专题一：PostgreSQL数据库内核技术研究实践大作业

一、postgreSQL数据库开发环境安装与使用

1.postgreSQL数据库软件安装与使用

1）在Windows10操作系统下，安装最新版本postgreSQL数据库软件。

首先打开postgresql官网，下载最新版postgresql17.RC1。

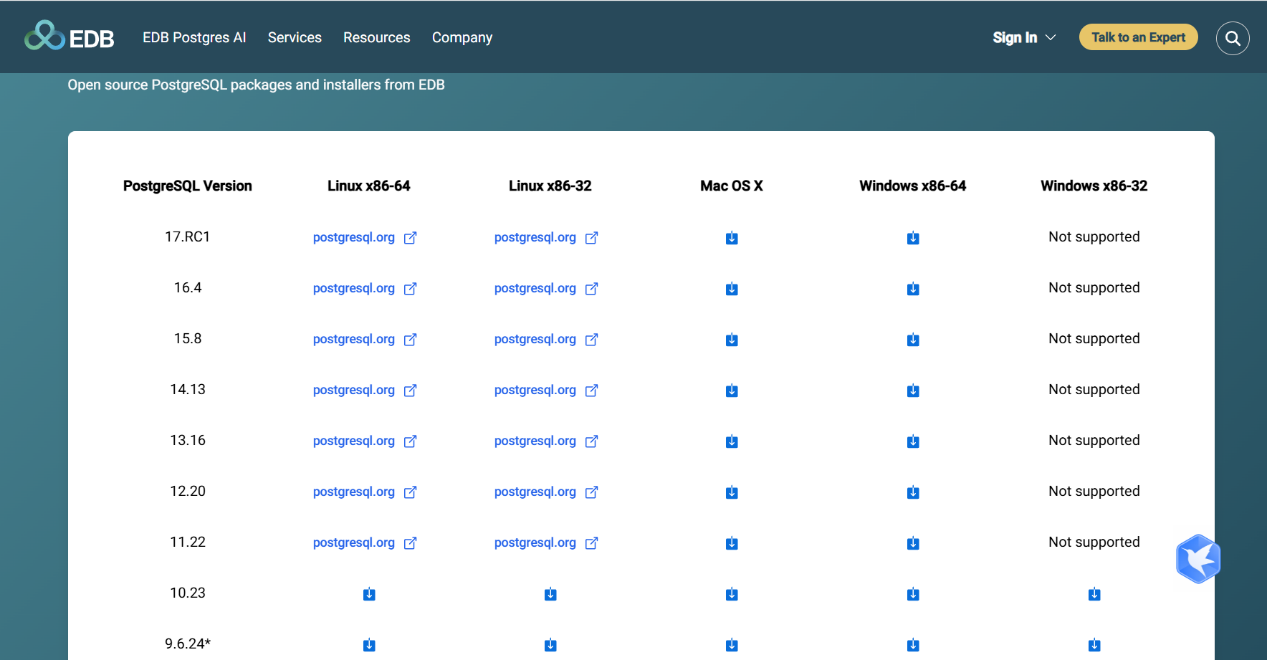


图1-1 官方下载界面

接着按照指定的步骤安装软件，最后SQL Shell中成功登录用户postgres。

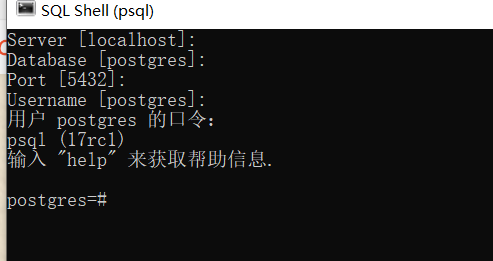


图1-2 SQL Shell中运行示例

值得一提的是，在postgresql 17 RC1安装的过程中，在安装包运行到最后一步时一直出现报错。在CSDN上查阅相关资料，有的说为安装的文件夹赋予更高的权限，尝试以后问题没有解决。又有的说安装过程中，语言得设置成英文，密码得是数字12345678，经尝试，问题得到解决，成功安装。

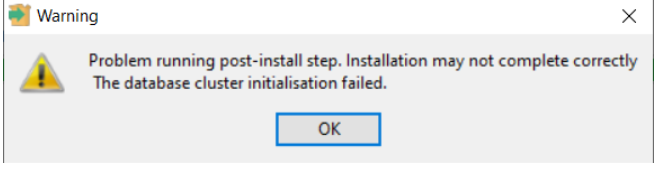


图1-3 postgresql 17 RC1安装报错信息

2）在安装数据库服务器后，使用postgreSQL数据库软件服务器程序工具pg\_ctl启停数据库实例、查看数据库实例状态。

首先要配置PGDATA这个环境变量，对应数据库实例的数据路径。然后是在Path中指定pg\_ctl的路径，因为pg\_ctl是bin目录下的一个可执行程序，在命令行中无法直接运行。

完成上述操作后，可以在命令正确运行pg\_ctl命令了。接下来使用pg\_ctl依次启动，关闭数据库，以及查看数据库的状态。

首先使用pg\_ctl start命令，启动数据库。

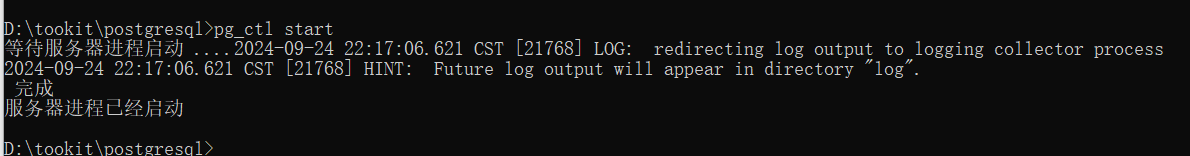


图1-4 pg\_ctl start命令启动数据库

运行pg\_ctl status命令，可以看到数据库成功启动

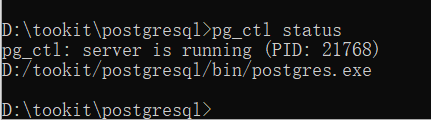


图1-5 pg\_ctl status确认成功启动数据库

接着运行pg\_ctl stop关闭服务器，成功关闭。

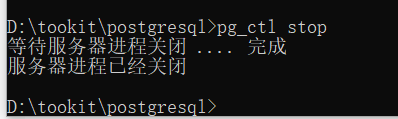


图1-6 pg\_ctl stop命令关闭数据库

运行pg\_ctl status命令，可以看到数据库成功关闭，无服务器在运行。

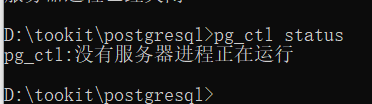


图1-7 pg\_ctl status命令确认成功关闭数据库

1. 说明initdb命令、createdb命令、create database命令之间是什么关系。

首先介绍一下上述命令的基本功能。initdb 用于初始化一个 PostgreSQL 数据目录，它会创建一个全新的数据库集群（包括 postgres 默认数据库和系统表），并在指定目录中创建相关的数据结构。值得一提的是，在windows系统中，一键安装程序时会自动运行initdb并完成数据库集群的创建和初始化，因此无需再人为的执行initdb命令；createdb 是一个 PostgreSQL 的命令行工具，用于创建新的数据库，是对 SQL 命令 create database 的一个封装；create database 是一个标准的 SQL 命令，用于创建新的数据库，通常在 psql 或其他 SQL 客户端中使用，用于创建一个新的数据库。

概括说来，initdb用于初始化数据库集群，createdb则是创建一个新的数据库，create database功能与createdb一致，区别在于前者是在命令行中运行，后者则是SQL命令，在psql或SQL客户端中使用。

