北京科技大学计算机科学与技术系

阅读 PostgreSQL 数据库源代码指导

PostgreSQL 数据库体系结构部分

《数据库系统原理与实现》课程教研组 曾庆峰

2024-5-1

目录

1 快速走进 PostgreSQL 数据库世界

PostgreSQL 是一种开源的数据库管理系统（DBMS）。PostgreSQL 数据库管理系统可以被简称为“**PostgreSQL 数据库服务器**”。在上下文明确、不影响读者理解的情况下，还可以将其简称为“**PostgreSQL 服务器**”，或者简称为“**PostgreSQL 数据库**”。

PostgreSQL 数据库管理系统由以下两部分组成：

- PostgreSQL 二进制程序。
- PostgreSQL 数据库集簇（数据）。

1.1 基本概念

1.1.1 数据、信息、元数据

数据（Data）是指可以记录和保存在计算机存储介质上的、关于对象和事件的事实。数据是在用户环境中，具有意义和重要性的对象或事件的存储表示。

通常有两种管理数据的方法：**数据库方法**、**文件系统方法**。这两种管理数据的方法各有优缺点。

信息（Information）是人们按照知识的方式处理后得到的数据。

例：室外温度是-3℃

- 温度是-3℃（数据）
- 温度是-3℃，根据人类的常识，天很冷（信息）
- 现在出门得多穿点衣服（信息的利用）

例：一组数据：

张三　　13301215503
李四　　13401215454
王五　　13501215505
赵六　　13601215836

对于这组数据有多种解释：

- 人名及其电话号码
- 人们及其社保号码

也就是说，不知道这些数据条目的实际意义，人们无法使用这组数据！

要把这组数据转变为信息，需要为其附加一些数据项、并提供一个数据结构：

课程表

姓名	学号	专业	绩点
张三	13301215503	计算机	3.8
李四	13401215454	机械	3.9
王五	13501215505	管理	4.4
赵六	13601215836	文法	4.2

也就是说，需要为数据添加“元数据”后，我们才知道，这组数据表示的是学生的姓名和学号。**元数据（Meta Data）**是数据的数据，用于描述数据的特征、性质以及数据的内容。

描述数据特征的元数据可以是：

- 数据名称
- 数据定义
- 数据长度(或大小)
- 数据可能的取值

描述数据上下文的元数据可以是：

- 数据的来源
- 数据的存储位置
- 数据的拥有者
- 数据的使用方式

元数据是从数据中得出的数据。也就是说，元数据描述数据的性质，但是与数据独立。基于元数据，用户可以理解存在哪些数据、数据的意义以及如何区分那些看起来很类似的数据项之间的不同。元数据管理与数据管理一样重要，因为数据如果没有清晰的语义将毫无用处。可以像检索数据或信息一样，检索元数据。

1.1.2 数据库集簇、数据库、模式与模式对象

数据库（Database）是长期存储在计算机上、有组织、可共享的大量数据的集合。数据库包含了某个组织机构的信息。例如，我们的大学应用数据库 universitydb，以关系模型的方式组织在一起，包含了管理一个大学所需的全部数据。

像大学这样的组织机构，由于部门繁多，因此我们想在大学应用数据库 universitydb 中，按部门来组织数据库中的数据。为了实现这个目的，我们引入“模式”这一概念。**模式（Schema）**按照一定的分类方法，将数据库中的数据进行逻辑分组。也就是说，一个数据库可以包含多个模式，但是一个模式只能属于一个数据库。

模式中的数据对象，被称为**模式对象（Schema Object）**。在关系数据库中，模式对象可以是关系表（Relation Table）、视图（View）、系列（Sequence）、索引（Index）、函数（Function）与存储过程（Stored Procedure）、触发器（Triggers）、……。

模式也被称为**命名空间（Name Space）**。在一个数据库的不同模式中，可以有同名的模式对象！实际上，我们可以使用以下的方法，来确定唯一的模式对象：

数据库名. 模式名. 模式对象名

例如：

universitydb. public.person

universitydb. academicoffice.person

universitydb. humanresources.person

是 3 个不同的关系表。

1.1.3 数据库管理系统

数据库管理系统（DataBase Management System，DBMS）由一个相互关联的数据的集合和访问这些数据的程序组成。它是位于用户和操作系统之间的一层数据管理软件。显然，作为基础设施软件，数据库管理系统（DBMS）是一个大型复杂的软件系统。

