。

- port

  服务端监听的端口，默认为 5432。

- database

  数据库名称，默认连接的数据库是与用户同名的数据库。比如连接用户为vbuser，如果不指定database参数，则默认连接到vbuser这个数据库。

- param1

  连接串参数，可参考连接参数中的表格。

- value

  连接串参数的取值，可参考连接参数中的表格，未指定时使用默认值。

4. 获取 JDBC 连接。

   使用 DriverManager.getConnection() 获取连接：

```
Connection connection = DriverManager.getConnection(url,username,password);
```

5. 关闭 JDBC 连接。

   关闭连接时调用Connection的close()方法即可：

```
Connection connection = DriverManager.getConnection(url,username,password);
connection.close();
```

连接参数

参数	说明
cmServerPortBase	主has Server端口。
cmServerPortStandby	备has Server端口号。
cmServerListenIp1	has Server用于侦听has Agent连接请求或DBA管理请求的IP地址。
cmServerHaIp1	主、备has Server间通信的IP地址。Value中左边为主has Server的主机IP地址，右边为备has Server的主机IP地址。 未设置时，默认根据主、备has Server所在主机的backIp1生成。

cmDir	has数据文件路径。保存has Server和has Agent用到的数据文件，参数文件等。

参数名称	参数类型	参数说明
PGDBNAME	String	指定用于连接的数据库名称（可直接在 JDBC URL 中指定），默认值为空。
PGHOST	String	指定数据库服务的主机名称（可直接在 JDBC URL 中指定），默认值为空。
PGPORT	String	指定数据库服务的端口号（可直接在 JDBC URL 中指定），默认值为空。
user	String	连接数据库的用户。
password	String	数据库用户的密码。
protocolVersion	String	连接协议版本号，目前仅支持 3。注意：设置该参数时将采用 md5 加密方式，需要同步修改数据库的加密方式：vb_guc set -N all -I all -c password_encryption_type=1 ，重启 VexDB 生效后需要创建用 md5 方录（不推荐）。默认值为空。
enable_ce	String	指定是否开启客户端加密特性，取值为 '1' 时用于打开此特性，默认值为空。
loggerLevel	String	指定驱动的日志级别，默认值为空，目前支持4种级别：OFF、INFO、DEBUG、TRACE。设置为 OFF 关闭日志，设置为 INFO、DEBUG 和 TRACE 记录的日志信息详细程度不同。默认值为 INFO。该参数设置值不
loggerFile	String	指定驱动的日志输出的文件名称，默认值为空。
logger	String	指定第三方程序使用的日志，默认为空。
prepareThreshold	Integer	指定使用 PreparedStatement() 构造的 SQL 语句在重复执行多少次后会缓存在数据库服务端，默认值为 5，即执行5次后会把查询计划等信息缓存到数据库端，随后的请求指挥发送该语句缓存后的句柄，建议使用默
preparedStatementCacheQueries	Integer	确定每个连接中缓存的查询数，默认情况下是 256。若在 prepareStatement() 调用中使用超过 256 个不同的查询，则最近最少使用的查询缓存将被丢弃。0 表示禁用缓存。建议使用默认值。
preparedStatementCacheSizeMiB	Integer	确定每个连接可缓存的最大值（以兆字节为单位），默认情况下是 5。若缓存了超过 5 MB 的查询，则最近最少使用的查询缓存将被丢弃。0表示禁用缓存。建议使用默认值。
databaseMetadataCacheFields	Integer	指定每个连接要缓存的最大字段数，当取值为 0 会禁用该缓存，默认值为 65536。
databaseMetadataCacheFieldsMiB	Integer	指定每个连接要缓存的字段的最大值（MB），当取值为 0 会禁用该缓存，默认值为 5。
defaultRowFetchSize	Integer	确定一次 fetch 在 ResultSet 中读取的行数。限制每次访问数据库时读取的行数可以避免不必要的内存消耗，从而避免 OutOfMemoryException。默认值是 0，这意味着 ResultSet 中将一次获取所有行，没有负数
binaryTransfer	Boolean	使用二进制格式发送和接收数据，默认值为 false。
readOnly	Boolean	将连接设置为只读模式。
binaryTransferEnable	String	指定以逗号分隔的二进制传输启用的类型列表，可以是 OID 编号或名称，默认值为空。
binaryTransferDisable	String	指定以逗号分隔的二进制传输禁用的类型列表，可以是 OID 编号或名称，设置本参数取值后可以覆盖驱动程序默认设置中的值和通过参数 binaryTransferEnable 设置的值，默认值为空。
stringtype	String	设置通过 setString() 方法使用的PreparedStatement() 参数的类型，可选字段为：false、 unspecified、 varchar。 如果 stringtype 设置为 VARCHAR（默认值），则这些参数将作为 varchar 参数发送给服务器。 当的类型。
unknownLength	Integer	默认为 Integereger.MAX_VALUE。某些 postgresql 类型（例如 TEXT）没有明确定义的长度，当通过 ResultSetMetaData.getColumnDisplaySize 和 ResultSetMetaData.getPrecision 等函数返回关于这些类型的

logUnclosedConnections	Boolean	客户端可能由于未调用 Connection 对象的 close() 方法而泄漏 Connection 对象。最终这些对象将被垃圾回收，并且调用 finalize() 方法。如果调用者自己忽略了此操作，该方法将关闭 Connection。
disableColumnSanitiser	Boolean	指定是否启用禁用列名净化的优化器，默认值为 false。
ssl	Boolean	以 SSL 方式连接。ssl=true 可支持 NonValidatingFactory 通道和使用证书的方式：1、NonValidatingFactory 通道需要配置用户名和密码，同时将 SSL 设置为 true。2、配置客户端证书、密钥、根证书，将 SSL
sslmode	Boolean	SSL 认证方式。取值范围为：disable、allow、prefer、require、verify-ca、verify-full。 disable：不使用 SSL 安全连接。allow：如果数据库服务器要求使用，则可以使用 SSL 安全加密连接，但不验证数据库服务性。require：只尝试 SSL 连接，如果存在 CA 文件，则应设置成 verify-ca 的方式验证。verify-ca：只尝试 SSL 连接，并且验证服务器是否具有由可信任的证书机构签发的证书。verify-full：只尝试 SSL 连接，并且致。
sslfactory	String	提供的值是 SSLSocketFactory 在建立 SSL 连接时用的类名。
sslfactoryarg	String	此值是上面提供的 sslfactory 类的构造函数的可选参数（不推荐使用）。
sslhostnameverifier	String	主机名验证程序的类名。
sslcert	String	提供证书文件的完整路径。客户端和服务端证书的类型为 End Entity。
sslkey	String	提供密钥文件的完整路径。使用时将客户端证书转换为 DER 格式：openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt
sslrootcert	String	SSL 根证书的文件名。根证书的类型为 CA。
sslpassword	String	sslpassword：String 类型。提供给 ConsoleCallbackHandler 使用。
sslprivatekeyfactory	String	客户端密钥 ssl 的 factory，默认值为空。
sslpasswordcallback	String	SSL 密码提供者的类名。
tcpKeepAlive	Boolean	启用或禁用 TCP 保活探测功能。默认为 false。
logInTimeout	Integer	指建立数据库连接的等待时间。超时时间单位为秒。
connectTimeout	Integer	使用的超时值或套接字连接操作，如果连接到服务器的时间超过该值，则连接断开。
socketTimeout	Integer 类	用于套接字读取操作的超时值。如果从服务器读取的时间超过此值，则连接将关闭。这既可以用作强力全局查询超时，也可以用作检测网络问题的方法。超时以秒为单位指定，值为零表示禁用。
cancelSignalTimeout	Integer	发送取消消息本身可能会阻塞，此属性控制用于取消命令的connectTimeout和socketTimeout。超时时间单位为秒，默认值为 10 秒。
socketFactory	String	用于创建与服务器 socket 连接的类的名称。该类必须实现了接口javax.net.SocketFactory，并定义无参或单 String 参数的构造函数。
socketFactoryArg	String	此值是上面提供的 socketFactory 类的构造函数的可选参数，不推荐使用。
sendBufferSize	Integer	该值用于设置连接流上的 SO_SNDBUF。
receiveBufferSize	Integer	在连接流上设置 SO_RCVBUF。
assumeMinServerVersion	String	客户端会发送请求进行 float 精度设置。该参数设置要连接的服务器版本，如 assumeMinServerVersion=9.0，可以在建立时减少相关包的发送。

ApplicationName	String	设置正在使用连接的 JDBC 驱动的名称。通过在数据库主节点上查询 pg_stat_activity 表可以看到正在连接的客户端信息，JDBC 驱动名称显示在 application_name 列。默认值为 PostgreSQL JDBC Driver。
ApplicationType	String	设置正在使用连接的 JDBC 驱动的类型。
jaasLogin	Boolean	用于启用/禁用在进行身份验证之前通过 JAAS 登录获取 Gss 凭据的标志。 如果设置系统属性 javax.security.auth.useSubjectCredsOnly=false，则有效 或使用系统属性为 sun.security.jgss.native-true 的本机 GS
jaasApplicationName	String	指定 JAAS 系统或应用程序登录配置的名称。
kerberosServerName	String	使用 GSSAPI 进行身份验证时要使用的 kerberos 服务名称。这相当于 libpq 的 PGKRBSRVNAME 环境变量。
useSpnego	Boolean	指定是否在 SSPI 身份验证请求中使用 spnego，默认值为 false，表示不使用。
gsslib	String	指定强制 SSPI 或 GSSAPI，取值可以为 auto、sspi、gssapi，默认值为 auto。
sspiServiceClass	String	指定用于 SPN 的 Windows SSPI 服务类，默认值为空。
allowEncodingChanges	Boolean	设置该参数值为“true”时进行字符集类型更改，配合 characterEncoding 设置字符集。
currentSchema	String	指定设置到 search-path 中的 schema。
targetServerType	String	指定要连接的服务器类型，取值可以为 any、master、slave、preferSlave，默认值为 any。 any：连接到任意一个可用的数据库节点。master：连接到任意一个可写的数据库节点。slave：连接到任意一个只读的 的数据库节点，如果没有可用的只读节点，则连接到可读可写节点。preferSecondary：连接到任意一个只读的数据库节点，如果没有可用的只读节点，则连接到可读可写节点。
priorityServers	Integer	此值用于指定 url 上配置的前 n 个节点作为主集群被优先连接。默认值为 null。该值为数字，大于 0，且小于 url 上配置的 DN 数量。
usingeip	Boolean	负载平衡时使用弹性 IP 地址，默认值为 true。
loadBalanceHosts	Boolean	配合 targetServerType 使用，若为 true，则随机分配节点，若为 false，则按顺序分配节。
hostRecheckSeconds	Integer	指定检查主机状态的周期，以防它们发生更改，单位是毫秒，默认值为 10。
forceTargetServerSlave	Boolean	此值用于控制是否开启强制连接备机功能，并在集群发生主备切换时，禁止已存在的连接在升主备机上继续使用。 默认值为 false，表示不开启强制连接备机功能。true 表示开启强制连接备机功能。
preferQueryMode	String	用于指定执行查询的模式，共有 4 种取值：“extended”、“extendedForPrepared”、“extendedCacheEverything”和”simple”。 simple 模式会 excute，不 parse 和 bind；extended 模式会 bind 和 exc 模式会缓存每个 statement。
autosave	String	如果查询失败，指定驱动程序应该执行的操作。共有 3 种取值：“always”、“never”、“conservative”。 在 autosave=always 模式下，JDBC 驱动程序在每次查询之前设置一个保存点，并在失败时回滚到 在 autosave=never 模式（默认）下，无保存点。在 autosave=conservative 模式下，每次查询都会设置保存点，但是只会在”statement XXX 无效”等情况下回滚并重试。
autoBalance	String	jdbc 可以通过 URL 中设置多个数据库节点，实现对主备集群的访问。通过设置负载均衡算法，jdbc 可以在建立连接时，依照特定规则将客户端与主备集群的连接依次建立在 URL 中配置的各个节点上，以此实现连接 接。连接主备集群时，使用此参数需要保证业务中没有写操作，或者与 targetServerType=slave 搭配，限制客户端只连接备机。目前，jdbc 提供了 roundrobin、priority roundrobin、leastconn、shuffle 四种负 • roundrobin：轮询模式，即与各候选节点轮流建立连接。取值：“roundrobin”、“true”、“balance”。 • priority roundrobin：带优先级的轮询模式，优先对前 n 个候选节点做轮询建连，取值：“proprity[n]”，n 为非负整数。 • leastconn：最小连接模式，对候选节点依照各节点的有效连接数做优先级排序，优先与连接数少的节点建立连接，该方法只会统计通过当前驱动在当前集群内使用 leastconn 模式建立的连接。取值：“leastconn” • shuffle：随机模式，即随机选择候选节点建立连接，取值：shuffle。loadBalanceHosts 设置为”true”等同于 autoBalance 设置为”shuffle”。 如果 loadBalanceHosts 设置为”true”的同时，autoBalance 设置为以上四种负载均衡模式，优先使 autoBalance 生效；如果同时配置了 autoBalance 和 targetServerType，jdbc 会在满足 targetServerType 的 jdbc 会将其视为同一集群上的连接，并整体进行负载均衡。