2.postgreSQL数据库客户端管理工具pgAdmin4安装与使用

1）在Windows10操作系统下，安装postgreSQL数据库客户端管理工具pgAdmin4

在安装postgresql时，有多选框界面选择额外安装的软件，其中就有pgAdmin4，因此勾选后，pgAdmin4也一起安装了。

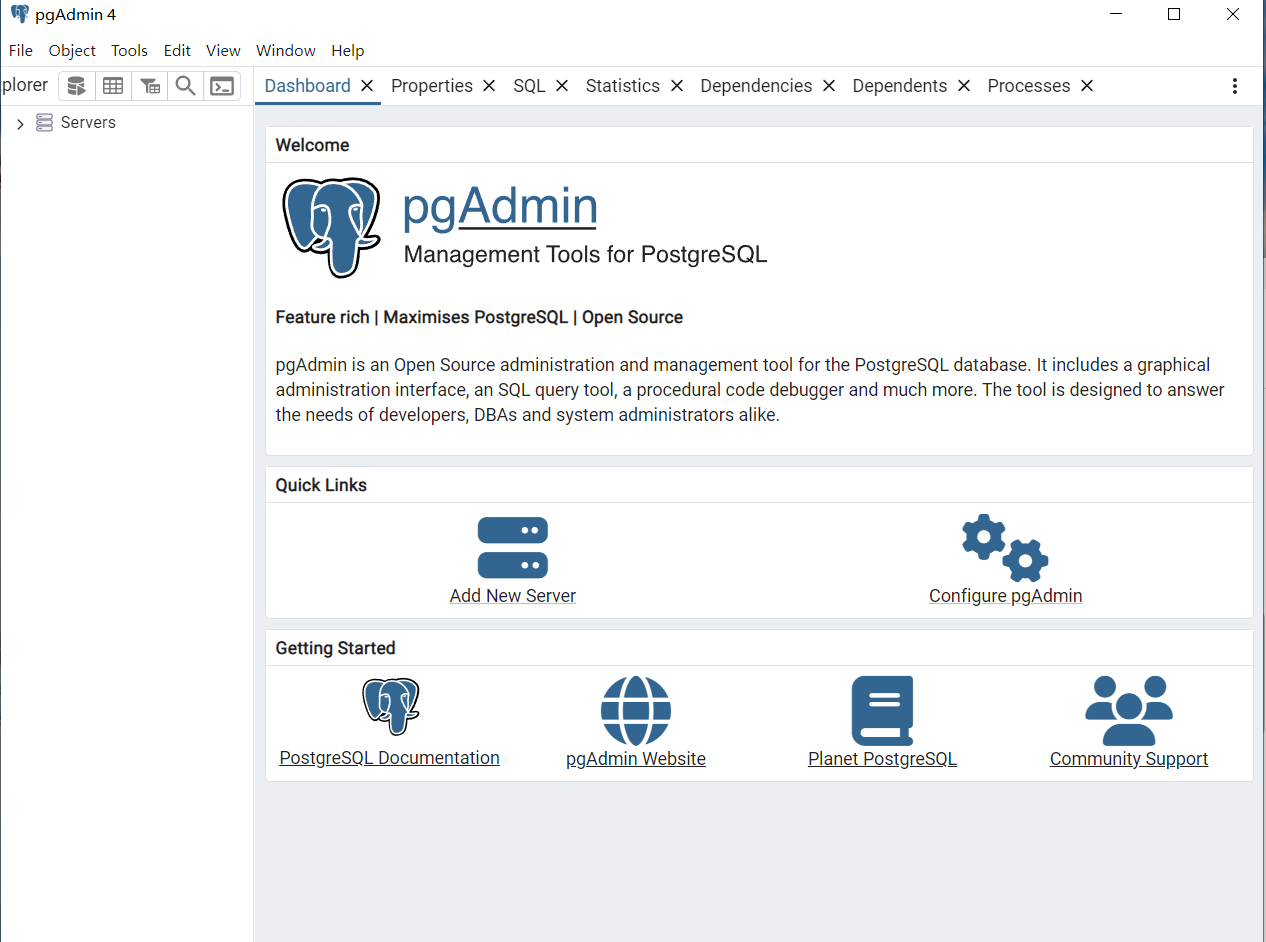


图1-8 成功下载并运行pgAdmin4

2）在pgAdmin4连接配置界面中，创建postgres数据库的连接conn\_本人学号。

按照Service\Register\Server的顺序，进入导数据库创建界面，然后输入name，ip，username，password等信息，然后save创建数据库。