数据库管理系统（DBMS）的目标是要为人们提供一种方便、高效地存取数据库中信息的方法（这句话隐含 DBMS 包括数据+程序）。

数据库管理系统（DBMS）具有如下的功能：

- 定义信息的存储结构；
- 提供操作数据库中所存储的信息的机制；
- 保证所存储信息的安全，即使在系统崩溃或有人企图越权访问时，也应保障信息的安全性；
- 提供并发控制的机制，当数据被多用户共享时，避免可能产生的异常结果。

当前存在的数据库管理系统，有如下的类别：

- 关系型数据库管理系统，如 Oracle、MS SQL Server、MySQL 和 PostgreSQL。

- NoSQL，如 Redis（KV 键值对类型数据库）、Hbase（列族类型数据库）、MongoDB（文档类型）、Neo4j（图类型数据库）。
- 分布式 NewSQL，如 Oceanbase、TiDB；

PostgreSQL 是一种开源的数据库管理系统。PostgreSQL 数据库管理系统可以被简称为 PostgreSQL 数据库服务器。在上下文明确的情况下，还可以被简称为 PostgreSQL 服务器、PostgreSQL 数据库。

请读者区别 5.1.3 小节的术语“数据库”与本小节的术语“PostgreSQL 数据库”。

1.1.4 数据库系统

数据库系统（DBS）是引入数据库管理系统（DBMS）后的计算机系统。

1.2 编译安装 PostgreSQL 数据库服务器

准备一台机器名为 dbsvr，IP 地址为 192.168.100.31/24，安装了 openEuler 22.03 操作系统的 Linux 服务器。

可以使用本书提供的 Vmware 虚拟机资源文件 Pgserver-1OS，作为实验环境。

1.2.1 安装前的准备工作

（1） 安装必要的 openEuler 软件包

使用 root 用户，执行下面的命令，为编译安装 PostgreSQL 数据库管理系统，安装必要的操作系统软件包：

```
yum -y install readline readline-devel zlib zlib-devel bzip2 tar \
               gettext gettext-devel openssl openssl-devel pam pam-devel \
               libxml2 libxml2-devel libxslt libxslt-devel \
               perl perl-devel perl-ExtUtils* tcl-devel uuid-devel \
               python3-devel gcc gcc-c++ make \
               flex bison
```

readline (命令行编辑支持库)

zlib (数据压缩支持库)

flex (词法分析库)

bison (语法分析库)

（2） 创建用户和组

为了安装 PostgreSQL 数据库，需要创建一些操作系统用户和用户组。

使用 root 用户，执行下面的命令，创建用户组 dba 和用户 postgres，并将用户 postgres 的密码设置为“postgres123”：

```
groupadd dba -g 3000
useradd postgres -g 3000 -u 3000
id postgres
echo "postgres123"|passwd --stdin postgres
```

（3） 创建安装 PostgreSQL 数据库所需的目录

为了安装 PostgreSQL 数据库，需要创建一些目录。使用 root 用户，执行下面的命令，创建这些目录，然后修改这些目录的属主和权限：

```
# 源代码驻留目录
mkdir -p /opt/db/soft

# 修改目录的属主
chown -R postgres:dba  /opt/db
```

（4） 下载 PostgreSQL 源代码

可以使用浏览器打开网址：https://www.postgresql.org/ftp/source/v14.11/，在这个 URL 上下载文件 postgresql-14.11.tar.gz，这是在编写本书时，最新版本的 PostgreSQL 数据库源代码包。

执行下面的 wget 命令，下载最新的 PostgreSQL14 源代码：

```
[root@dbsvr ~]# su - postgres
（省略了一些输出）
[postgres@dvsvr ~]$ cd /opt/db/soft
[postgres@dvsvr soft]$
wget https://ftp.postgresql.org/pub/source/v14.11/postgresql-14.11.tar.gz
(省略了一些输出)
postgresql-14.11.tar.gz       100%[===================>]  27.88M  6.02MB/s    in 5.8s

2024-04-11 11:47:00 (4.78 MB/s) - ‘postgresql-14.11.tar.gz’ saved [29234901/29234901]
```

```
[postgres@dvsvr soft]$
```

下载完成后，执行下面的命令，解压缩 PostgreSQL 数据库源代码：

```
[postgres@dvsvr soft]$ tar -zxf postgresql-14.11.tar.gz
```

提示：PostgreSQL 数据库的源代码位于目录/opt/db/soft/postgresql-14.11 下

（5） 配置 postgres 用户的环境变量

在 openEuler 上，以 postgres 用户的身份，使用 vi 编辑器，编辑 postgres 用户的环境文件/home/postgres/.bash_profile：

```
[postgres@dbsvr ~]$ vi /home/postgres/.bash_profile
```