reWriteBatchedInserts	Boolean	批量导入时，该参数设置为 on，可将 N 条插入语句合并为一条： `insert Integero TABLE_NAME values(values1, …, valuesN), …, (values1, …, valuesN);` 使用该参数时，需设置 batchMode=off。
replication	String	指定启动报文的连接参数 replication 的值(true，database)。布尔值 true 告诉后端进入 walsender 模式，其中可以发出一小组复制命令而不是 SQL 语句。在 walsender 模式下只能使用简单查询协议。将数据库作行逻辑复制。（后端>=9.4）
TLSCiphersSupperted	String	用于设置支持的 TLS 加密套件，默认为 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE
xmlFactoryFactort	String	指定用于进行实例化 Factory 以进行 xml 处理的 Factory 类。
bulkloadCtlFile	String	指定 bulkload 控制文件的路径。
resultCaseMode	String	用于控制返回字段名的大小写。该参数会设置当前连接的 result_case_mode 参数。
timeTextMode	Boolean	指定是否使用 text 类型绑定时间参数，为了方式 JDBC 对时间的处理导致服务器收到的数据失真。默认值为 OFF。
exitCommit	Boolean	指定是否在 session 关闭时提交事务，默认值为 OFF，表示不在会话关闭时提交事务。本功能仅在内部功能 Debug 调试需要，对实际开发、生产使用无相关意义。且不推荐用户使用该参数。
extraFloatDigits	Integer	对 GUC 参数进行修改，为 session 级别，默认值为3，表示指定额外的浮动数字为3。
zzkk	String	内部测试表现行为调试参数，不用于实际业务中，即用户不可使用。
oraBlobMode	Boolean	针对 Blob 的新类型 oraBlob 进行相关逻辑改动。
userVexDBProductName	Boolean	指定是否使用 VexDB 作为productName 的返回。
quoteReturningIdentifiers	Boolean	指定是否引用返回列，默认值为 false，表示不引用。
loginWithHostname	Boolean	指定是否在系统视图 v$session 中查询中带有操作系统用户名称和主机名称。默认值为 false，表示不在系统视图中添加上述信息。
enableStatementLoadBalance	Boolean	是否开启语句级负载均衡，默认值为 false，表示不开启负载均衡。说明该参数仅在 JDBC V2.12 及以上版本支持。
writeDataSourceAddress	String	指定集群内写节点的 IP 地址与端口号。仅支持配置一个 IP:Port 作为写节点。

使用方式

基础用法

应用程序通过执行SQL语句来操作数据库的数据（不用传递参数的语句），需要按以下步骤执行：

1. 调用Connection的createStatement方法创建语句对象。

```
Connection conn = DriverManager.getConnection("url","user","password");
Statement stmt = conn.createStatement();
```

2. 调用Statement的executeUpdate方法执行SQL语句。

```
int rc = stmt.executeUpdate("DROP TABLE IF EXISTS fruit_tbl;"    +
"CREATE TABLE fruit_tbl(fruit_id INTEGER, "       +
"fruit_name VARCHAR(32),"  +
"fruit_period INTEGER,"    +
"fruit_email VARCHAR(64))");
```

- 数据库中收到的一次执行请求（不在事务块中），如果含有多条语句，将会被打包成一个事务，如果其中有一个语句失败，那么整个请求都将会被回滚。
- 事务块中不支持 vacuum 操作。
- 使用 Statement 执行多语句时应以 `;` 作为各语句间的分隔符。存储过程、函数、匿名块不支持多语句执行。
- `/` 可用作创建单个存储过程、函数、匿名块的结束符。

3. 关闭语句对象。

```
stmt.close();
```

VexDB-JDBC 主要类与接口

主要类或接口	包名	类或接口说明
Driver 类	com.vexdb.Driver	当注册驱动的时候或配置软件以使用 VexDB JDBC 驱动的时候，应该使用这个类名。
DriverManager 类	java.sql.DriverManager	跟踪可用的驱动程序，在数据库与相应的驱动程序之间建立连接。应用服务器使用 JDBC 时，DriverManager 类管理连接的建立。需要为 DriverManager 配置 JDBC 驱动，最简单的方法就是使用实现了接口 java.sql.Driver 的类的 Class.forName() 方法。在 VexDB JDBC 驱动中，该类的名称为 com.vexdb.Driver。该方法可以在连接一个数据库时，使用一个外部配置文件来给驱动提供类名和驱动参数。
Connection 接口	com.vexdb.PGConnection	与特定数据库的连接（会话）。在连接上下文中执行 SQL 语句并返回结果。
Statement 接口	com.vexdb.PGStatement	用于执行 SQL 语句并返回结果。
ResultSet 接口	com.vexdb.PGResultSetMetaData	存储执行 SQL 语句产生的结果集。

示例

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import com.vexdb.pgvector.PGvector;
public class pgVectorDemo {
static final String JDBC_DRIVER = "com.vexdb.Driver";
static final String DB_URL = "jdbc:vexdb://172.16.100.26:5432/postgres";
static final String USER = "vector_test";
static final String PASS = "Aa123456";
public static void main(String[] args) {
Connection conn = null;
try{
Class.forName(JDBC_DRIVER);
System.out.println("连接数据库...");
           conn = DriverManager.getConnection(DB_URL,USER,PASS);
System.out.println(" 实例化 Statement 对象...");
Statement setupStmt = conn.createStatement();
           setupStmt.executeUpdate("DROP TABLE IF EXISTS jdbc_items");
PGvector.addVectorType(conn);
/*
此处的 PGvector.addVectorType(conn)不可省略
如果缺少会导致 com.vexdb.util.PGobject cannot be cast to com.vexdb.pgvector.PGvector 的报错
*/
Statement createStmt = conn.createStatement();
       createStmt.executeUpdate("CREATE TABLE jdbc_items (id bigserial PRIMARY KEY, embedding floatvector(3))");
PreparedStatement insertStmt = conn.prepareStatement("INSERT INTO jdbc_items (embedding) VALUES (?), (?), (?), (?)");
       insertStmt.setObject(1, new PGvector(new float[] {1, 1, 1}));
       insertStmt.setObject(2, new PGvector(new float[] {2, 2, 2}));
       insertStmt.setObject(3, new PGvector(new float[] {1, 1, 2}));
       insertStmt.setObject(4, null);
       insertStmt.executeUpdate();
PreparedStatement neighborStmt = conn.prepareStatement("SELECT * FROM jdbc_items ORDER BY embedding <-> ? LIMIT 5");
       neighborStmt.setObject(1, new PGvector(new float[] {1, 1, 1}));
ResultSet rs = neighborStmt.executeQuery();
List<Long> ids = new ArrayList<>();
List<PGvector> embeddings = new ArrayList<>();
while (rs.next()) {
           ids.add(rs.getLong("id"));
           embeddings.add((PGvector) rs.getObject("embedding"));
}
System.out.print("embeddings: " +  embeddings.get(0));
System.out.print("embeddings: " +  embeddings.get(1));
System.out.print("embeddings: " +  embeddings.get(2));
Statement indexStmt = conn.createStatement();
       indexStmt.executeUpdate("CREATE INDEX ON jdbc_items USING ivfflat (embedding floatvector_l2_ops) WITH (ivf_nlist = 100)");
           setupStmt.close();
           createStmt.close();
           insertStmt.close();
           neighborStmt.close();
           indexStmt.close();
           conn.close();
}catch(SQLException se){
```

```
            se.printStackTrace();
}catch(Exception e){
            e.printStackTrace();
}finally{
try{
if(conn!=null) conn.close();
}catch(SQLException se){
                se.printStackTrace();
}
}
System.out.println("Goodbye!");
}
}
```

Python

VexDB 为使用 Python 语言进行数据库开发的用户提供了以下两个软件包。

- vexdb-psycopg2 基于 psycopg2 开发的数据库驱动包，它允许应用程序与数据库进行交互和通信，通过 Python 程序执行原始的 SQL 语句并返回结果。

  该包必须依赖 vexdb 的开发库进行编译（libpq），不能与 pyvector-vexdb 同时安装。

- pyvector-vexdb

  pyvector-vexdb 是一个向量扩展包。不能与 vexdb-psycopg2 同时安装。

  - 通过 psycopg2 驱动的 register_adapter 接口为 psycopg2 驱动扩展了向量类型 FloatVector。
  - 通过 SQLAlchemy ORM 的 TypeDecorator 接口为SQLAlchemy ORM扩展了向量类型 FloatVector。

vexdb-psycopg2

本节介绍通过 Python 驱动连接并操作 VexDB 的方式，并提供示例代码。

软件安装

1. 使用驱动软件前，请确保已经参考 安装 VexDB 的内容完成了数据库的安装。
2. 参考以下步骤配置环境变量：

```
# 应按实际的环境信息进行配置
export GAUSSHOME=/usr/local/vexdb
export PATH=$GAUSSHOME/bin:$PATH
export LD_LIBRARY_PATH=$GAUSSHOME/lib:$LD_LIBRARY_PATH
```

3. 安装 Python3 和 GCC。

```
yum install -y python3 python3-devel gcc
```

4. 使用root用户卸载已安装的 psycopg2（如有，否则跳过此步骤）

```
pip3 uninstall psycopg2 -y
```

5. 上传并解压 vexdb-psycopg2 源码包。

```
tar -xvf vexdb-psycopg2-{version}.tar.gz
```

6. 切换到解压后的目录，执行安装脚本。

```
cd vexdb-psycopg2-{version}
python3 setup.py build
# 此步骤可能需要 root 用户执行
python3 setup.py install
```

7. 检查安装结果。

   进入python3命令行，执行以下命令：

   ```
   import psycopg2
   ```

   导入成功即表示安装完成。

连接数据库

获取数据库连接

使用 psycopg2.connect() 获取 connection 对象，支持通过 dsn 与 key-value 两种格式连接数据库。

- DSN 格式：

  ```
  conn = psycopg2.connect("host=127.0.0.1 port=5432 dbname=postgres user=vbadmin password=vbase@123")
  ```

  对于DSN 格式，不支持用户密码中有 @ 等特殊字符。

- key-value 格式：

```
conn = psycopg2.connect(host="127.0.0.1",
                        port=5432,
                        dbname='postgres',
                        user='vbadmin',
                        password='vbase@123')
```

关闭数据库连接

关闭连接时调用 connect 的 close() 方法。

```
conn = psycopg2.connect(database="postgres", user="vbadmin", password="Vbase@123", host="127.0.0.1", port="5432")
conn.close()
```

配置数据库集群连接

psycopg2 支持在 DSN 中配置多个 Host:Port 对（Key-Value 格式则是配置多个 host），以逗号分隔。target_session_attrs 用于控制连接的数据库属性。

如下示例中，将 target_session_attrs 配置为 primary 表示仅允许连接主库，在发生主备切换后，psycopg2 会连接到切换后的新主库。

```
conn = psycopg2.connect(
                        host="xxx.xxx.xxx.xxa,xxx.xxx.xxx.xxb",
                        port=5432,
                        database='postgres',
                        user='vbadmin',
                        password='Vbase@123',
                        target_session_attrs='primary'
                        )
```

target_session_attrs 的可选取值包括：

- read-write

  只接受可以读写的数据库。

- any

  表示可以允许连接到任意数据库。

- read-only

  表示仅允许连接到只读的数据库。

- primary

  表示仅允许连接主库。

- standby

表示仅允许连接备库。

- prefer-standby

  表示连接到任意一个只读的数据库节点，如果没有可用的只读节点，则连接到可读可写节点。

示例

```
import string
import ctypes
import decimal
import platform
import unittest
import random
import testutils
import ast
import csv
import datetime
import io
from io import StringIO
import json
import struct
import numpy
from typing import Any
# 注意修改ConnectingTestCase中的数据库连接信息
from testutils import ConnectingTestCase, restore_types

import psycopg2
from psycopg2.extensions import FloatVector

class FloatVectorTest(ConnectingTestCase):

    def test_floatvector_dql(self):
        curs = self.conn.cursor()
        try:
            curs.execute("select '[1,2,3]'::floatvector(3);")
            result = curs.fetchone()[0]
            print(result.tolist())
            print(FloatVector([1,2,3]).tolist())
            print(str(FloatVector([1,2,3])))

            self.assertEqual(result.tolist(),FloatVector([1,2,3]).tolist())
            self.assertEqual(str(result),str(FloatVector([1,2,3])))

            curs.execute("select '[0]'::floatvector(1);")
            result = curs.fetchone()[0]
            print(str(result))
            print("------------------")
            self.assertEqual(result.tolist(),FloatVector([0]).tolist())
            self.assertEqual(str(result),str(FloatVector([0])))

            curs.execute("select '[1.00001,0.001234]'::floatvector(2);")
            result = curs.fetchone()[0]
            print(str(result))
            print("------------------")
            self.assertEqual(result.tolist(),FloatVector([1.00001,0.001234]).tolist())
            self.assertEqual(str(result),str(FloatVector([1.00001,0.001234])))

            curs.execute("SELECT ARRAY[1,1,1]::floatvector;")
            result = curs.fetchone()[0]
            print(str(result))
```