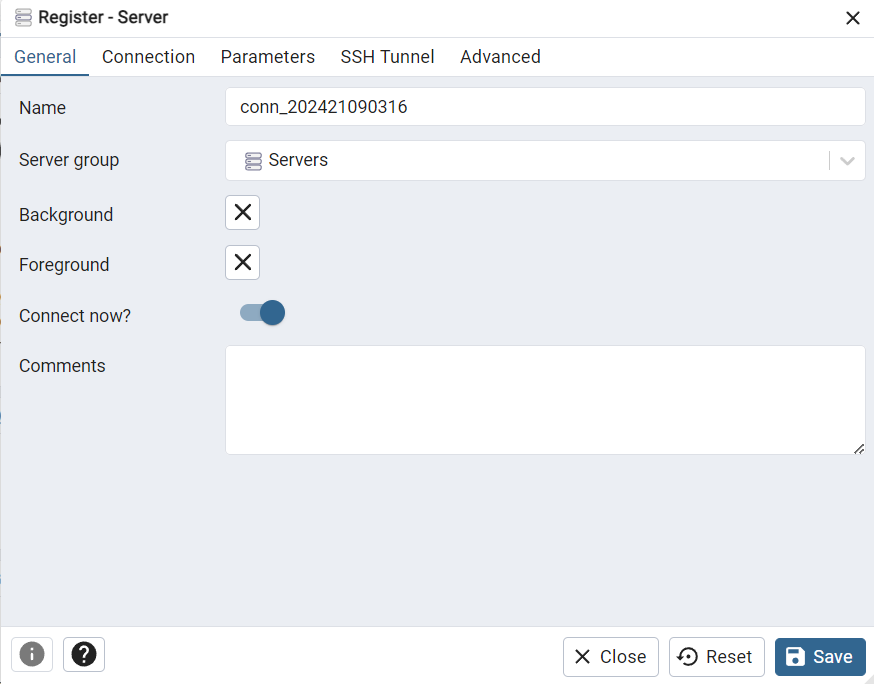


图1-9 创建数据库conn\_202421090316

但是点击save时，出现报错：connection timeout expired。

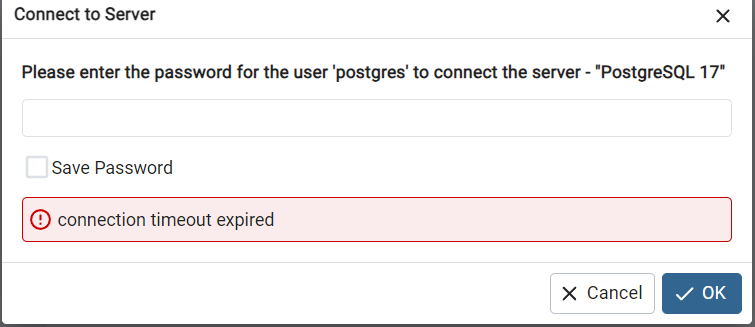


图1-10 创建数据库conn\_202421090316连接超时

在CSDN上查阅相关资料，在系统服务中手动启动服务postgresql-x64-17

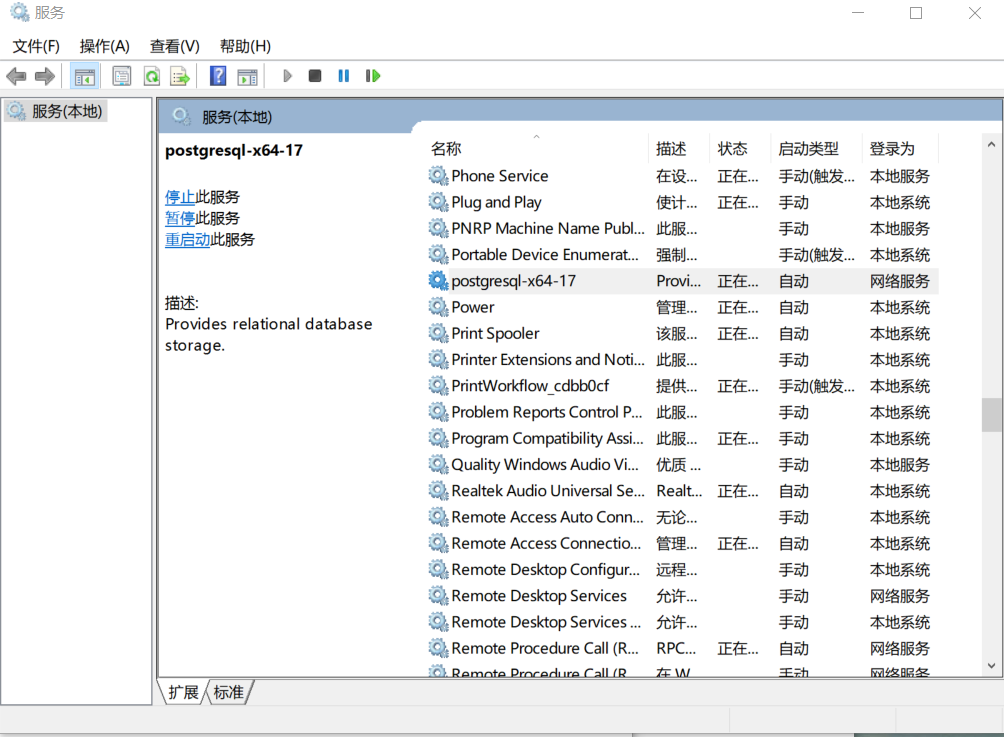


图1-11 手动启动服务postgresql-x64-17

最后再次创建数据库conn\_202421090316，数据库创建成功。

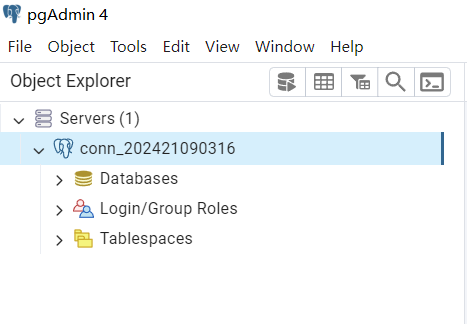


图1-12 数据库conn\_202421090316创建成功

3）在pgAdmin4管理工具中，使用“conn\_本人学号”登录postgres数据库，在仪表盘页面查看数据库运行状态，并进行基本监控管理。

在2）中已经完成了conn\_202421090316登录postgres数据库，接下来在dashboard中activity中查看数据库的运行状态，按照左到右，上到下的顺序，仪表盘依次显示的内容是活动：显示当前数据库的活跃/等待的查询数；事务/秒：显示每秒完成的事务数量；元组返回/秒和元组获取/秒：显示每秒从磁盘读取和返回给[客户端](https://www.zhihu.com/search?q=%E5%AE%A2%E6%88%B7%E7%AB%AF&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22article%22%2C%22sourceId%22%3A644955668%7D)的数据行数；块读取/秒和块命中/秒：显示块的读取和命中率，这有助于了解缓存效率。

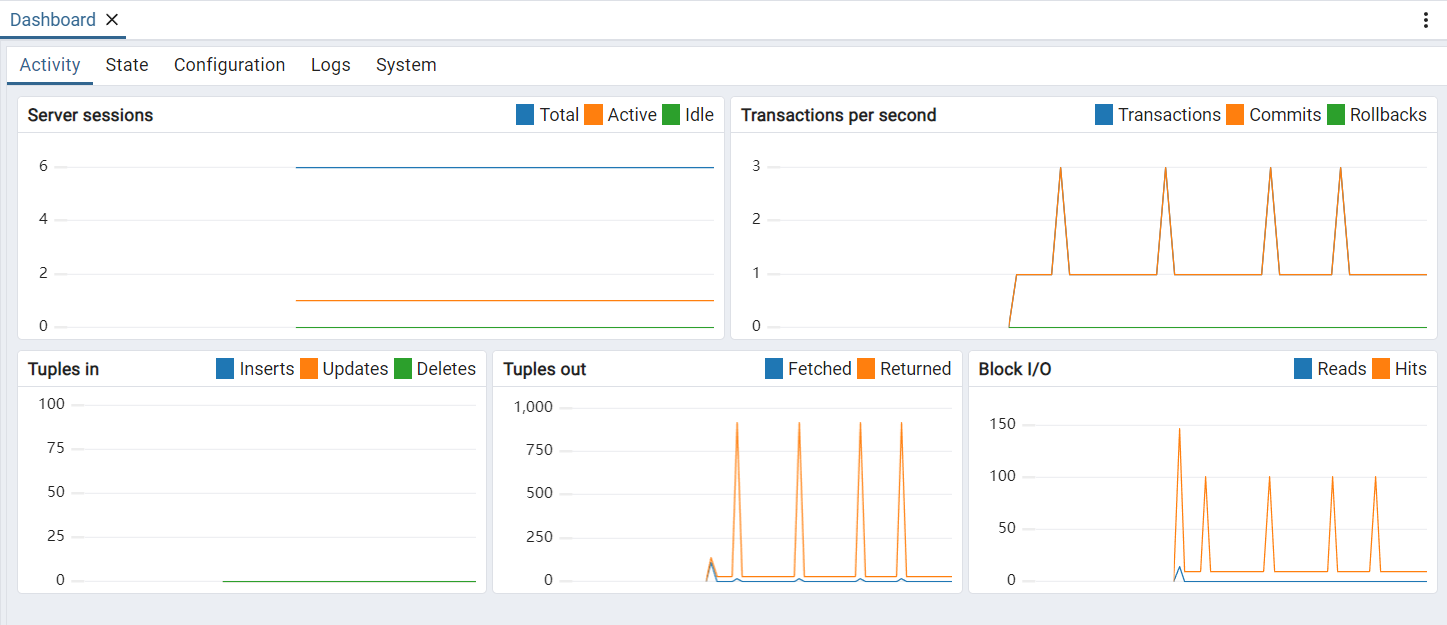


图1-13 仪表盘查看数据库运行状态

在dashboard的state中，可以进行基本的监控管理。Locks中显示当前数据库中的锁定情况，以便你监控并防止死锁。Session中显示当前连接到数据库的具体进程情况以及后台程序的创建时间。