在文件的末尾，添加如下的行：

```
export PGPORT=5432
export PGUSER=postgres
export PGHOME=/opt/db/pgsql
export PGDATA=/opt/db/userdb/u00/pgdata
export PATH=$PGHOME/bin:$PATH
```

接下来以 postgres 用户的身份，执行如下的命令，启用这些环境变量：

```
[postgres@dvsvr ~]$ source ~/.bash_profile
[postgres@dbsvr ~]$ exit
logout
[root@dbsvr ~]#
```

（6） 修改 openEuler 内核参数

使用 root 用户，执行下面的脚本，修改 openEuler 的内核参数，以满足编译安装 PostgreSQL 数据库管理系统的要求：

```
cat>>/etc/sysctl.conf<<EOF
fs.file-max = 76724200
kernel.sem = 10000 10240000 10000 1024
kernel.shmmni = 4096
kernel.shmall = 253702
kernel.shmmax = 1039163392
net.ipv4.ip_local_port_range = 9000 65500
net.core.rmem_default = 262144
net.core.wmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_max = 1048576
fs.aio-max-nr = 40960000
vm.dirty_ratio=20
```

```
vm.dirty_background_ratio=3
vm.dirty_writeback_centisecs=100
vm.dirty_expire_centisecs=500
vm.swappiness=10
vm.min_free_kbytes=524288
vm.swappiness=0
vm.overcommit_memory=2
vm.overcommit_ratio=75
net.ipv4.ip_local_port_range = 10000 65535
EOF
```

（7） 修改用户的操作系统限制

使用 root 用户，执行下面的命令，配置用户堆栈限制、打开文件数限制、用户的最大进程数限制：

```
cat>>/etc/security/limits.conf<<EOF
postgres soft nofile 1048576
postgres hard nofile 1048576
postgres soft nproc 131072
postgres hard nproc 131072
postgres soft stack 10240
postgres hard stack 32768
postgres soft core 6291456
postgres hard core 6291456
EOF
```

（8） 修改 RemoveIPC 参数

systemd-logind 服务中引入了一个特性：当一个用户退出系统后，会删除所有与这个用户有关的 IPC 对象。该特性由/etc/systemd/logind.conf 文件中的 RemoveIPC 参数控制。在某些操作系统中会默认打开该参数，有可能会造成程序信号丢失的问题，因此我们需要关掉这个参数。

在 openEuler 上，以 root 用户的身份，编辑文件/etc/systemd/logind.conf：

```
[root@dbsvr ~]# vi /etc/systemd/logind.conf
```

将下面的行：

#RemoveIPC=no

修改为：

RemoveIPC=no

（9） 允许用户 postgres 执行 sudo

在 openEuler 上，以 root 用户的身份，编辑文件/etc/sudoers：

```
[root@dbsvr ~]# vi /etc/sudoers
```

找到如下的行：

```
## Next comes the main part: which users can run what software on
## which machines (the sudoers file can be shared between multiple
## systems).
## Syntax:
##
##      user    MACHINE=COMMANDS
##
## The COMMANDS section may have other options added to it.
##
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
```

在后面添加一行：

```
postgres      ALL=(ALL)       ALL
```

保存文件需要先按“ESC”键，然后输入字符串“:wq!”。

（10） 设置 openEuler 的时区（在中国不需要执行这一步）

使用 root 用户，执行下面的命令，查看 openEuler 系统当前的时区设置：

```
[root@dbsvr ~]# timedatectl status
               Local time: Sun 2021-12-26 16:38:29 CST
           Universal time: Sun 2021-12-26 08:38:29 UTC
                 RTC time: Sun 2021-12-26 08:38:28
                Time zone: Asia/Shanghai (CST, +0800)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
[root@dbsvr ~]#
```

如果你执行上面的命令后的输出，不同于上面的内容，那么可以用 root 用户，执行下面的命令，修改 openEuler 操作系统的时区：

```
timedatectl set-timezone Asia/Shanghai
```

（11） 打开防火墙端口 5432

默认情况下，PostgreSQL 数据库的客户端，将连接 PostgreSQL 数据库服务器的 TPC 端口 5432。

使用 root 用户，执行下面的命令，打开防火墙 firewalld.service 的数据库访问端口 5432：

```
[root@dbsvr ~]# firewall-cmd --zone=public --add-port=5432/tcp --permanent
[root@dbsvr ~]# firewall-cmd --reload
[root@dbsvr ~]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens33 ens34
  sources:
  services: dhcpv6-client mdns ssh
  ports: 5432/tcp
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
[root@dbsvr ~]#
```