```
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1,1,1]).tolist())
        self.assertEqual(str(result),str(FloatVector([1,1,1])))

        curs.execute("SELECT '[1,1,1]'::floatvector;")
        result = curs.fetchone()[0]
        print(str(result))
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1,1,1]).tolist())
        self.assertEqual(str(result),str(FloatVector([1,1,1])))

        curs.execute("SELECT '[1.0]'floatvector AS vector_example;")
        result = curs.fetchone()[0]
        print(str(result))
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1.0]).tolist())
        self.assertEqual(str(result),str(FloatVector([1.0])))

        curs.execute("SELECT '[1.0]'::char(5)::floatvector AS vector_example;")
        result = curs.fetchone()[0]
        print(str(result))
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1.0]).tolist())
        self.assertEqual(str(result),str(FloatVector([1.0])))

        curs.execute("SELECT '[1.0]'::nchar(5)::floatvector AS vector_example;")
        result = curs.fetchone()[0]
        print(str(result))
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1.0]).tolist())
        self.assertEqual(str(result),str(FloatVector([1.0])))

        curs.execute("SELECT '[1.0]'::varchar(5)::floatvector AS vector_example;")
        result = curs.fetchone()[0]
        print(str(result))
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1.0]).tolist())
        self.assertEqual(str(result),str(FloatVector([1.0])))

        curs.execute("SELECT '[1.0]'::nvarchar(5)::floatvector AS vector_example;")
        result = curs.fetchone()[0]
        print(str(result))
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1.0]).tolist())
        self.assertEqual(str(result),str(FloatVector([1.0])))

        curs.execute("SELECT '[1.0]'floatvector AS vector_example;")
        result = curs.fetchone()[0]
        print(str(result))
        print("------------------")
        self.assertEqual(result.tolist(),FloatVector([1.0]).tolist())
        self.assertEqual(str(result),str(FloatVector([1.0])))

        curs.execute("SELECT '[1.0]'::bpchar(5)::floatvector AS vector_example;")
        result = curs.fetchone()[0]
        print(str(result))
```

```
            print("------------------")
            self.assertEqual(result.tolist(),FloatVector([1.0]).tolist())
            self.assertEqual(str(result),str(FloatVector([1.0])))

            print("query result: ",result)
            self.conn.commit()
        except Exception:
            self.conn.rollback()
            raise
        finally:
            curs.close()
```

接口参考

vexdb-psycopg2 为使用 Python 语言的应用程序定义了一组访问和操作 VexDB 的标准接口。

psycopg2.connect()

此方法用于创建新的数据库会话并返回新的connection对象。

返回值：connection对象（连接数据库实例的对象）。

参数说明如下：

关键字	参数说明
dbname	数据库名称。
user	用户名。
password	密码。
host	数据库所在主机IP地址。
port	连接端口号，默认为5432。
sslmode	ssl模式，ssl连接时用。
sslcert	客户端证书路径，ssl连接时用。
sslkey	客户端秘钥路径，ssl连接时用。
sslrootcert	根证书路径，ssl连接时用。
target_session_attrs	配置多个 host 时，数据库连接的优先级。该参数会使用目标库的 libpq 中的target_session_attrs。 • read-write 在连接的时候，只接受可以读写的数据库。建立连接后，会发送 SHOW transaction_read_only，如果是o n，代表是只读库，psycopg2 会把连接关闭；然后测试第二个数据库，以此类推，直至连接到支持读写的数据库为止。 • any 表示可以允许连接到任意数据库，它会从所有配置的连接中随机选择一个，如果连接的数据库出现故障导致连接断开，会尝试连接其他数据库，从而实现故障转移。 • read-only 表示仅允许连接到只读的数据库。 • primary 表示仅允许连接主库。 • standby 表示仅允许连接备库。 • prefer-standby 表示连接到任意一个只读的数据库节点，如果没有可用的只读节点，则连接到可读可写节点。

connection.cursor()

此方法用于返回新的 cursor 对象。参数说明如下：

关键字	参数说明
name	cursor 名称，默认为 None
cursor_factory	用于创造非标准 cursor，默认为 None
scrollable	设置 SCROLL 选项，默认为 None
withhold	设置 HOLD 选项，默认为 False

cursor.execute()

此方法执行被参数化的SQL语句（即占位符，而不是SQL文字）。psycopg2模块支持用%s标志的占位符。参数说明如下：

关键字	参数说明
query	待执行的 sql 语句
vars_list	变量列表，匹配 query 中%s 为占位符

connection.commit()

此方法将当前挂起的事务提交到数据库。默认情况下，psycopg2在执行第一个命令之前打开一个事务：如果不调用commit()，任何数据操作的效果都将丢失。

参数：无。

返回值：无。

connection.rollback()

此方法回滚当前挂起事务。

参数：无。

返回值：无

cursor.fetchone()

此方法提取查询结果集的下一行，并返回一个元组。

参数：无。

返回值：单个元组，为结果集的第一条结果，当没有更多数据可用时，返回为“None”。

cursor.fetchall()

此方法获取查询结果的所有（剩余）行，并将它们作为元组列表返回。

参数：无。

返回值：元组列表，为结果集的所有结果。空行时则返回空列表。

cursor.close()

此方法用于关闭当前连接的游标。

参数：无。

返回值：无。

connection.close()

此方法用于关闭数据库连接。

参数：无。

返回值：无。

pyvector-vexdb

开源库 psycopg2 允许 Python 程序与 PostgreSQL 数据库进行交互，VexDB 提供了基于 psycopg2 开发的 SDK： pyvector-vexdb。pyvector-vexdb 在 psycopg2 的基础上进行修改，兼容 SQLAlchemy，Django ORM，Peewee 等 ORM 框架。

软件安装

> **说明**
>
> 操作前请确保已经参考 安装VexDB 完成了数据库安装，并部署了 python3 环境。 依赖 psycopg2，pyvector-vexdb 对操作系统和 CPU 没有要求，python 版本需要 ≥3.8。

将 pyvector-vexdb 安装到本地环境中，可参考如下命令：

```
pip install psycopg2-binary #必须
pip install sqlalchemy #如果要使用SQLAlchemy集成
pip install numpy #可选，支持直接映射numpy到向量类型
pip install pyvector-vexdb.xxx.whl
```

它提供了一个功能齐全的 API 来执行 SQL 语句，可以处理事务、获取查询结果、执行批量数据插入等操作。

psycopg2 模式

原生 psycopg2 并不支持向量类型，pyvector-vexdb 通过 psycopg2 的 register_adapter 机制，可以为 psycopg2 新增向量类型 floatvector 的支持。

1. 和数据库建立连接。

```
from vexdb.psycopg2 import register_vector
conn = psycopg2.connect("dbname=postgres,user=vexdb,password=123456,host=172.0.0.1,port=5432")
```

2. 将向量类型注册到您的连接或游标。

```
#将向量类型注册到当前连接
register_vector(conn)
#或将向量类型注册到游标
#cur = conn.cursor()
#register_vector(cur)
```

3. 创建一个向量表。

```
cur.execute('CREATE TABLE items (id bigserial PRIMARY KEY, embedding floatvector(3))')
```

4. 插入一个向量（向量嵌入-embedding）

```
embedding = np.array([1, 2, 3])
cur.execute('INSERT INTO items (embedding) VALUES (%s)', (embedding,))
```

5. 对向量进行函数/操作符操作：

   - 获取一个向量最近邻

     ```
     cur.execute('SELECT * FROM items ORDER BY embedding <-> %s LIMIT 5', (embedding,))
     cur.fetchall()
     ```

   - 计算两个向量的余弦距离

     余弦相似度 = 1-余弦距离，比较两个向量的方向而不是它们的大小。余弦相似度的范围在-1到1之间，1表示向量相同，0表示无关，-1表示向量指向相反方向。

     ```
     cur.execute('SELECT 1-(%s <->%s) as value', (embedding1,embedding2))
     cur.fetchall()
     ```

   - 计算两个向量的负内积。内积 = -1*负内积，内积表示两个向量在相同维度上的对应分量相乘后再求和的结果，内积越大表示方向越一致。

```
cur.execute(' -1 * (%s <#> %s) AS value',(embedding1,embedding2))
cur.fetchall()
```

- 两个向量逐元素相加。

```
cur.execute('SELECT %s + %s AS value',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_combine(double precision[], double precision[])

  将两个double precision类型的数组逐元素相加成一个新的向量，返回这个新向量。

```
cur.execute('SELECT floatvector_combine(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_accum(double precision[], floatvector)

  将一个向量累加到数组中，返回一个新的数组。其中，被累加的数组及结果数组的第一个元素为累加次数，之后的元素为各维度的累积值。

```
cur.execute('SELECT floatvector_accum(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_cmp(floatvector, floatvector)

  函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量小于第二个，返回 -1；如果相等，返回 0；如果第一个向量大于第二个，返回 1。

```
cur.execute('SELECT floatvector_cmp(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_gt(floatvector, floatvector)

  函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量大于第二个向量，则返回true，否则返回false。

```
cur.execute('SELECT floatvector_gt(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_ge(floatvector, floatvector)

  函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量大于等于第二个向量，则返回true，否则返回false。

```
cur.execute('SELECT floatvector_ge(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_ne(floatvector, floatvector)

  函数用于比较两个向量，如果两个向量的任意元素不相等，则返回true，否则返回false。

```
cur.execute('SELECT floatvector_ne(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_eq(floatvector, floatvector)

  这个函数用于比较两个向量，如果两个向量的所有元素都相等，则返回true，否则返回false。

```
cur.execute('SELECT floatvector_eq(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_le(floatvector, floatvector)

  这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量小于等于第二个向量，则返回true，否则返回false。

```
cur.execute('SELECT floatvector_le(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_lt(floatvector, floatvector)

  这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量小于于第二个向量，则返回true，否则返回false。

```
cur.execute('SELECT floatvector_lt(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_spherical_distance(floatvector, floatvector)

  这个函数用于计算两个向量之间的球面距离（spherical distance），即两个向量之间的夹角的余弦值。

```
cur.execute('SELECT floatvector_spherical_distance(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_negative_inner_product(floatvector, floatvector)

  这个函数用于计算两个向量的负内积（negative inner product），即两个向量对应位置上的元素相乘后求和并取负值。

```
cur.execute('SELECT floatvector_negative_inner_product(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_l2_squared_distance(floatvector, floatvector)

这个函数用于计算两个向量之间的L2范数的平方距离。L2范数距离是向量元素差的平方和。

```
cur.execute('SELECT floatvector_l2_squared_distance(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_avg(double precision[])

  这个函数用于计算一个double precision类型数组中所有元素的值进行N等分。N是数组的第一个元素，并返回一个N等分后的向量。

```
cur.execute('SELECT floatvector_avg(%s)',(embedding))
cur.fetchall()
```

- floatvector_sub(floatvector, floatvector)

  这个函数用于计算两个向量的元素级相减，返回一个新的向量，其中每个元素是对应位置上两个向量元素的差。

```
cur.execute('SELECT floatvector_sub(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_add(floatvector, floatvector)

  这个函数用于计算两个向量的元素级相加，返回一个新的向量，其中每个元素是对应位置上两个向量元素的和。

```
cur.execute('SELECT floatvector_add(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- floatvector_norm(floatvector)

  这个函数用于计算给定向量的范数，即向量元素的平方和的平方根。

```
cur.execute('SELECT floatvector_norm(%s)',(embedding))
cur.fetchall()
```

- floatvector_dims(floatvector)

  这个函数用于返回给定向量的维度（即向量中元素的数量）。

```
cur.execute('SELECT floatvector_dims(%s)',(embedding))
cur.fetchall()
```

- l2_distance(floatvector, floatvector)

  这个函数用于计算两个向量之间的L2范数距离。L2范数距离也称为欧氏距离，表示两个向量之间的直线距离。

```
cur.execute('SELECT l2_distance(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- inner_product(floatvector, floatvector)

  这个函数用于计算两个向量的内积。内积是两个向量对应元素乘积的和。

```
cur.execute('SELECT inner_product(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

- cosine_distance(floatvector, floatvector)

  这个函数用于计算两个向量之间的余弦距离。余弦距离是通过计算两个向量之间的夹角余弦值来衡量它们之间的相似度。

```
cur.execute('SELECT cosine_distance(%s,%s)',(embedding1,embedding2))
cur.fetchall()
```

6. 添加一个近似索引。

```
cur.execute('CREATE INDEX ON items USING hnsw (embedding floatvector_l2_ops)')
# or
cur.execute('CREATE INDEX ON items USING ivfflat (embedding floatvector_l2_ops) WITH (lists = 100)')
```

> **说明**
>
> 更多索引类型、参数和操作符使用详见《向量检索指南》。

示例

```
import numpy as np
from vexdb.psycopg2 import register_vector
import psycopg2
from psycopg2.extras import DictCursor, RealDictCursor, NamedTupleCursor
from psycopg2.pool import ThreadedConnectionPool

# 使用dsn
conn = psycopg2.connect("dbname=postgres user=test password=Test@1234 host=localhost port=5432")
# 使用关键字
# conn = psycopg2.connect(
#     dbname="postgres",
#     user="test",
#     password="Test@1234",
#     host="localhost",
#     port=5432)
conn.autocommit = True

cur = conn.cursor()
cur.execute('DROP TABLE IF EXISTS psycopg2_items')
cur.execute('CREATE TABLE psycopg2_items (id bigserial PRIMARY KEY, embedding floatvector(3))')

# 注册到连接
register_vector(conn)
# 注册到游标
# register_vector(cur)

class TestPsycopg2:
    def setup_method(self, test_method):
        cur.execute('DELETE FROM psycopg2_items')

    def test_vector(self):
        embedding = np.array([1.5, 2, 3])
        cur.execute('INSERT INTO psycopg2_items (embedding) VALUES (%s), (NULL)', (embedding,))

        cur.execute('SELECT embedding FROM psycopg2_items ORDER BY id')
        res = cur.fetchall()
        assert np.array_equal(res[0][0], embedding)
        assert res[0][0].dtype == np.float32
        assert res[1][0] is None

    def test_query(self):
        pass

    def test_cursor_factory(self):
        for cursor_factory in [DictCursor, RealDictCursor, NamedTupleCursor]:
            conn = psycopg2.connect(host='localhost', port=5432, database='postgres', user='test', password='Test@1234')
            cur = conn.cursor(cursor_factory=cursor_factory)
            register_vector(cur, globally=False)
            conn.close()

    def test_cursor_factory_connection(self):
        for cursor_factory in [DictCursor, RealDictCursor, NamedTupleCursor]:
            conn = psycopg2.connect(host='localhost', port=5432, database='postgres', user='test', password='Test@1234', cursor_factory=cursor_factory)
            register_vector(conn, globally=False)
            conn.close()
```

```
    def test_pool(self):
        pool = ThreadedConnectionPool(1, 1, host='localhost', port=5432, database='postgres', user='test', password='Test@1234')

        conn = pool.getconn()
        try:
            # use globally=True for apps to ensure registered with all connections
            register_vector(conn, globally=False)
        finally:
            pool.putconn(conn)

        conn = pool.getconn()
        try:
            cur = conn.cursor()
            cur.execute("SELECT '[1,2,3]'::floatvector")
            res = cur.fetchone()
            assert np.array_equal(res[0], np.array([1, 2, 3]))
        finally:
            pool.putconn(conn)

        pool.closeall()
```