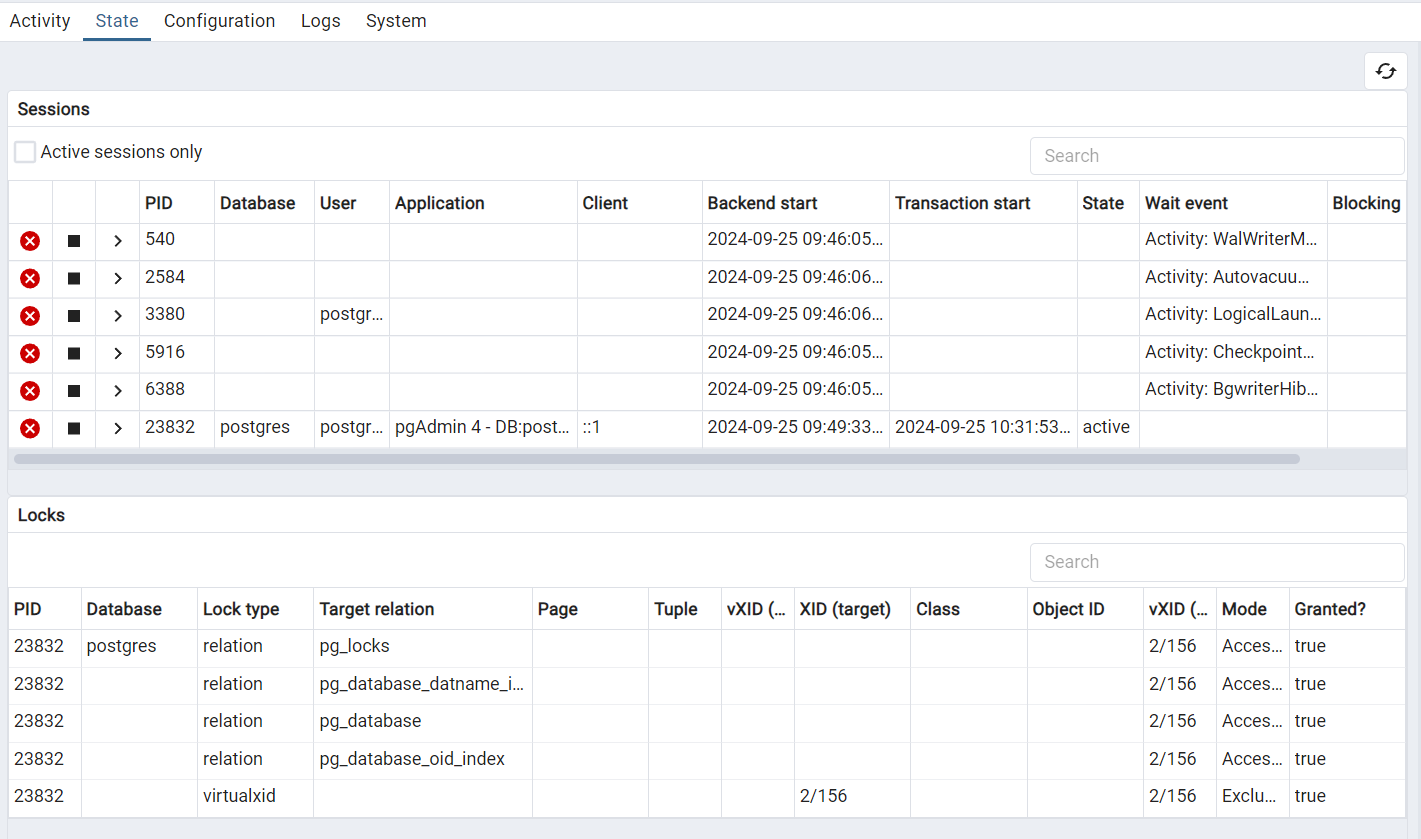


图1-14 dashboard state中进行监控管理

二、SQL引擎的查询处理与优化技术研究

1.SQL查询处理流程分析

首先从官网下载postgreSQL软件源码包，解压后，目录如图所示：

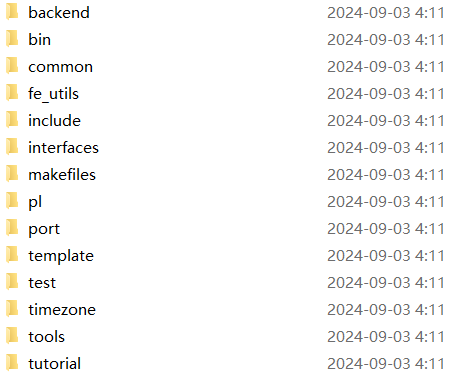


图2-1 postgreSQL源码目录

下面结合postgreSQL查询处理流程图和源代码目录，对各个功能模块对应的源码目录路径以及SQL查询处理流程进行说明。

第一步是客户端请求的接收与调度。在用户发送了SQL命令后，请求首先被 Postmaster（守护进程）接收，它负责管理客户端连接并启动一个新的 Postgres 后端进程来处理请求，该部分的源码目录是：src/backend/postmaster/。Postmaster 进程根据客户端的请求启动新的 Postgres 后端进程，分配一个独立的后端进程处理该客户端的所有请求，直到连接断开。该部分的源码目录是src/backend/postmaster/postmaster.c。

第二步是查询处理的具体流程。首先是查询分析部分，解析器（Parser）模块负责将 SQL 语句转换为语法树，其中包括选择的字段、表名等信息。该部分对应的源码路径是src/backend/parser。接着就是调度，对于那些功能性的命令，即非查询命令，如creat table等，在功能性命令处理中完成，而select，update等就进入导查询重写。模块将语法树转换为查询树，应用规则（如视图展开、规则触发）来重写查询，该部分的源码路径是src/backend/rewrite。接着要查询进行优化，提高运行效率。优化器会将查询树转换为查询计划，生成最优的执行路径（包括连接顺序、索引使用等），该部分的源码路径是src/backend/optimizer。最后就是查询执行，执行器根据优化器生成的计划树，逐步执行操作，如顺序扫描表、索引查找、结果排序等，该部分的源码路径是src/backend/executor。

第三步是查询结果返回。执行器完成查询后，将结果返回给客户端。客户端可以通过接口（如 libpq）接收并展示结果，该部分的源码路径是src/interfaces/libpq（客户端通信）。

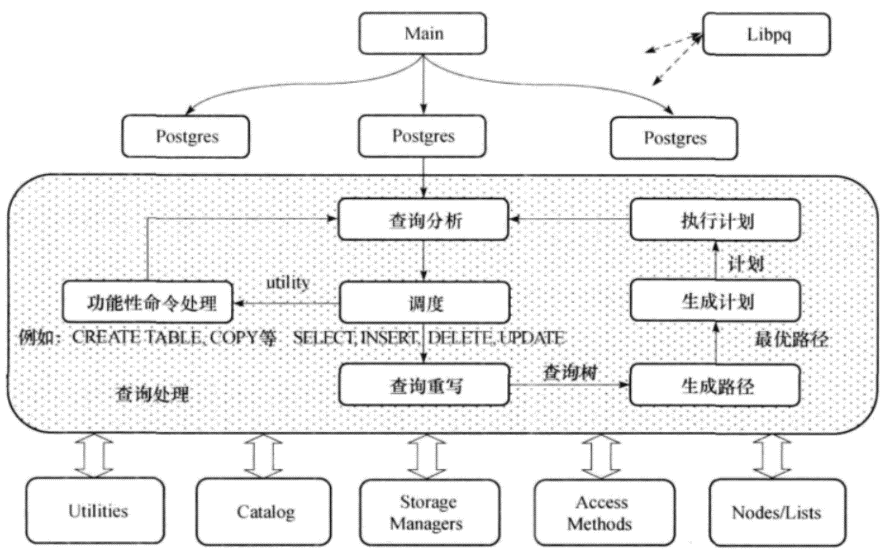


图2-2 SQL查询处理流程

2.SQL查询优化

首先是创建新的数据库ProjectDB\_202421090316，成功创建。

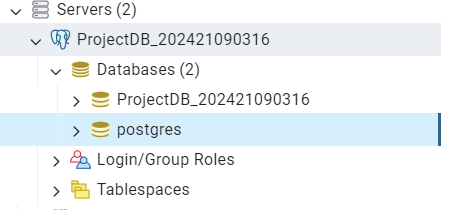


图2-3 创建数据库ProjectDB\_202421090316

接着根据表结构定义，依次创建雇员表（employee）、部门表（department）、项目表（Project）、任务表（assignment）。

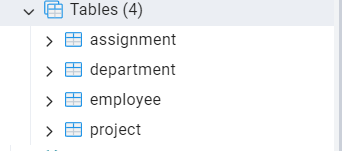


图2-3 成功创建要求的4张表

接着是往各个表中插入一定数量的数据，这里根据样例数据，然后进行一定修改，生成新的数据，然后插入表中，以下是插入数据后表的具体情况。

首先是雇员表（employee），往其中插入了10条数据。

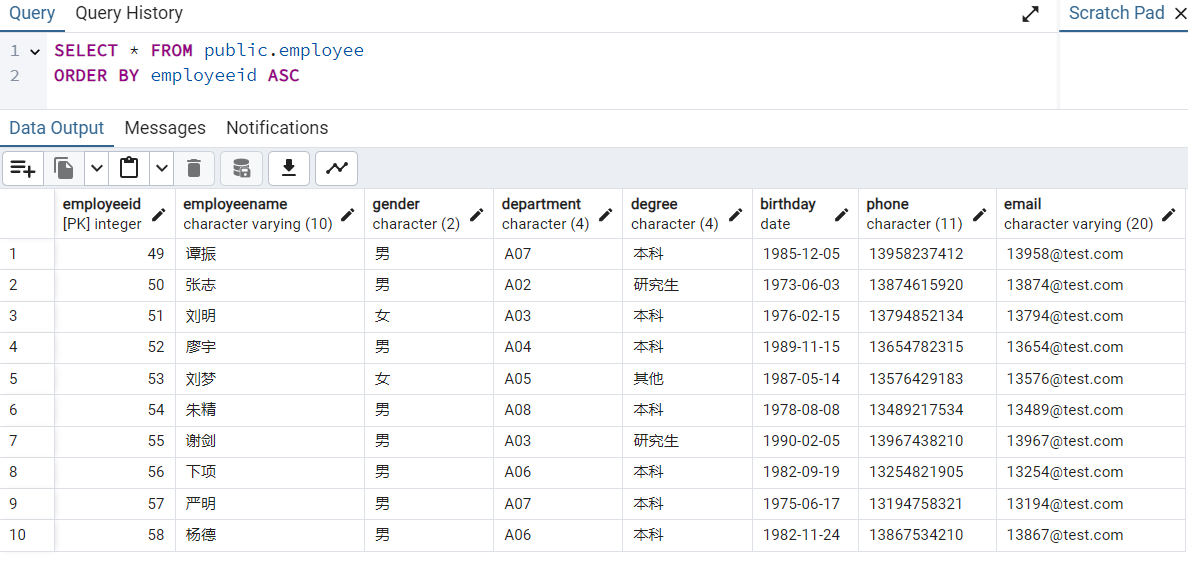


图2-4 雇员表（employee）中插入数据情况

接着是部门表（department），往其中插入了10条数据。

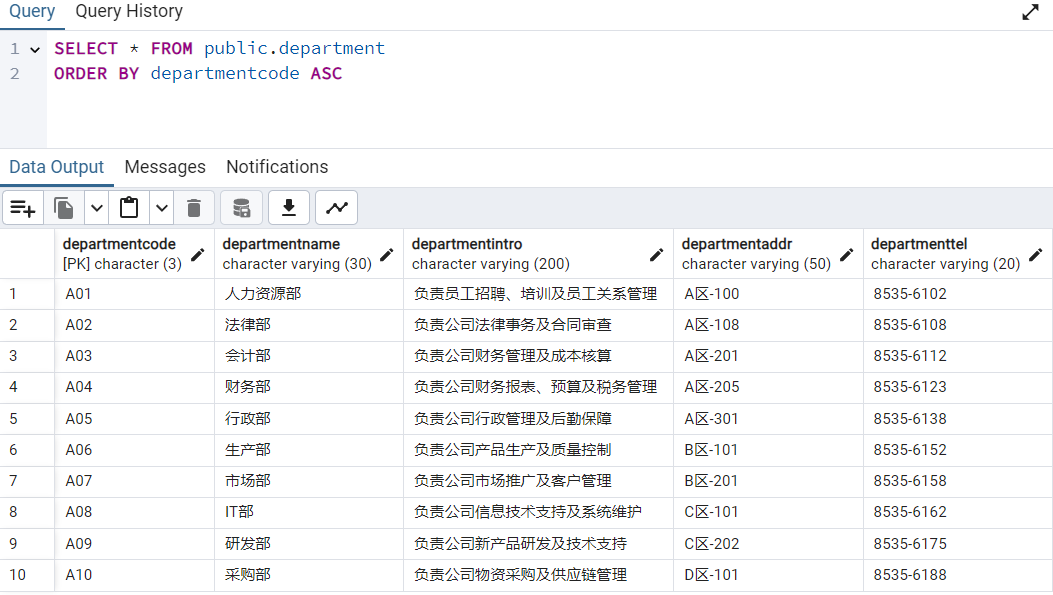


图2-4 部门表（department）中插入数据情况

然后是项目表（Project），往其中插入了10条数据。

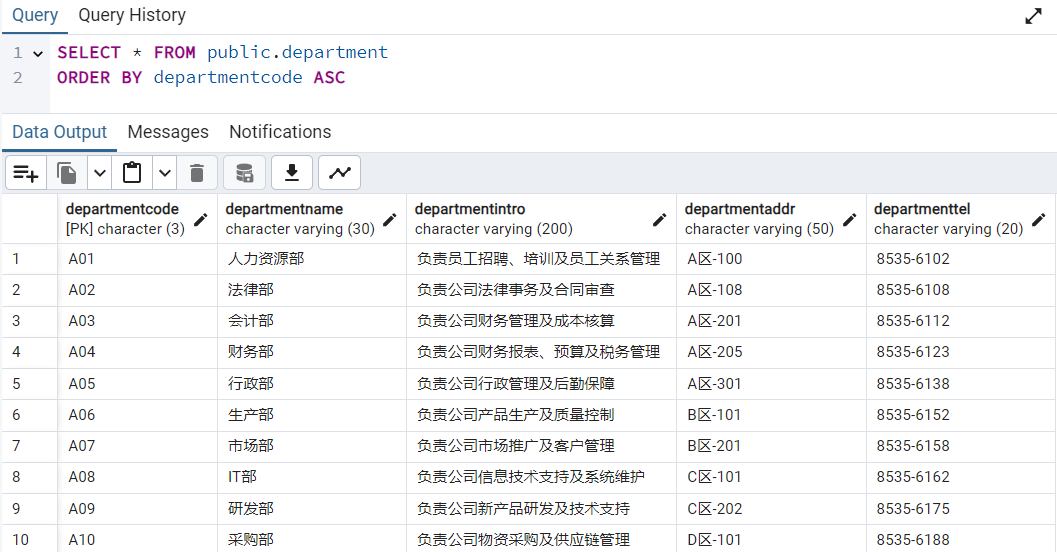


图2-5 项目表（Project）中插入数据情况

最后是任务表（Assignment），其中要考虑到跟项目表和雇员表的主外键关系。

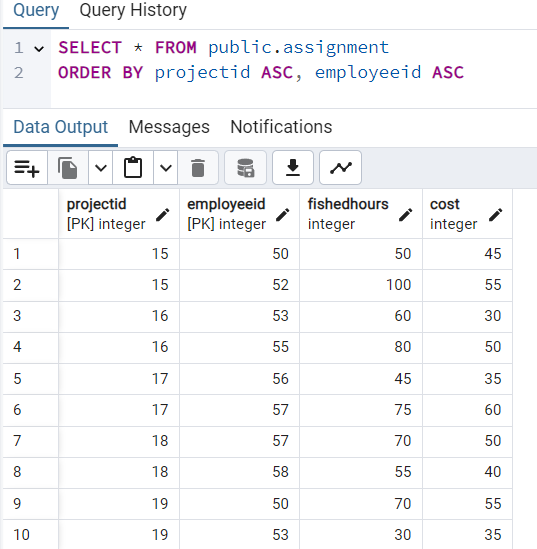


图2-6 任务表（Assignment）中插入数据情况

要对SQL查询语句进行性能的测试和判断，首先要明确测试的方法。通过查阅官方文档，这里采用EXPLAIN命令，包含ANALYZE，COST等option，从而多样化的查看SQL查询语句的执行性能。