SQLAlchemy 兼容模式

pyvector-vexdb 提供了 SQLAlchemy 集成，可以直接映射到 VexDB 的向量类型和向量函数。这允许您直接使用向量类型 FloatVector 定义 SQLAlchemy 模型，并使用熟悉的 SQLAlchemy 语法执行向量相似性搜索。

使用方式

1. 映射向量列。

```
from vexdb.sqlalchemy import FloatVector

class Item(Base):
    embedding = mapped_column(FloatVector(3))
```

2. 插入向量。

```
item = Item(embedding=[1, 2, 3])
session.add(item)
session.commit()
```

3. 对向量进行函数/操作符操作：

   - 查询近似向量：使用欧几里得距离操作符l2_distance(<->)

```
session.scalar(select(Item).order_by(Item.embedding.l2_distance([3, 1, 2])).limit(5))
```

除此以外，还支持如下操作符，具体使用详见函数和操作符。

- negative_inner_product(<#>)
- consine_distance(<=>)
- add(+)
- sub(-)

- l2_distance函数，查询向量的欧几里得距离：

```
session.scalar(select(Item.embedding.l2_distance([3, 1, 2])))
```

或者查询近似向量（使用欧几里得距离）

```
session.scalar(select(Item).filter(Item.embedding.l2_distance([3, 1, 2]) < 5))
```

- floatvector_combine函数，将两个double precision类型的数组逐元素相加成一个新的向量，返回这个新向量。

```
session.scalar(session.query(floatvector_combine([1.0,2.0,3.0], [4,5,6])))
```

- floatvector_accum函数，用于将一个向量累加到数组中，返回一个新的数组。其中，被累加的数组及结果数组的第一个元素为累加次数，之后的元素为各维度的累积值。

```
session.scalar(session.query(floatvector_accum([1.0, 2.0, 3.0], [4, 5])))
```

- floatvector_cmp函数，这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量小于第二个，返回 -1；如果相等，返回 0；如果第一个向量大于第二个，返回 1。

```
session.scalar(session.query(floatvector_cmp([1,1,1,1],[2,2,2,2])))
```

- floatvector_gt，这个函数逐个比较两个向量的元素，找到第一个不同的元素并根据它决定大小关系。如果第一个向量大于第二个向量，则返回true，否则返回false。

```
session.scalar(session.query(floatvector_gt([1,1,1,1],[2,2,2,2])))
```

- floatvector_spherical_distance函数，函数用于计算两个向量之间的球面距离，即两个向量之间的夹角的余弦值。

```
session.scalar(session.query(floatvector_spherical_distance([1,1,1,1],[2,2,2,2])))
```

除此以外，还支持如下函数，具体使用详见函数和操作符。

- floatvector_ge
- floatvector_ne
- floatvector_eq
- floatvector_le
- floatvector_lt
- floatvector_negative_inner_product
- floatvector_l2_squared_distance
- floatvector_avg
- floatvector_sub
- floatvector_add
- floatvector_norm
- floatvector_dims
- l2_distance
- inner_product
- cosine_distance

4. 近似最近邻索引。

```
index = Index(
    'my_index',
    Item.embedding,
    postgresql_using='hnsw',
    postgresql_with={'m': 16, 'ef_construction': 64},
    postgresql_ops={'embedding': 'floatvector_l2_ops'}
)
# or
index = Index(
    'my_index',
    Item.embedding,
    postgresql_using='ivfflat',
    postgresql_with={'ivf_nlist': 100},
    postgresql_ops={'embedding': 'floatvector_l2_ops'}
)
```

> **说明**
>
> 更多索引类型、参数和操作符使用详见《向量检索指南》。

示例1：典型向标联合查询

以下程序展示了使用 SQLAlchemy 语法，构建在手机商城中执行向量相似性搜索的示例。

```
from sqlalchemy import create_engine, Column, Integer, Text, func
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from sqlalchemy.sql import text
from vexdb.sqlalchemy import FloatVector
import numpy as np
from typing import List, Optional, Tuple
import logging

# 配置日志
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# 数据库配置
#DATABASE_URL = "postgresql+psycopg2://test:Test%401234@172.16.100.97:5432/postgres"
DATABASE_URL = "postgresql+psycopg2://postgres:12345Aa*@172.16.97.114:5432/postgres"

# 创建基类
Base = declarative_base()

class Product(Base):
    """手机产品模型"""
    __tablename__ = 'products'

    id = Column(Integer, primary_key=True)
    name = Column(Text, nullable=False)
    color = Column(Text, nullable=False)  # 颜色枚举值：'black', 'white', 'red', 'blue', 'gold'
    description = Column(Text, nullable=False)
    features = Column(FloatVector(512), nullable=False)  # 512维特征向量

    def __repr__(self):
        return f"<Product(id={self.id}, name='{self.name}', color='{self.color}')>"


class ProductVectorSearch:
    """手机产品向量检索服务"""

    def __init__(self, database_url: str):
        self.engine = create_engine(database_url)
        self.SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=self.engine)

        # 创建表
        self._create_tables_and_extensions()

    def _create_tables_and_extensions(self):
        """创建表"""
        # 清空表，便于重新创建
        Base.metadata.drop_all(bind=self.engine)
        # 创建所有表
        Base.metadata.create_all(bind=self.engine)

        # 创建向量索引以提升检索性能
        with self.engine.connect() as conn:
            try:
                conn.execute(text("""
                    CREATE INDEX IF NOT EXISTS products_features_idx
```

```
                    ON products USING ivfflat (features floatvector_cosine_ops)
                    WITH (ivf_nlist = 100)
                """))
                conn.commit()
                logger.info("向量索引创建成功")
            except Exception as e:
                logger.warning(f"创建向量索引失败: {e}")

    def add_product(self, id: int, name: str, color: str, description: str, features: np.ndarray) -> Product:
        """添加新产品"""
        with self.SessionLocal() as session:
            product = Product(
                id=id,
                name=name,
                color=color,
                description=description,
                features=features.tolist()  # 转换为列表格式
            )
            session.add(product)
            session.commit()
            session.refresh(product)
            return product

    def vector_search_with_color_filter(
        self,
        query_vector: np.ndarray,
        color_filter: Optional[str] = None,
        colors_filter: Optional[List[str]] = None,
        limit: int = 10,
        similarity_threshold: float = 0.5
    ) -> List[Tuple[Product, float]]:
        """
        使用向量检索结合颜色过滤的联合查询

        Args:
            query_vector: 查询向量 (512维)
            color_filter: 单个颜色过滤条件
            colors_filter: 多个颜色过滤条件列表
            limit: 返回结果数量限制
            similarity_threshold: 相似度阈值 (0-1)

        Returns:
            List[Tuple[Product, float]]: 产品和相似度分数的元组列表
        """
        with self.SessionLocal() as session:
            # 构建基础查询，计算余弦相似度
            query = session.query(
                Product,
                (1 - Product.features.cosine_distance(query_vector)).label('similarity')
            )

            # 添加颜色过滤条件
            if color_filter:
                query = query.filter(Product.color == color_filter)
            elif colors_filter:
                query = query.filter(Product.color.in_(colors_filter))
```

```
        # 添加相似度阈值过滤
        query = query.filter(
            (1 - Product.features.cosine_distance(query_vector)) >= similarity_threshold
        )

        # 按相似度降序排列并限制结果数量
        results = query.order_by(
            (1 - Product.features.cosine_distance(query_vector)).desc()
        ).limit(limit).all()

        return [(product, float(similarity)) for product, similarity in results]

def hybrid_search(
    self,
    query_vector: np.ndarray,
    color_filter: Optional[str] = None,
    price_range: Optional[Tuple[float, float]] = None,
    limit: int = 10
) -> List[Tuple[Product, float]]:
    """
    混合搜索：向量相似度 + 多重标量字段过滤
    """
    with self.SessionLocal() as session:
        query = session.query(
            Product,
            (1 - Product.features.cosine_distance(query_vector)).label('similarity')
        )

        # 颜色过滤
        if color_filter:
            query = query.filter(Product.color == color_filter)

        # 可以扩展更多过滤条件，比如价格范围
        # if price_range:
        #     query = query.filter(Product.price.between(price_range[0], price_range[1]))

        results = query.order_by(
            (1 - Product.features.cosine_distance(query_vector)).desc()
        ).limit(limit).all()

        return [(product, float(similarity)) for product, similarity in results]

def get_similar_products_by_description(
    self,
    description: str,
    embedding_function,  # 文本转向量的函数
    color_filter: Optional[str] = None,
    limit: int = 5
) -> List[Tuple[Product, float]]:
    """
    根据描述文本查找相似产品

    Args:
        description: 产品描述文本
        embedding_function: 将文本转换为向量的函数
```

```
            color_filter: 颜色过滤
            limit: 结果数量限制
        """
        # 将描述转换为向量
        query_vector = embedding_function(description)

        return self.vector_search_with_color_filter(
            query_vector=query_vector,
            color_filter=color_filter,
            limit=limit
        )

    def advanced_search(
        self,
        query_vector: np.ndarray,
        filters: dict,
        limit: int = 10
    ) -> List[Tuple[Product, float]]:
        """
        高级搜索：支持动态过滤条件

        Args:
            query_vector: 查询向量
            filters: 过滤条件字典，如 {'color': 'black', 'colors': ['black', 'white']}
            limit: 结果数量限制
        """
        with self.SessionLocal() as session:
            query = session.query(
                Product,
                (1 - Product.features.cosine_distance(query_vector)).label('similarity')
            )

            # 动态添加过滤条件
            if 'color' in filters:
                query = query.filter(Product.color == filters['color'])

            if 'colors' in filters:
                query = query.filter(Product.color.in_(filters['colors']))

            if 'name_contains' in filters:
                query = query.filter(Product.name.ilike(f"%{filters['name_contains']}%"))

            results = query.order_by(
                (1 - Product.features.cosine_distance(query_vector)).desc()
            ).limit(limit).all()

            return [(product, float(similarity)) for product, similarity in results]


# 使用示例
def example_usage():
    """使用示例"""

    # 初始化搜索服务
    search_service = ProductVectorSearch(DATABASE_URL)
```

```
# 模拟添加一些手机产品数据
sample_products = [
    {
        "id": 1,
        "name": "iPhone 15 Pro",
        "color": "black",
        "description": "高端智能手机，配备A17 Pro芯片，钛金属机身，三摄系统",
        "features": np.random.rand(512).astype(np.float32)  # 实际应用中这里是真实的嵌入向量
    },
    {
        "id": 2,
        "name": "Samsung Galaxy S24",
        "color": "white",
        "description": "Android旗舰手机，骁龙8 Gen3处理器，AI拍照功能",
        "features": np.random.rand(512).astype(np.float32)
    },
    {
        "id": 3,
        "name": "小米14 Pro",
        "color": "blue",
        "description": "徕卡影像系统，骁龙8 Gen3，120W快充",
        "features": np.random.rand(512).astype(np.float32)
    }
]

# 添加产品到数据库
for product_data in sample_products:
    search_service.add_product(**product_data)

# 示例1：基于向量查询，过滤黑色手机
query_vector = np.random.rand(512).astype(np.float32)
results = search_service.vector_search_with_color_filter(
    query_vector=query_vector,
    color_filter="black",
    limit=5
)

print("基于向量查询，过滤黑色手机:")
for product, similarity in results:
    print(f"  {product.name} (相似度: {similarity:.3f})")

# 示例2：多颜色过滤
results = search_service.vector_search_with_color_filter(
    query_vector=query_vector,
    colors_filter=["black", "white"],
    limit=5
)

print("\n黑色或白色手机搜索结果:")
for product, similarity in results:
    print(f"  {product.name} - {product.color} (相似度: {similarity:.3f})")

# 示例3：高级搜索
results = search_service.advanced_search(
    query_vector=query_vector,
    filters={
```

```
            "colors": ["blue", "white"],
            "name_contains": "Pro"
        },
        limit=3
    )

    print("\n高级搜索结果（蓝色或白色 + 名称包含Pro):")
    for product, similarity in results:
        print(f"  {product.name} - {product.color} (相似度: {similarity:.3f})")


if __name__ == "__main__":
    # 运行示例
    example_usage()
```