明确方法以后，接下来思考SQL语句优化的方案。第一个优化是思路就是创建index。EmployeeID和ProjectID分别是employee表和project表中的主键，会自动分配index，但它们刚好是assignment表中的外键，外键不会自动创建索引，因此这里可以对assignment表中的这两个属性创建索引；第二个优化思路是优化SQL查询语句的表述。WHERE E.EmployeeID =A.EmployeeID AND P.ProjectID=A.ProjectID 本质是隐式JOIN语法，即将 FROM 子句中的多个表直接列出，并在 WHERE 子句中添加连接条件。这样带来的缺点是代码可读性差，不易于维护。在此改为显示JOIN语法，更易于优化器理解，修改后的SQL查询语句如下：

代码2-1 优化后的相关SQL查询语句

|  |
| --- |
| -- 为 Assignment 表的 EmployeeID 创建索引  CREATE INDEX idx\_assignment\_employeeid ON Assignment (EmployeeID);  -- 为 Assignment 表的 ProjectID 创建索引  CREATE INDEX idx\_assignment\_projectid ON Assignment (ProjectID);  EXPLAIN ANALYZE  SELECT  P.ProjectName AS 项目名称，  E.EmployeeName AS 员工姓名,  A.FinishedHours AS 实际工时  FROM  Assignment AS A  JOIN  Employee AS E ON A.EmployeeID = E.EmployeeID  JOIN  Project AS P ON A.ProjectID = P.ProjectID  ORDER BY  P.ProjectID, A.EmployeeID; |

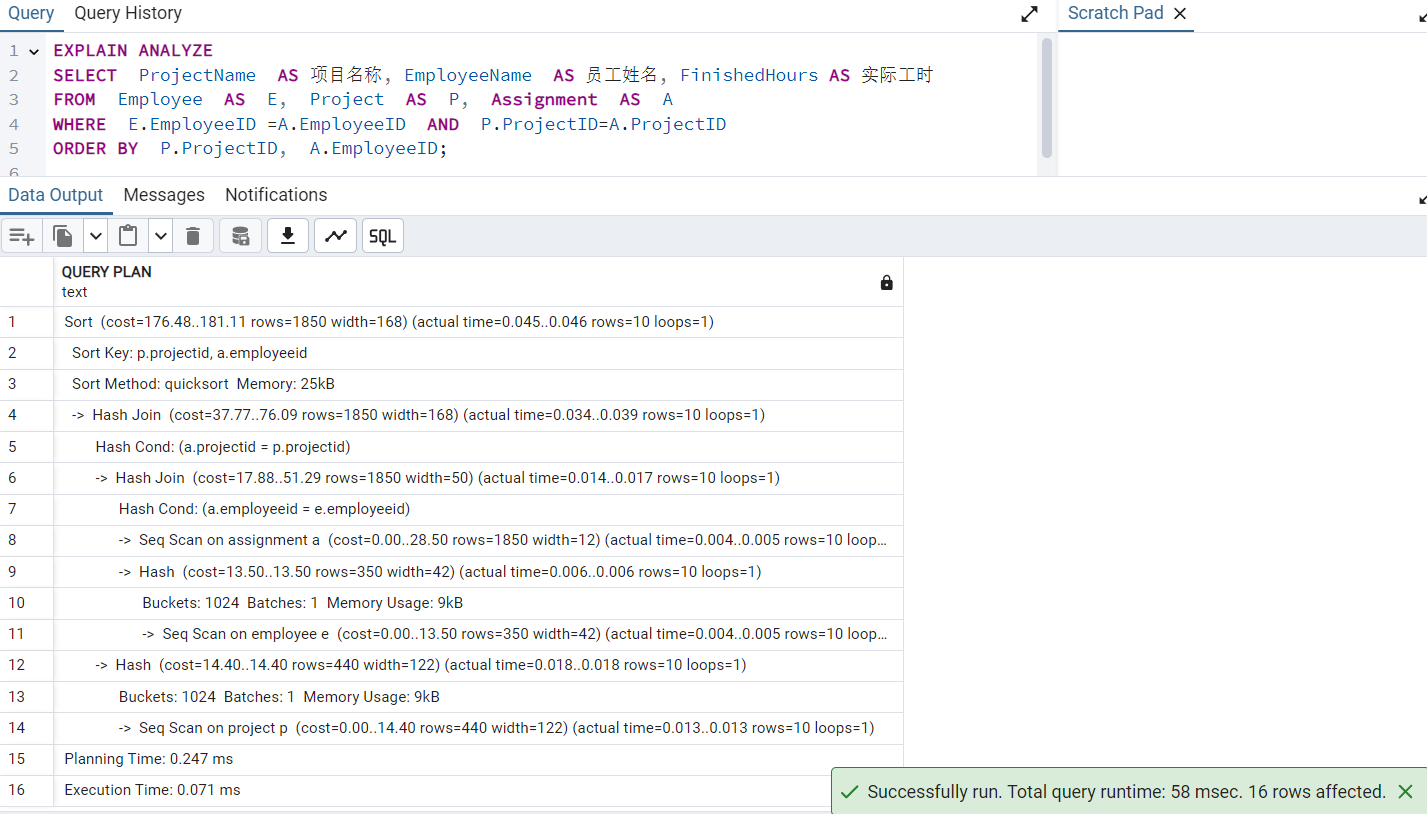
最后分别执行优化前后的SQL查询语句，并通过EXPLAIN命令进行性能分析。通过查看execution time，发现时间反而变长了，这应该是由于数据量过少（10条），顺序查询反而比带索引的查询更快，为了验证猜想，分别往employee，project，assignment表中插入2000条数据。

图2-7 优化前SQL查询语句执行情况

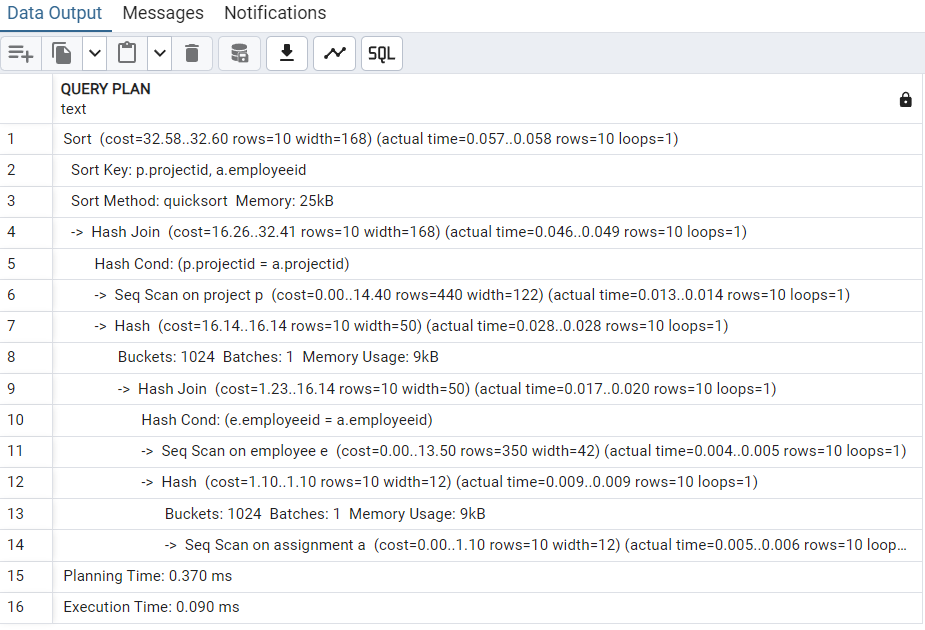


图2-8 优化后SQL查询语句执行情况

在往employee，project，assignment表中重新插入数据时，考虑到主键冲突问题，在此删除原有的数据，然后分别定义和调用函数generate\_employee\_data();generate\_project\_data();generate\_assignment\_data();从而完成数据的重新插入，下面以employee表为例，展示插入后表的数据情况。