示例2：BM25多路召回和融合排序

融合排序通常用于信息检索和推荐系统中，旨在将多个排序列表（如来自不同算法或模型的搜索结果）合并，以提高排序质量。最终得到更优的排序列表。

本示例展示了一个结合了BM25全文检索和向量语义搜索的程序，并提供了两种融合方法：线性加权融合和RRF（Reciprocal Rank Fusion）融合。

- 加权求和：一种基于分数的融合方法，为每个列表分配权重，计算每个项目的加权分数总和，然后根据最终分数排序。

  示例中，分别设置了向量语义检索，和BM25检索的权重，并对向量检索和BM25查询的结果进行归一化处理。

- RRF（Reciprocal Rank Fusion）融合算法：一种基于排名的融合方法，通过计算每个项目在每个列表中的排名的倒数之和来得到最终分数，常用于合并来自不同源的排序结果。

  RRF算法的优势在于它不依赖于具体的评分机制，只利用排名信息，因此可以融合来自不同来源、不同评分标准的排序列表。同时，由于使用了倒数函数，排名越靠前（数值小）的文档贡献的分数越大，且随着排名靠后，贡献的分数会迅速减小。

  在本例中，RRF算法将BM25和向量搜索的排名进行融合，从而结合了关键词匹配和语义相似度的优势，得到更全面的搜索结果。

另外，本例提供了分别执行BM25和向量搜索的方法，以及一个测试函数，便于对比两种融合方法的结果和性能。

```
"""
BM25 + 向量融合检索Demo
基于VexDB实现BM25全文检索与向量相似度融合搜索

数据表结构:
products (
  id INT PRIMARY KEY,        #产品ID
  name TEXT NOT NULL,        #产品名称
  color TEXT NOT NULL,       #颜色
  description TEXT NOT NULL, #产品描述
  title_tokens TEXT NOT NULL,#标题关键字
  embedding_vec floatvector(384)  #向量嵌入
)

主要特性:
1. BM25和向量搜索分别在本地Python中执行，避免SQL中的复杂计算
2. 支持两种融合方法:
   - 线性融合: BM25分数经过min-max归一化后，加权融合 (默认权重: 文本0.6, 向量0.4)
   - RRF融合: Reciprocal Rank Fusion, 平滑参数k=60
3. 向量搜索增加了相似度阈值判断 (similarity > 0.3)，过滤低相关性结果

使用示例:
    demo = HybridSearchDemo()
    results = demo.hybrid_search("iPhone 15 Pro", fusion_method="linear")
    results = demo.hybrid_search("旗舰手机", fusion_method="rrf")
"""

import numpy as np
import psycopg2
from psycopg2 import sql
from psycopg2.extras import DictCursor
import logging
import time
from typing import List, Dict, Optional

# 设置日志
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# 尝试导入transformers库
try:
    from sentence_transformers import SentenceTransformer
    TRANSFORMER_AVAILABLE = True
    logger.info("sentence-transformers 可用")
except ImportError:
    TRANSFORMER_AVAILABLE = False
    logger.warning("sentence-transformers 不可用，将使用随机向量")

class HybridSearchDemo:
    """融合检索demo"""

    def __init__(self, host: str = "localhost", port: int = 5438,
                 dbname: str = "postgres", user: str = "test", password: str = "Test@1234",
                 model_name: str = "sentence-transformers/all-MiniLM-L6-v2"):
        self.host = host
        self.port = port
```

```
        self.dbname = dbname
        self.user = user
        self.password = password
        self.table_name = "products"
        self.model_name = model_name

        # 初始化transformer模型
        self.model = None
        self.vector_size = 384  # 固定向量维度
        self.transformer_available = TRANSFORMER_AVAILABLE

        if self.transformer_available:
            try:
                logger.info(f"正在加载模型: {model_name}")
                self.model = SentenceTransformer(model_name)
                logger.info(f"模型加载成功，向量维度: {self.vector_size}")
            except Exception as e:
                logger.error(f"模型加载失败: {e}")
                self.model = None
                self.transformer_available = False

        # 保持单一连接
        self.conn = None

    def get_connection(self):
        """获取单一数据库连接"""
        if not self.conn:
            try:
                self.conn = psycopg2.connect(
                    host=self.host, port=self.port,
                    dbname=self.dbname, user=self.user, password=self.password,
                    connect_timeout=10
                )
                logger.info(f"单一连接建立成功: {self.host}:{self.port}")
            except Exception as e:
                logger.error(f"连接失败: {e}")
                self.conn = None
        return self.conn

    def init_data(self):
        """初始化数据"""
        conn = self.get_connection()
        if not conn:
            return False

        try:
            with conn.cursor() as cur:
                # 创建表
                cur.execute(f"""
                    DROP TABLE IF EXISTS {self.table_name};
                    CREATE TABLE {self.table_name} (
                        id INT PRIMARY KEY,
                        name TEXT NOT NULL,
                        color TEXT NOT NULL,
                        description TEXT NOT NULL,
                        title_tokens TEXT NOT NULL,
```

```
            embedding_vec floatvector({self.vector_size})
        );
    """)

    # 插入测试数据
    products = [
        (1, "iPhone 15 Pro Max", "蓝色",
         "Apple iPhone 15 Pro Max 苹果旗舰手机，配备A17 Pro芯片，钛金属设计，支持5G网络，拥有三摄像头系统",
         ["苹果", "iPhone", "15", "Pro", "Max", "旗舰", "A17", "Pro", "芯片", "钛金属", "5G", "三摄像头"]),
        (2, "Galaxy S24 Ultra", "黑色",
         "Samsung Galaxy S24 Ultra 三星旗舰手机，搭载骁龙8 Gen3处理器，配备S Pen手写笔，支持卫星通信",
         ["三星", "Galaxy", "S24", "Ultra", "旗舰", "骁龙", "8", "Gen3", "S", "Pen", "手写笔", "卫星通信"]),
        (3, "小米14 Pro", "白色",
         "小米14 Pro 徕卡光学镜头，骁龙8 Gen3处理器，徕卡影像系统，120W快充，专业摄影手机",
         ["小米", "14", "Pro", "旗舰", "徕卡", "光学", "镜头", "骁龙", "8", "Gen3", "120W", "快充", "摄影"]),
        (4, "Mate 60 Pro", "紫色",
         "华为Mate 60 Pro，搭载麒麟9000S芯片，支持卫星通话，运行鸿蒙操作系统，超可靠玄武架构",
         ["华为", "Mate", "60", "Pro", "旗舰", "麒麟", "9000S", "卫星", "通话", "鸿蒙", "系统", "玄武架构"])
    ]

    # 生成文本数据
    all_texts = []
    for id, name, color, desc, tokens in products:
        search_text = f"{name} {color} {desc} {' '.join(tokens)}"
        all_texts.append(search_text)

    # 生成嵌入向量
    if self.transformer_available and self.model:
        logger.info(f"正在生成 {len(all_texts)} 个文本的语义嵌入向量...")
        embeddings = self.model.encode(all_texts, normalize_embeddings=True)
        logger.info("嵌入向量生成完成")
    else:
        logger.warning("使用随机向量作为回退方案")
        embeddings = []
        for i, text in enumerate(all_texts):
            seed = hash(text) % (2**32)
            np.random.seed(seed)
            vec = np.random.normal(0, 1, self.vector_size)
            vec = vec / np.linalg.norm(vec)
            embeddings.append(vec)

    # 插入数据
    for i, (id, name, color, desc, tokens) in enumerate(products):
        vector = embeddings[i].tolist() if hasattr(embeddings[i], 'tolist') else embeddings[i].astype(float).tolist()
        # 将 tokens 数组转换为空格分隔的字符串
        tokens_str = " ".join(tokens)

        cur.execute(f"""
            INSERT INTO {self.table_name}
            (id, name, color, description, title_tokens, embedding_vec)
            VALUES (%s, %s, %s, %s, %s, %s);
        """, (id, name, color, desc, tokens_str, vector))

    # 创建索引
    try:
        # BM25索引覆盖 description 和 title_tokens 字段
```

```
                    cur.execute(f"CREATE INDEX text_idx_{self.table_name} ON {self.table_name} USING fulltext(description, title_tokens);")
                    cur.execute(f"CREATE INDEX vector_idx_{self.table_name} ON {self.table_name} USING hnsw (embedding_vec floatvector_cosine_ops);")
                    logger.info("索引创建成功")
                except Exception as e:
                    logger.warning(f"索引创建失败: {e}")

                # 验证数据插入
                cur.execute(f"SELECT COUNT(*) FROM {self.table_name};")
                count = cur.fetchone()[0]
                logger.info(f"数据初始化成功，插入了 {count} 条记录")

                conn.commit()
                return True

        except Exception as e:
            logger.error(f"数据初始化失败: {e}")
            import traceback
            traceback.print_exc()
            return False

    def hybrid_search(self, query: str, text_weight: float = 0.6, vector_weight: float = 0.4,
                      top_n: int = 5, fusion_method: str = "linear"):
        """BM25 + 向量融合搜索

        Args:
            query: 搜索查询
            text_weight: 文本搜索权重 (线性融合时使用)
            vector_weight: 向量搜索权重 (线性融合时使用)
            top_n: 返回结果数量
            fusion_method: 融合方法，可选 'linear' 或 'rrf'
        """
        conn = self.get_connection()
        if not conn:
            return []

        try:
            # 确保没有未完成的交易
            conn.rollback()

            # 生成查询向量
            if self.transformer_available and self.model:
                query_embedding = self.model.encode(query, normalize_embeddings=True)
                query_vector = query_embedding.tolist() if hasattr(query_embedding, 'tolist') else query_embedding.astype(float).tolist()
            else:
                seed = hash(query) % (2**32)
                np.random.seed(seed)
                query_vector = np.random.normal(0, 1, self.vector_size)
                query_vector = (query_vector / np.linalg.norm(query_vector)).tolist()

            # 1. 执行BM25文本搜索
            bm25_results = self._bm25_search(conn, query, top_n * 2)

            # 2. 执行向量搜索
            vector_results = self._vector_search(conn, query_vector, top_n * 2)

            # 3. 在本地进行融合
```

```
            if fusion_method == "linear":
                return self._linear_fusion(bm25_results, vector_results, top_n, text_weight, vector_weight)
            elif fusion_method == "rrf":
                return self._rrf_fusion(bm25_results, vector_results, top_n, k=60)
            else:
                logger.warning(f"未知的融合方法: {fusion_method}，使用线性融合")
                return self._linear_fusion(bm25_results, vector_results, top_n, text_weight, vector_weight)

        except Exception as e:
            logger.error(f"搜索失败: {e}")
            import traceback
            traceback.print_exc()
            return []

    def _bm25_search(self, conn, query: str, limit: int):
        """执行BM25文本搜索"""
        try:
            with conn.cursor(cursor_factory=DictCursor) as cur:
                desc_query = query + " @<PARAMS:BOOST=1.0>@"
                title_tokens_query = query + " @<PARAMS:BOOST=2.0>@"

                sql = f"""
                SELECT id, name, color, description, bm25_score() as "SCORE"
                FROM {self.table_name}
                WHERE description @~@ '{desc_query}' AND title_tokens @~@ '{title_tokens_query}'
                ORDER BY bm25_score DESC
                LIMIT {limit}
                """
                cur.execute(sql)
                results = cur.fetchall()
                return [dict(row) for row in results]
        except Exception as e:
            logger.warning(f"BM25搜索失败: {e}")
            return []

    def _vector_search(self, conn, query_vector: List[float], limit: int):
        """执行向量搜索"""
        try:
            with conn.cursor(cursor_factory=DictCursor) as cur:
                sql = f"""
                SELECT id, name, color, description,
                       (1 - (embedding_vec <=> ARRAY{query_vector})) AS "SIMILARITY"
                FROM {self.table_name}
                WHERE (1 - (embedding_vec <=> ARRAY{query_vector})) > 0.3
                ORDER BY embedding_vec <=> ARRAY{query_vector}
                LIMIT {limit}
                """
                cur.execute(sql)
                results = cur.fetchall()
                return [dict(row) for row in results]
        except Exception as e:
            logger.warning(f"向量搜索失败: {e}")
            return []

    def _linear_fusion(self, bm25_results: List[Dict], vector_results: List[Dict],
                       top_n: int, text_weight: float, vector_weight: float):
```

```
        """线性融合：对BM25分数进行min-max归一化后加权融合"""
        if not bm25_results and not vector_results:
            return []

        # 将结果按id索引
        all_results = {}

        # 处理BM25结果 - 进行min-max归一化
        if bm25_results:
            scores = [r.get("SCORE", 0) for r in bm25_results]
            min_score = min(scores)
            max_score = max(scores)
            score_range = max_score - min_score if max_score > min_score else 1.0

            for result in bm25_results:
                doc_id = result["id"]
                raw_score = result.get("SCORE", 0)
                # min-max归一化
                normalized_score = (raw_score - min_score) / score_range if score_range > 0 else 0

                if doc_id not in all_results:
                    all_results[doc_id] = {
                        "id": doc_id,
                        "name": result["name"],
                        "color": result["color"],
                        "description": result["description"],
                        "text_score": normalized_score,
                        "vector_score": 0,
                        "hybrid_score": 0
                    }
                else:
                    all_results[doc_id]["text_score"] = normalized_score

        # 处理向量结果
        if vector_results:
            for result in vector_results:
                doc_id = result["id"]
                similarity = result.get("SIMILARITY", 0)

                if doc_id not in all_results:
                    all_results[doc_id] = {
                        "id": doc_id,
                        "name": result["name"],
                        "color": result["color"],
                        "description": result["description"],
                        "text_score": 0,
                        "vector_score": similarity,
                        "hybrid_score": 0
                    }
                else:
                    all_results[doc_id]["vector_score"] = similarity

        # 计算融合分数
        for doc_id, result in all_results.items():
            result["hybrid_score"] = (result["text_score"] * text_weight +
                                      result["vector_score"] * vector_weight)
```

```
        # 按融合分数排序并返回top_n
        sorted_results = sorted(all_results.values(), key=lambda x: x["hybrid_score"], reverse=True)
        return sorted_results[:top_n]

    def _rrf_fusion(self, bm25_results: List[Dict], vector_results: List[Dict], top_n: int, k: int = 60):
        """RRF融合：Reciprocal Rank Fusion"""
        rrf_scores = {}

        # 处理BM25结果
        for rank, result in enumerate(bm25_results, start=1):
            doc_id = result["id"]
            score = 1.0 / (k + rank)

            if doc_id not in rrf_scores:
                rrf_scores[doc_id] = {
                    "id": doc_id,
                    "name": result["name"],
                    "color": result["color"],
                    "description": result["description"],
                    "rrf_score": score,
                    "text_rank": rank,
                    "vector_rank": None
                }
            else:
                rrf_scores[doc_id]["rrf_score"] += score
                rrf_scores[doc_id]["text_rank"] = rank

        # 处理向量结果
        for rank, result in enumerate(vector_results, start=1):
            doc_id = result["id"]
            score = 1.0 / (k + rank)

            if doc_id not in rrf_scores:
                rrf_scores[doc_id] = {
                    "id": doc_id,
                    "name": result["name"],
                    "color": result["color"],
                    "description": result["description"],
                    "rrf_score": score,
                    "text_rank": None,
                    "vector_rank": rank
                }
            else:
                rrf_scores[doc_id]["rrf_score"] += score
                rrf_scores[doc_id]["vector_rank"] = rank

        # 转换为列表并排序
        results = list(rrf_scores.values())
        results.sort(key=lambda x: x["rrf_score"], reverse=True)

        # 只返回top_n，并添加文本分数和向量分数（如果有的话）
        final_results = results[:top_n]
        for result in final_results:
            # 从原始结果中查找对应的文本分数和向量分数
            bm25_match = next((r for r in bm25_results if r["id"] == result["id"]), None)
```

```
            vector_match = next((r for r in vector_results if r["id"] == result["id"]), None)

            result["text_score"] = bm25_match["SCORE"] if bm25_match else 0
            result["vector_score"] = vector_match["SIMILARITY"] if vector_match else 0
            result["hybrid_score"] = result["rrf_score"]

        return final_results

    def search_separate(self, query: str, top_n: int = 5):
        """分别执行BM25和向量搜索，用于对比"""
        conn = self.get_connection()
        if not conn:
            return {"bm25": [], "vector": []}

        try:
            conn.rollback()

            # 生成查询向量
            if self.transformer_available and self.model:
                query_embedding = self.model.encode(query, normalize_embeddings=True)
                query_vector = query_embedding.tolist() if hasattr(query_embedding, 'tolist') else query_embedding.astype(float).tolist()
            else:
                seed = hash(query) % (2**32)
                np.random.seed(seed)
                query_vector = np.random.normal(0, 1, self.vector_size)
                query_vector = (query_vector / np.linalg.norm(query_vector)).tolist()

            # 使用新的辅助方法进行搜索
            bm25_results = self._bm25_search(conn, query, top_n)
            vector_results = self._vector_search(conn, query_vector, top_n)

            return {
                "bm25": bm25_results,
                "vector": vector_results
            }

        except Exception as e:
            logger.error(f"搜索失败: {e}")
            return {"bm25": [], "vector": []}

    def test_search(self, query: str):
        """测试搜索，比较两种融合方法"""
        print(f"\n搜索: {query}")
        print("=" * 80)

        # 1. 线性融合搜索
        print("[线性融合搜索] BM25分数min-max归一化 + 加权融合 (文本权重0.6, 向量权重0.4)")
        print("-" * 40)
        start_time = time.time()
        linear_results = self.hybrid_search(query, fusion_method="linear")
        linear_time = time.time() - start_time

        if linear_results:
            print(f"找到 {len(linear_results)} 条结果 (耗时: {linear_time:.3f}s):")
            for i, result in enumerate(linear_results, 1):
                print(f"{i}. 【{result['name']}】颜色: {result['color']}")
```

```
            print(f"   描述: {result['description'][:80]}...")
            print(f"   BM25分数: {result.get('text_score', 0):.4f}")
            print(f"   向量相似度: {result.get('vector_score', 0):.4f}")
            print(f"   融合分数: {result.get('hybrid_score', 0):.4f}")
            print()

    # 2. RRF融合搜索
    print("\n[RRF融合搜索] Reciprocal Rank Fusion, k=60")
    print("-" * 40)
    start_time = time.time()
    rrf_results = self.hybrid_search(query, fusion_method="rrf")
    rrf_time = time.time() - start_time

    if rrf_results:
        print(f"找到 {len(rrf_results)} 条结果 (耗时: {rrf_time:.3f}s):")
        for i, result in enumerate(rrf_results, 1):
            text_rank = result.get('text_rank') if result.get('text_rank') else 'N/A'
            vector_rank = result.get('vector_rank') if result.get('vector_rank') else 'N/A'
            print(f"{i}. 【{result['name']}】颜色: {result['color']}")
            print(f"   描述: {result['description'][:80]}...")
            print(f"   BM25排名: {text_rank}, 向量排名: {vector_rank}")
            print(f"   RRF分数: {result.get('hybrid_score', 0):.4f}")
            print()

    # 3. 分别搜索对比
    print("\n[分别搜索对比]:")
    print("-" * 40)
    start_time = time.time()
    separate_results = self.search_separate(query)
    separate_time = time.time() - start_time

    # BM25结果
    print(f"[BM25搜索结果]:")
    if separate_results["bm25"]:
        for i, result in enumerate(separate_results["bm25"], 1):
            print(f"  {i}. 【{result['name']}】颜色: {result['color']}(分数: {result.get('SCORE', 0):.4f})")
    else:
        print("  无结果")

    # 向量搜索结果
    print(f"\n[向量搜索结果]:")
    if separate_results["vector"]:
        for i, result in enumerate(separate_results["vector"], 1):
            print(f"  {i}. 【{result['name']}】颜色: {result['color']}(相似度: {result.get('SIMILARITY', 0):.4f})")
    else:
        print("  无结果")

    # 4. 结果分析
    print(f"\n[搜索性能与结果分析]:")
    print(f"  线性融合搜索耗时: {linear_time:.3f}s")
    print(f"  RRF融合搜索耗时: {rrf_time:.3f}s")
    print(f"  分别搜索耗时: {separate_time:.3f}s")

    if linear_results:
        bm25_hits = sum(1 for r in linear_results if r.get('text_score', 0) > 0)
        vector_hits = sum(1 for r in linear_results if r.get('vector_score', 0) > 0)
```

```
            print(f"  线性融合中 - BM25命中: {bm25_hits}条, 向量命中: {vector_hits}条")

        if rrf_results and linear_results:
            # 比较两种融合方法的结果差异
            linear_ids = [r['id'] for r in linear_results]
            rrf_ids = [r['id'] for r in rrf_results]
            common = set(linear_ids) & set(rrf_ids)
            print(f"  两种融合方法共同命中: {len(common)}条")
            print(f"  线性融合特有: {len(set(linear_ids) - set(rrf_ids))}条")
            print(f"  RRF融合特有: {len(set(rrf_ids) - set(linear_ids))}条")

        # 语义相似度测试
        if self.transformer_available and len(linear_results) >= 2:
            try:
                text1 = f"{linear_results[0]['name']} {linear_results[0]['color']} {linear_results[0]['description']}"
                text2 = f"{linear_results[1]['name']} {linear_results[1]['color']} {linear_results[1]['description']}"
                similarity = self.model.similarity([text1], [text2])[0][0]
                print(f"  Top1与Top2语义相似度: {similarity:.4f}")
            except Exception as e:
                logger.debug(f"语义相似度计算失败: {e}")

        return linear_results

    def close(self):
        """关闭连接"""
        if self.conn:
            self.conn.close()
            self.conn = None
            logger.info("数据库连接已关闭")

def main():
    print("=" * 80)
    print("BM25 + 向量融合检索Demo")
    print("使用sentence-transformers生成语义嵌入向量")
    print("基于VexDB实现BM25全文检索与向量相似度融合")
    print("=" * 80)

    demo = HybridSearchDemo()

    if not demo.transformer_available:
        print("\n[sentence-transformers库未安装]")
        print("请运行: pip install sentence-transformers")
        print("当前将使用随机向量进行演示")
        print()

    try:
        if not demo.init_data():
            print("数据初始化失败")
            return

        print("\n数据准备完成，开始测试搜索\n")

        # 测试查询
        test_queries = [
            "苹果 iPhone",
            "徕卡相机",
```

```
            "卫星通话",
            "快充手机",
            "华为手机",
            "小米 徕卡"
        ]

        for query in test_queries:
            demo.test_search(query)
            print("=" * 80)

        print("所有测试完成!")

    except Exception as e:
        logger.error(f"测试失败: {e}")
        import traceback
        traceback.print_exc()
    finally:
        demo.close()

if __name__ == "__main__":
    main()
```

编排组件

VexDB 实现了对常见 AI 应用开发平台的适配，助力构建各类大语言模型应用。

通过阅读本章内容，您可以使用以下组件轻松搭建 RAG 服务，并将 VexDB 作为 RAG 的语料库。

- LangChain
- Dify
- MaxKB
- RAGFlow
- Open WebUI

LangChain

LangChain 是一个用于开发由大语言模型驱动的应用程序的框架。

本文介绍如何将 VexDB 作为 LangChain 向量存储使用。通过本方案可实现企业级向量数据的持久化存储，支持复杂的元数据过滤，并直接利用 VexDB 的事务特性保障数据一致性。

前提条件

在进行 LangChain 的安装之前，请确保已经参考 安装 VexDB 的内容完成了数据库的安装，并部署了 Python3 环境（Python ≥ 3.9）。

安装 LangChain-VexDB 插件包

```
pip install langchain_vexdb-{version}-py3-none-any.whl
```

注意事项

- 维度一致性：确保VECTOR(n)的维度与嵌入模型输出维度一致。
- 选择合适的索引类型：如HNSW、IVFFlat、DiskANN，根据数据量和查询需求选择合适的索引类型。
  - HNSW 适合大规模数据。
  - IVFFlat 适合小规模数据。
  - DiskANN 适合大规模数据且需要高精度查询。

  索引参数需要根据实际情况调整。
- 距离度量：根据场景选择合适的方法（`<->` 为L2距离，`<=>` 为余弦距离）。

使用示例

1. 安装依赖包。 程序适配的 langchain-core 版本范围是>=0.2.13且<0.4.0。 `shell # 核心依赖 pip install langchain-core=0.2.13 psycopg2-binary sqlalchemy numpy # 可选嵌入模型（示例使用cohere） pip install langchain_cohere`
2. 初始化实例。

```
from langchain_cohere import CohereEmbeddings
from langchain_vexdb import VexdbVector
from langchain_vexdb.vectorstores import VexdbVector
from langchain_core.documents import Document

# See docker command above to launch a postgres instance with pgvector enabled.
connection = "postgresql+psycopg://langchain:langchain@localhost:6024/langchain"
collection_name = "my_docs"
embeddings = CohereEmbeddings()

vectorstore = VexdbVector(
    embeddings=embeddings,
    collection_name=collection_name,
    connection=connection,
    use_jsonb=True,
)
```

3. 文档存储。

```
docs = [
    Document(page_content='there are cats in the pond', metadata={"id": 1, "location": "pond", "topic": "animals"}),
    Document(page_content='ducks are also found in the pond', metadata={"id": 2, "location": "pond", "topic": "animals"}),
    Document(page_content='fresh apples are available at the market', metadata={"id": 3, "location": "market", "topic": "food"}),
    Document(page_content='the market also sells fresh oranges', metadata={"id": 4, "location": "market", "topic": "food"}),
    Document(page_content='the new art exhibit is fascinating', metadata={"id": 5, "location": "museum", "topic": "art"}),
    Document(page_content='a sculpture exhibit is also at the museum', metadata={"id": 6, "location": "museum", "topic": "art"}),
    Document(page_content='a new coffee shop opened on Main Street', metadata={"id": 7, "location": "Main Street", "topic": "food"}),
    Document(page_content='the book club meets at the library', metadata={"id": 8, "location": "library", "topic": "reading"}),
    Document(page_content='the library hosts a weekly story time for kids', metadata={"id": 9, "location": "library", "topic": "reading"}),
    Document(page_content='a cooking class for beginners is offered at the community center', metadata={"id": 10, "location": "community center", "topic": "classes"})
]

# 存储文档
vectorstore.add_documents(docs, ids=[doc.metadata['id'] for doc in docs])
```