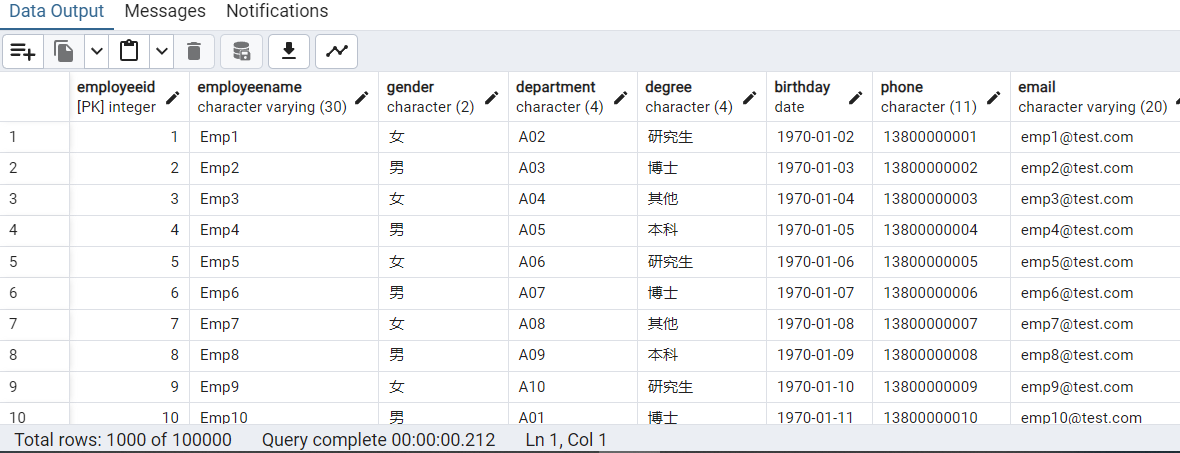


图2-9 employee表中的100000条新数据

往表中分别插入了100000条数据，再次对比优化前后的性能。可以看到，即使是100000条数据，查询速度仍然非常快，但已经可以看到，优化后的性能对比优化前的性能，提升了接近15%，证明索引的增加确实起了效果。

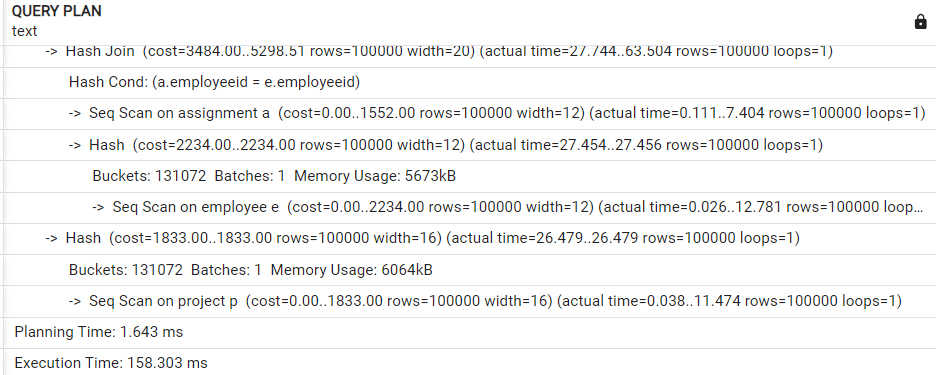


图2-10 优化前查询效果（100000条）



图2-11 优化后查询效果（100000条）

三、postgreSQL数据库存储管理技术研究

1.逻辑存储结构

逻辑存储管理实现机理：

 主要从三个方面实现逻辑存储管理。

第一是系统目录与系统表：PostgreSQL 使用系统表（如 pg\_class、pg\_database、pg\_tablespace）管理所有数据库对象的元数据，包括表、索引、视图等。每个数据库簇都有一个全局系统目录（如 pg\_global），用于存储全局对象（如角色、权限）。

 第二是对象 ID（OID）与映射机制：PostgreSQL 中的每个对象（数据库、表、索引等）都有一个唯一的 OID，用于在系统表中进行标识和引用。OID 用于在数据库簇的不同层次间进行数据组织和引用。

 第三是多版本并发控制（MVCC）：PostgreSQL 使用 MVCC 来管理元组的并发访问，避免写操作锁定整个表。元组的 xmin 和 xmax 字段用于跟踪元组的创建和删除状态，配合事务 ID 实现数据隔离

数据组织模式：

在 PostgreSQL 数据库的逻辑存储管理中，各个层次的对象之间有着明确的组织关系和管理机制。概括的说来，数据库簇（Cluster） 包含多个 表空间（Tablespace）。表空间（Tablespace） 包含多个 数据库（Database）。数据库（Database） 包含多个 模式（Schema）。模式（Schema） 包含多个表（Table）。表（Table） 包含多个 元组（Tuple），元组是表中数据的实际存储单元。

Hash索引的组织结构：

Hash 索引使用哈希函数将键值映射到桶（Bucket）中，桶作用过程指的是Hash 索引将键值通过哈希函数映射到不同的桶中，每个桶中存储了与该哈希值对应的数据行的指针。当一个桶中的数据量超过其容量时，会创建溢出页来存储额外的数据。

Hash索引适用于等值查询多的情况，尤其是数据分布均匀时。但其不支持范围查询（如 <、>）。并且在数据倾斜或分布不均时性能较差。

研究postgreSQL数据库逻辑存储管理实现机理，分析数据库簇、表空间、数据库、模式、数据库表、元组之间的数据组织模式，并说明postgreSQL数据库。

2.物理存储结构

PostgreSQL 的物理存储结构主要由数据库簇的根目录及其子目录组成，数据库簇根目录如下图所示：

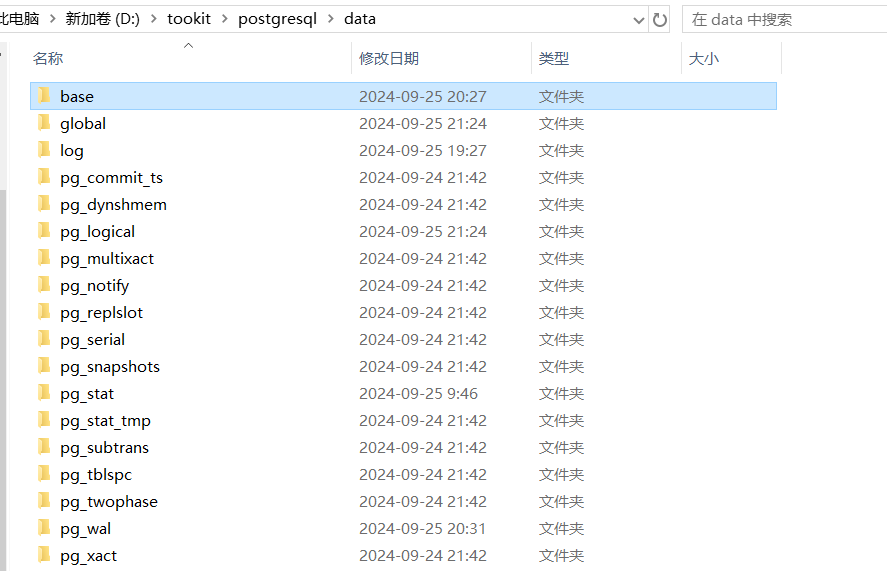


图3-1 数据库簇根目录

下面对数据库簇根目录的一些文件夹功能进行说明：base 目录存储默认表空间（pg\_default）中的所有数据库的物理文件.。global 目录中保存了数据库簇的元数据和全局系统表的数据文件。存储全局数据和系统表信息，如角色和权限。pg\_xlog（或 pg\_wal）目录中包含多个日志文件，记录数据库的所有变更操作。pg\_tblspc存储表空间的符号链接，指向实际的表空间目录。Pg\_stat\_temp存储临时统计信息，如数据库连接数、查询次数等。