4. 进行向量检索。

使用 similarity_search 方法进行向量检索。例如，查询与 “kitty” 相关的文档。

```
vectorstore.similarity_search('kitty', k=10)

# 查询结果输出
[Document(page_content='there are cats in the pond', metadata={'id': 1, 'topic': 'animals', 'location': 'pond'}),
Document(page_content='the book club meets at the library', metadata={'id': 8, 'topic': 'reading', 'location': 'library'}),
Document(page_content='the library hosts a weekly story time for kids', metadata={'id': 9, 'topic': 'reading', 'location': 'library'}),
Document(page_content='the new art exhibit is fascinating', metadata={'id': 5, 'topic': 'art', 'location': 'museum'}),
Document(page_content='ducks are also found in the pond', metadata={'id': 2, 'topic': 'animals', 'location': 'pond'}),
Document(page_content='the market also sells fresh oranges', metadata={'id': 4, 'topic': 'food', 'location': 'market'}),
Document(page_content='a cooking class for beginners is offered at the community center', metadata={'id': 10, 'topic': 'classes', 'location': 'community center'}),
Document(page_content='fresh apples are available at the market', metadata={'id': 3, 'topic': 'food', 'location': 'market'}),
Document(page_content='a sculpture exhibit is also at the museum', metadata={'id': 6, 'topic': 'art', 'location': 'museum'}),
Document(page_content='a new coffee shop opened on Main Street', metadata={'id': 7, 'topic': 'food', 'location': 'Main Street'})]
```

过滤器支持

向量存储支持一组可应用于文档的元数据字段的过滤器。

过滤器	含义/类别
eq < /td > < td > *Equality*(= =) < /td > < /tr > < tr > < td>ne	Inequality (!=)
lt</td> <td>Less than (<)</td> </tr> <tr> <td>lte	Less than or equal (<=)
gt</td> <td>Greater than (>)</td> </tr> <tr> <td>gte	Greater than or equal (>=)
in < /td > < td > *SpecialCased*(*in*) < /td > < /tr > < tr > < td>nin	Special Cased (not in)
between < /td > < td > *SpecialCased*(*between*) < /td > < /tr > < tr > < td>exists	Special Cased (is null)
like < /td > < td > *Text*(*like*) < /td > < /tr > < tr > < td>ilike	Text (case-insensitive like)
and < /td > < td > *Logical*(*and*) < /td > < /tr > < tr > < td>or	Logical (or)

5. 进行带过滤器的查询：

```
vectorstore.similarity_search('kitty', k=10, filter={
    'id': {'$in': [1, 5, 2, 9]}
})

# 查询结果输出
[Document(page_content='there are cats in the pond', metadata={'id': 1, 'topic': 'animals', 'location': 'pond'}),
Document(page_content='the library hosts a weekly story time for kids', metadata={'id': 9, 'topic': 'reading', 'location': 'library'}),
Document(page_content='the new art exhibit is fascinating', metadata={'id': 5, 'topic': 'art', 'location': 'museum'}),
Document(page_content='ducks are also found in the pond', metadata={'id': 2, 'topic': 'animals', 'location': 'pond'})]
```

如果你提供一个包含多个字段但没有操作符的字典，顶层将被解释为逻辑 AND 过滤器。

```
vectorstore.similarity_search('ducks', k=10, filter={
    'id': {'$in': [1, 5, 2, 9]},
    'location': {'$in': ["pond", "market"]}
})

# 查询结果输出
[Document(page_content='ducks are also found in the pond', metadata={'id': 2, 'topic': 'animals', 'location': 'pond'}),
Document(page_content='there are cats in the pond', metadata={'id': 1, 'topic': 'animals', 'location': 'pond'})]
```

```
vectorstore.similarity_search('ducks', k=10, filter={
    '$and': [
        {'id': {'$in': [1, 5, 2, 9]}},
        {'location': {'$in': ["pond", "market"]}},
    ]
})

# 查询结果输出
[Document(page_content='ducks are also found in the pond', metadata={'id': 2, 'topic': 'animals', 'location': 'pond'}),
Document(page_content='there are cats in the pond', metadata={'id': 1, 'topic': 'animals', 'location': 'pond'})]
```

```
vectorstore.similarity_search('bird', k=10, filter={
    'location': { "$ne": 'pond'}
})

# 查询结果输出
[Document(page_content='the book club meets at the library', metadata={'id': 8, 'topic': 'reading', 'location': 'library'}),
Document(page_content='the new art exhibit is fascinating', metadata={'id': 5, 'topic': 'art', 'location': 'museum'}),
Document(page_content='the library hosts a weekly story time for kids', metadata={'id': 9, 'topic': 'reading', 'location': 'library'}),
Document(page_content='a sculpture exhibit is also at the museum', metadata={'id': 6, 'topic': 'art', 'location': 'museum'}),
Document(page_content='the market also sells fresh oranges', metadata={'id': 4, 'topic': 'food', 'location': 'market'}),
Document(page_content='a cooking class for beginners is offered at the community center', metadata={'id': 10, 'topic': 'classes', 'location': 'community center'}),
Document(page_content='a new coffee shop opened on Main Street', metadata={'id': 7, 'topic': 'food', 'location': 'Main Street'}),
Document(page_content='fresh apples are available at the market', metadata={'id': 3, 'topic': 'food', 'location': 'market'})]
```

用于检索增强生成

有关如何使用此向量存储进行检索增强生成（RAG）的指南，请查阅以下内容：

检索 - Retrieval

查询示例： 使用高级筛选器

- 演示如何使用 `$nin` 筛选器查找不在特定列表中的项目：

```
results = vector_store.similarity_search(
    "kitty", k=10, filter={"id": {"$nin": [1, 5, 2]}})
    for doc in results:
print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$between` 筛选器查找在特定范围内的项目：

```
results = vector_store.similarity_search(
    "apples", k=10, filter={"id": {"$between": [3, 5]}})
for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$ilike` 筛选器进行不区分大小写的搜索：

```
results = vector_store.similarity_search(
    "book", k=10, filter={"page_content": {"$ilike": "%book%"}}
)
for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$and` 筛选器组合多个条件：

```
results = vector_store.similarity_search(
    "coffee", k=10, filter={"$and": [
        {"location": {"$eq": "Main Street"}},
        {"topic": {"$eq": "food"}}
    ]}
)

for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$or` 筛选器进行多条件之一的匹配：

```
results = vector_store.similarity_search(
    "reading", k=10, filter={"$or": [
        {"location": {"$eq": "library"}},
        {"location": {"$eq": "community center"}}
    ]}
)

for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$nin` 筛选器查找不在特定列表中的项目：

```
results = vector_store.similarity_search(
    "kitty", k=10, filter={"id": {"$nin": [1, 5, 2]}})

for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$between` 筛选器查找在特定范围内的项目：

```
results = vector_store.similarity_search(
    "apples", k=10, filter={"id": {"$between": [3, 5]}})

for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$ilike` 筛选器进行不区分大小写的搜索：

```
results = vector_store.similarity_search(
    "book", k=10, filter={"page_content": {"$ilike": "%book%"}}
)

for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$and` 筛选器组合多个条件：

```
results = vector_store.similarity_search(
    "coffee", k=10, filter={"$and": [
        {"location": {"$eq": "Main Street"}},
        {"topic": {"$eq": "food"}}
    ]}
)

for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

- 演示如何使用 `$or` 筛选器进行多条件之一的匹配：

```
results = vector_store.similarity_search(
    "reading", k=10, filter={"$or": [
        {"location": {"$eq": "library"}},
        {"location": {"$eq": "community center"}}
    ]}
)

for doc in results:
    print(f"* {doc.page_content} [{doc.metadata}]")
```

Dify

Dify 是一个开源的大语言模型应用开发平台，旨在帮助开发者快速构建和部署生成式 AI 应用。

本文介绍了如何部署 Dify，并使用 VexDB 实现 RAG 向量数据存储的教程。

前提条件

在部署 Dify 之前，请确保已经参考 安装 VexDB 的内容完成了数据库的安装，并部署了 Python3 环境。

安装 Dify

1. 从工程师处获取已经适配了 VexDB 的 Dify 镜像文件，使用如下命令将 Docker 镜像加载到本地：

   ```
   docker load -i vexdb.dify-api.{version}.tar.gz
   ```

2. 修改 docker-compose.yaml：

   Docker-Compose.yaml 是 Docker Compose 的配置文件，用于定义和启动多容器 Docker 应用，可以使用相关命令一键启动文件中定义的所有服务。

```
# ==================================================================
# WARNING: This file is auto-generated by generate_docker_compose
# Do not modify this file directly. Instead, update the .env.example
# or docker-compose-template.yaml and regenerate this file.
# ==================================================================

x-shared-env: &shared-api-worker-env
... 省略
VEXDB_HOST: ${VEXDB_HOST:-vexdb}
VEXDB_PORT: ${VEXDB_PORT:-5432}
VEXDB_USER: ${VEXDB_USER:-dify}
VEXDB_PASSWORD: ${VEXDB_PASSWORD:-Difyai123456}
VEXDB_DATABASE: ${VEXDB_DATABASE:-dify}
VEXDB_MIN_CONNECTION: ${VEXDB_MIN_CONNECTION:-1}
VEXDB_MAX_CONNECTION: ${VEXDB_MAX_CONNECTION:-5}

... 省略

services:
# API service
api:
image: vexdb/dify-api:1.7.1
restart: always
environment:
    # Use the shared environment variables.
    <<: *shared-api-worker-env
    # Startup mode, 'api' starts the API server.
    MODE: api
    SENTRY_DSN: ${API_SENTRY_DSN:-}
    SENTRY_TRACES_SAMPLE_RATE: ${API_SENTRY_TRACES_SAMPLE_RATE:-1.0}
    SENTRY_PROFILES_SAMPLE_RATE: ${API_SENTRY_PROFILES_SAMPLE_RATE:-1.0}
    PLUGIN_REMOTE_INSTALL_HOST: ${EXPOSE_PLUGIN_DEBUGGING_HOST:-localhost}
    PLUGIN_REMOTE_INSTALL_PORT: ${EXPOSE_PLUGIN_DEBUGGING_PORT:-5003}
    PLUGIN_MAX_PACKAGE_SIZE: ${PLUGIN_MAX_PACKAGE_SIZE:-52428800}
    INNER_API_KEY_FOR_PLUGIN: ${PLUGIN_DIFY_INNER_API_KEY:-QaHbTe77CtuXmsfyhR7+vRjI/+XbV1AaFy691iy+kGDv2Jvy0/eAh8Y1}
depends_on:
    db:
    condition: service_healthy
    redis:
    condition: service_started
volumes:
    # Mount the storage directory to the container, for storing user files.
    - ./volumes/app/storage:/app/api/storage
networks:
    - ssrf_proxy_network
    - default

# worker service
# The Celery worker for processing the queue.
worker:
image: vexdb/dify-api:1.7.1
restart: always
environment:
    # Use the shared environment variables.
    <<: *shared-api-worker-env
    # Startup mode, 'worker' starts the Celery worker for processing the queue.
```

```
    MODE: worker
    SENTRY_DSN: ${API_SENTRY_DSN:-}
    SENTRY_TRACES_SAMPLE_RATE: ${API_SENTRY_TRACES_SAMPLE_RATE:-1.0}
    SENTRY_PROFILES_SAMPLE_RATE: ${API_SENTRY_PROFILES_SAMPLE_RATE:-1.0}
    PLUGIN_MAX_PACKAGE_SIZE: ${PLUGIN_MAX_PACKAGE_SIZE:-52428800}
    INNER_API_KEY_FOR_PLUGIN: ${PLUGIN_DIFY_INNER_API_KEY:-QaHbTe77CtuXmsfyhR7+vRjI/+XbV1AaFy691iy+kGDv2Jvy0/eAh8Y1}
depends_on:
    db:
    condition: service_healthy
    redis:
    condition: service_started
volumes:
    # Mount the storage directory to the container, for storing user files.
    - ./volumes/app/storage:/app/api/storage
networks:
    - ssrf_proxy_network
    - default

# worker_beat service
# Celery beat for scheduling periodic tasks.
worker_beat:
image: vexdb/dify-api:1.7.1
restart: always
environment:
    # Use the shared environment variables.
    <<: *shared-api-worker-env
    # Startup mode, 'worker_beat' starts the Celery beat for scheduling periodic tasks.
    MODE: beat
depends_on:
    db:
    condition: service_healthy
    redis:
    condition: service_started
networks:
    - ssrf_proxy_network
    - default

... 省略
```

3. 手动在宿主机创建映射目录，并参考以下步骤分配目录权限：

```
# 在 docker-compose.yaml 文件目录下执行
mkdir vexdb
chown 1000:1000 vexdb
cd vexdb
mkdir lic
chown 1000:1000 lic
mkdir data
chown 1000:1000 data
mkdir backup
chown 1000:1000 backup
mkdir backup_log
chown 1000:1000 backup_log
```

4. 执行启动命令：

```
docker compose --profile vexdb up -d
```

访问 Dify

在浏览器中访问本地部署的Dify WEB 服务页面，在这里创建管理员账户，然后登录 Dify 主页以便进行下一步使用。

```
http://localhost/install
```

更多使用方法和指导，请查阅 Dify 官方文档。

MaxKB

MaxKB（Max Knowledge Brain）是一款基于 LLM 大语言模型的知识库问答系统。其核心目标是通过结合大模型的生成能力与结构化知识库的检索功能，提升问答的准确性和实用性。

MaxKB 中默认集成了 PostgreSQL 15，并借助开源向量插件 pgvector 兼顾了关系数据和向量数据的存储。本文介绍了使用 VexDB 作为 MaxKB 数据存储并构建 RAG 的工作流程。

> **说明**
>
> 当前仅支持 MaxKB V1 版本。

前提条件

在部署 MaxKB 之前，请确保已经参考 安装 VexDB 的内容完成了数据库的安装，并部署了 Python3 环境。

安装 MaxKB

1. 从工程师处获取适配了 VexDB 的 MaxKB 镜像文件压缩包，参考以下命令加载到本地环境。

   ```
   docker load -i vexdb.maxkb_{version}.tar.gz
   ```

2. 手动在宿主机创建映射目录，并参考以下步骤分配目录权限：

   ```
   # 在 docker-compose.yaml 文件目录下执行
   mkdir vexdb
   chown 1000:1000 vexdb
   cd vexdb
   mkdir lic
   chown 1000:1000 lic
   mkdir data
   chown 1000:1000 data
   mkdir backup
   chown 1000:1000 backup
   mkdir backup_log
   chown 1000:1000 backup_log
   ```

3. 编辑 docker-compse.yaml，填写 VexDB 连接信息。

   Docker-Compose.yaml 是 Docker Compose 的配置文件，用于定义和启动多容器 Docker 应用，可以使用相关命令一键启动文件中定义的所有服务。

```
services:
# 无需同时安装vexdb 则注释下面的配置
#db:
#    image: vexdb/vexdb-vector:3.0.8.25159
#    container_name: vexdb
#    restart: always
#    environment:
#    - VB_DB=maxkb
#    - VB_USERNAME=maxkb
#    - VB_PASSWORD=Maxkb@123456
#    ports:
#    - '5434:5432'
#    volumes:
#    - ./vexdb/lic:/home/vexdb/vexdb/lic
#    - ./vexdb/data:/home/vexdb/data
#    - ./vexdb/backup:/home/vexdb/backup
#    - ./vexdb/backup_log:/home/vexdb/backup_log
#    healthcheck:
#    test: ["CMD-SHELL", "pg_isready"]
#    interval: 10s
#    timeout: 5s
#    retries: 5

app:
    image: vexdb/maxkb:v1.0.1
    container_name: maxkb
    restart: always
    # depends_on:
    #   db:
    #     condition: service_healthy
    environment:
    - MAXKB_CONFIG_TYPE=ENV
    - MAXKB_VECTOR_STORE_NAME=vexdb
    - MAXKB_DB_ENGINE=django_vexdb_backend
    - MAXKB_DB_NAME=maxkb
    - MAXKB_DB_HOST=vexdb
    - MAXKB_DB_PORT=5432
    - MAXKB_DB_USER=maxkb
    - MAXKB_DB_PASSWORD=Maxkb@123456
    - MAXKB_DB_MAX_OVERFLOW=80
    - MAXKB_EMBEDDING_MODEL_NAME=/opt/maxkb/model/embedding/shibing624_text2vec-base-chinese
    - MAXKB_EMBEDDING_MODEL_PATH=/opt/maxkb/model/embedding
    - MAXKB_SANDBOX=1
    - LANG=en_US.UTF-8
    - PIP_TARGET=/opt/maxkb/app/sandbox/python-packages
    - PYTHONPATH=/opt/maxkb/app/sandbox/python-packages
    - PYTHONUNBUFFERED=1
    ports:
    - "8080:8080"
    volumes:
    - ./maxkb/data:/opt/maxkb/app/data
```

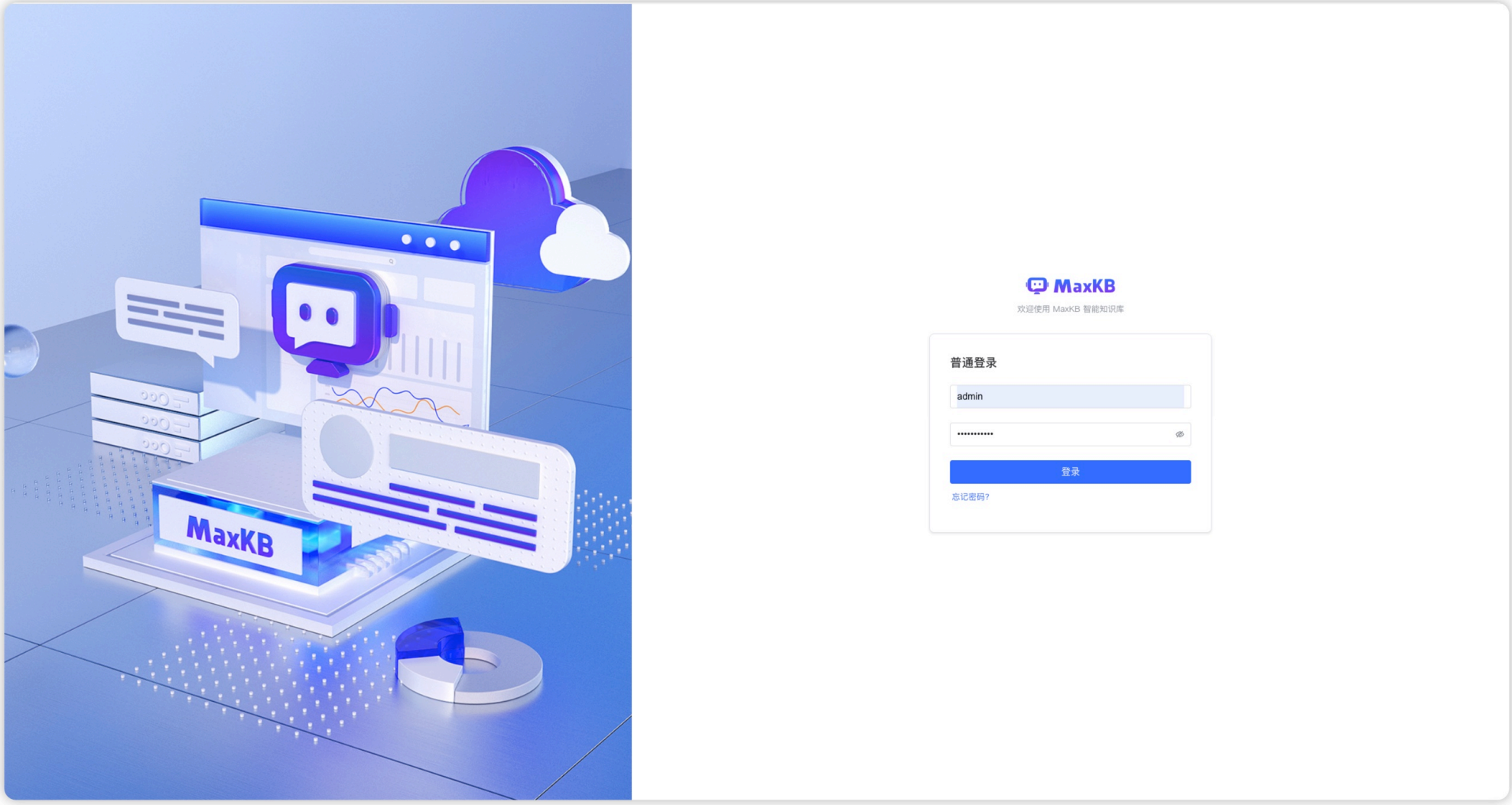
4. Docker Compose 一键启动。

```
docker compose up -d
```

访问 MaxKB

可以通过 `服务器ip+8080端口号` 直接访问 MaxKB。8080 是 MaxKB 的默认端口，请注意防火墙配置。

使用默认的管理员用户名 `admin` 和默认密码 `MaxKB@123..` 即可登录 MaxKB。



更多使用方法和指导，请查阅 MaxKB 官方文档。

RAGFlow

RAGFlow 是一款基于 “深度文档理解” 构建的开源 RAG (Retrieval-Augmented Generation) 引擎。

本文指导用户通过 VexDB 和 RAGFlow 构建 RAG 工作流程。

前提条件

在部署 RAGFlow 之前，请确保已经参考 安装 VexDB 的内容完成了数据库的安装，并部署了 Python3 环境。

安装 RAGFlow

1. 从工程师处获取适配了 VexDB 的 RAGFlow 镜像文件压缩包，参考以下命令加载到本地环境。

```
docker load -i vexdb.ragflow_{version}-slim.tar.gz
```

2. 从工程师处获取经过修改的 Docker Compose 启动脚本，并解压到本地。

```
tar -xzvf vexdb.ragflow.docker-compose.tar.gz
cd ragflow
```

3. 修改 `.env` 文件，填写 VexDB 连接信息。

```
# 设置DOC_ENGINE 为 vexdb
DOC_ENGINE=${DOC_ENGINE:-vexdb}
...
# 配置VEXDB数据库的访问地址
VEXDB_HOST=host.docker.internal
VEXDB_PORT=5434
VEXDB_DBNAME=rag_flow
VEXDB_USER=rag_flow
VEXDB_PASSWORD=Ragflow@123456
...
# 设置RAGFLOW_IMAGE镜像为我们的镜像
RAGFLOW_IMAGE=vexdb/ragflow:v0.19.0-slim
```

4. 启动服务。

```
docker compose up -d
```

访问 RAGFlow

在浏览器打开 RAGFlow 登录页面，点击 `sign up` 进行注册，填入电子邮箱和密码后，返回登录页使用刚刚注册的账号登录系统。

```
http://<服务器ip>/login
```

更多使用方法和指导，请查阅 RAGFlow: README。

Open WebUI

Open WebUI 是一个开源的、可拓展的 AI 应用平台，能够帮助轻松管理和使用 AI 模型。

本文介绍了使用 VexDB 作为向量数据存储并利用 Open WebUI 构建 RAG 的工作流程。

前提条件

在部署 Open WebUI 之前，请确保已经参考 安装 VexDB 的内容完成了数据库的安装，并部署了 Python3 环境，请确保您使用的是 Python 3.11 以避免兼容性问题。

安装 Open WebUI

1. 从工程师处获取适配了 VexDB 的 Open WebUI 镜像文件压缩包，参考以下命令加载到本地环境。

```
docker load -i vexdb.open-webui_{version}.tar.gz
```

2. 修改 `.env` 文件，填写 VexDB 连接信息。

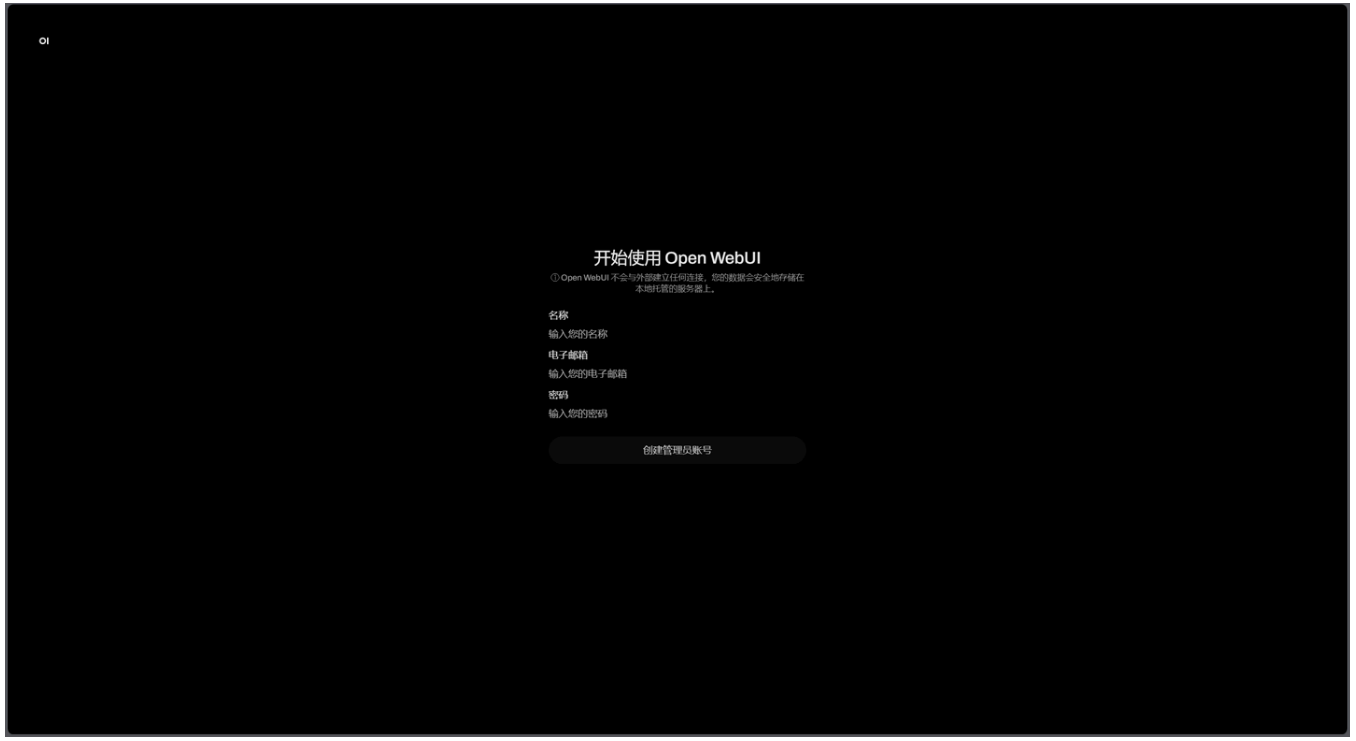
```
VECTOR_DB=vexdb
VEXDB_DB_URL=postgresql+psycopg2://test:Test%401234@host.docker.internal:5434/openwebui
VEXDB_INITIALIZE_MAX_VECTOR_LENGTH=1536
```

3. 在 `.env` 文件所在目录启动镜像。

```
docker run -d -p 3000:8080 -v open-webui:/app/backend/data --name open-webui --env-file .env vexdb/open-webui:v0.16.15
```

访问 Open WebUI

可以通过 `服务器ip+3000端口号` 直接访问 Open WebUI。点击 “开始使用” 后，填入用户名、邮箱和密码创建管理员：



更多使用方法和指导，请查阅 Open WebUI 官方文档。