数据库对象数据的组织以及命名：

同样的，对一些重要的数据库对象进行说明。表（Table）的数据存储在 base 目录下的数据库目录中，文件名为表的 OID，命名方式是base/数据库OID/表OID。索引文件也存储在 base 目录下的数据库目录中，文件名为索引的 OID。索引（index）的数据页存储在这些物理文件中，命名方式是base/数据库OID/索引OID。序列存储在与表相同的目录中，文件名为序列的 OID。序列（Sequence）用于生成唯一的递增或递减的数值，命名方式是base/数据库OID/序列OID。视图（View）的定义存储在 pg\_views 系统表中，其本身没有独立的物理存储文件。模式（Schema）本身是一个逻辑对象，用于组织数据库中的表、视图、函数等对象。它没有独立的物理存储文件，其定义信息存储在 pg\_namespace 系统表中。触发器（Trigger）的定义信息存储在 pg\_trigger 系统表中，它没有独立的物理存储文件。空间是物理存储位置的逻辑表示，用于指定数据库对象的存储目录。表空间（Tablespace）的目录结构存储在 pg\_tblspc 目录下，目录中存放符号链接指向实际的表空间位置。

四、postgreSQL事务处理与并发控制管理技术研究

1.在并发事务处理中，为什么使用多版本并发控制（MVCC）方法进行数据库并发访问比使用加锁机制访问具有更好的性能？

MVCC和加锁机制相比，主要有以下6点优势。

1避免了读写冲突（减少锁竞争）

加锁机制：在传统的加锁机制中，读取和写入操作会互相阻塞。一个事务在读取某行数据时，其他事务无法对该行数据进行更新操作，因为更新操作需要获取排它锁。

MVCC：通过维护多个版本，读取操作可以读取之前的版本，而写入操作可以生成新的版本而不影响当前读取。这意味着读写操作不会互相阻塞，从而大大提高了并发性能。

示例：事务 A 读取 employee 表中某行数据，事务 B 同时可以更新该行数据而不会阻塞事务 A。事务 A 看到的是更新前的版本，而事务 B 创建了一个新的版本。

2. 提高了读操作的并发性

加锁机制：多个事务同时读取同一行数据时，必须获取共享锁。尽管共享锁之间不互相阻塞，但仍存在锁的管理开销，并且在写操作到来时，所有读取操作都会被阻塞。

MVCC：由于每个事务看到的数据版本是相对独立的，读操作之间没有锁竞争。这使得读操作之间的并发性能大幅提升。

示例：事务 A 和事务 B 可以同时读取 employee 表中同一行数据，而不需要等待对方释放锁。

3. 减少了锁的管理开销

加锁机制：在高并发情况下，锁的管理和调度会带来额外的开销，包括锁的分配、释放、锁表的维护等。这些开销随着事务数量的增加而成倍增加。

MVCC：通过版本控制来实现并发访问，而不依赖锁来管理数据访问。因此，避免了锁的管理开销，大大降低了系统开销。

示例：在一个具有高并发读写操作的系统中，锁的管理开销可能会导致性能瓶颈，而 MVCC 的多版本存储可以在不加锁的情况下提高并发性能。

4. 提升写操作的吞吐量

加锁机制：写操作需要获取排它锁，当多个写操作同时修改同一行数据时，必须等待前一个事务完成才能继续，从而大大降低了写操作的吞吐量。

MVCC：写操作只需要在数据行上生成新的版本，不会阻塞其他事务的写操作。因此，即使在高并发写操作的场景下，MVCC 也能保持较高的吞吐量。

示例：事务 A 和事务 B 同时更新 employee 表中的不同行，两个写操作可以并发执行，而不会因为锁等待而阻塞。

5. 避免了死锁

加锁机制：多个事务在互相等待对方释放锁时，可能会导致死锁。例如，事务 A 锁住了行 1，事务 B 锁住了行 2，然后两者都尝试获取对方已锁住的行，导致死锁。

MVCC：由于 MVCC 通过版本控制来管理并发访问，而不是直接锁住数据行，因此避免了常见的死锁问题。

示例：在读写并发场景中，由于读写操作互不阻塞，因此不会出现死锁问题。

6. 快照隔离（Snapshot Isolation）

MVCC：每个事务在开始时都会拍摄数据库的快照，基于快照读取数据。这种机制避免了“不可重复读”和“幻读”问题，为事务提供了较高的隔离级别。

加锁机制：传统加锁机制难以同时解决“不可重复读”和“幻读”问题，需要复杂的锁策略，而这又会引入额外的性能开销。

示例：事务 A 在开始时拍摄 employee 表的快照，不会因为事务 B 的写入操作而导致读取数据的不一致。

2.研究postgreSQL多版本并发控制（MVCC）的实现机理，给出相应技术实现说明。

MVCC的原理主要是三点：数据版本管理，每个版本对应一个事务的修改；数据行状态判断，包括数据行的可见性规则和数据行的更新与删除；事务快照管理，包括事务快照和事务隔离。

具体的实现机制主要是5点。

1数据行的物理结构设置：数据行由元组头和数据体组成，前者包含元组的元数据信息，如 xmin、xmax、ctid、t\_xmin、t\_xmax、t\_cmin、t\_cmax 等字段，用于表示元组的事务状态、行位置等信息，后者主要是实际的数据值部分。

2数据页的物理结构设置：每个表的存储文件由多个数据页（Page）组成，每个数据页的大小通常为 8KB。每个页的起始位置有一个页头，记录该页的元数据，如数据行的数量、空闲空间位置等。

3事务管理与控制文件机制：PostgreSQL 使用 pg\_xact 系统目录中的事务日志来管理事务的提交和回滚状态。每个事务在开始时分配一个 XID，并在 pg\_clog 中记录事务的状态（已提交、已回滚、未决等）。通过pg\_control控制文件，其中包含数据库的元数据，如 XID 的最大值、数据库状态、检查点信息等。

4. 自动真空（Autovacuum）机制：MVCC 会产生大量的旧版本数据（Dead Tuples），这些旧数据会占用存储空间，影响查询性能。PostgreSQL 使用自动真空机制来清理这些旧版本数据。自动真空会标记和回收不可见的旧数据行，从而释放存储空间，并维护表的索引和统计信息。

5. 检查点（Checkpoint）与日志管理：检查点会将缓冲区中的脏数据（Dirty Data）写入磁盘，并更新控制文件（pg\_control）。有助于减少故障恢复时的日志重放时间。至于日志管理方面，WAL 记录了所有事务的变更操作，并在实际数据文件写入前将日志写入磁盘。即使系统崩溃，通过 WAL 日志也能恢复数据。