可以看到，端口 5432/tcp 已经被允许通过防火墙了。

（12） 创建脚本命令 pgps

为了让读者方便地查看 PostgreSQL 数据库实例中的所有进程，请以 Linux 超级用户 root 的身份，使用 vi 编辑器，创建一个工具脚本命令文件/usr/bin/pgps，

```
[root@dbsvr ~]#  vi /usr/bin/pgps
```

内容如下：

```
#!/bin/bash
data_dir="/opt/db/userdb/u00/pgdata"
pid_file="$data_dir/postmaster.pid"

if [ ! -d "$data_dir" ]; then
        echo "No PostgreSQL Data Directory!"
elif [ ! -f "$pid_file" ]; then
        echo "PostgreSQL is not up!"
else
        ps -ef | grep $(head -1 $pid_file) | grep -v grep
fi
```

创建完文件/usr/bin/pgps 后，使用 root 用户，执行下面的命令，让脚本/usr/bin/pgps 拥有可执行权限：

```
[root@dbsvr ~]# chmod 755 /usr/bin/pgps
```

之后，只要 PostgreSQL 数据库正在服务器上运行时，就可以使用 postgres 用户，执行 pgps 命令，查看 PostgreSQL 数据库实例的所有的进程。

（13） 重启动 openEuler

使用 root 用户，执行 reboot 命令，重启动 openEuler 服务器：

```
[root@dbsvr ~]# reboot
```

1.2.2 配置编译 PostgreSQL 源代码

配置编译 PostgreSQL 源代码的目的是生成一个 makefile，用于编译 PostgreSQL 源代码。

使用 postgres 用户，执行下面的命令，配置 PostgreSQL 源代码：

```
[postgres@dvsvr ~]$ cd /opt/db/soft/postgresql-14.11
[postgres@dvsvr postgresql-14.11]$ ./configure CFLAGS="-O0 -ggdb"   CXXFLAGS="-O0" \
                                               --prefix=/opt/db/pg14 \
                                               --enable-debug  \
                                               --enable-cassert \
                                               --enable-depend \
                                               --enable-dtrace \
                                               --with-python \
                                               --with-openssl \
                                               --with-libxml \
                                               --with-libxslt \
                                               --with-pgport=5432
```

其中：

- CFLAGS="-O0"：指定 C 语言编译器的优化级别。
  - 优化级别 O0：编译器不会进行任何代码优化，目的是缩短编译时间并生成调试友好的代码。由于不进行优化，生成的代码会更加接近原始源码，这使得调试过程更为直接和简单，因为代码行与机器指令之间有直接的对应关系。副作用是编译生成的可执行文件会运行得更慢，不太适用于用户生产环境，仅仅用于调试分析 PostgreSQL 数据库源代码。
  - 优化级别 O2：编译时默认的优化级别，该优化级别生成的代码，其执行顺序会改变，这使得追踪 PostgreSQL 数据库的代码更为困难。适用于生产环境。
- CXXFLAGS="-O0"：指定 C++语言编译器的优化级别，选项值同 CFLAGS。

- 选项--prefix：指定 PostgreSQL 数据库编译后的安装位置。
- 选项--enable-debug：允许调试 PostgreSQL 数据库编译后生成的程序。
- 选项--enable-cassert：打开断言开关，这在开发过程中对于验证假设和捕获错误很有帮助。
- 选项--enable-depend：启用自动依赖性跟踪，这意味着在编译过程中，构建系统会自动生成文件之间的依赖关系，确保在后续的编译中，如果某个文件被修改，只有依赖于该文件的代码会被重新编译。这可以提高后续编译的效率，减少编译时间。
- 选项--enable-dtrace：支持动态跟踪工具 DTrace。
- 选项--enable-libxml：支持 XML。
- 选项--enable-libxslt：支持 XSLT，用于将 XML 文档转换为另一种格式，如 HTML、文本或其他 XML 格式。
- 选项--with-pgport=5432：定义访问 PostgreSQL 数据库的 TCP 端口号。

此外，还可以使用如下的开发者编译配置选项：

- 选项--enable-coverage：该选项仅支持 GCC。如果使用 GCC，则所有程序和库都将使用代码覆盖率测试工具进行编译。
- 选项--enable-profiling：该选项仅支持 GCC。如果使用 GCC，则会编译所有程序和库，以便对其进行概要分析。

1.2.3 编译 PostgreSQL 数据库源代码

使用 postgres 用户，执行下面的命令，编译 PostgreSQL 源代码：

```
[postgres@dvsvr postgresql-14.11]$ gmake world -j 4
```

其中：

- 运行 gmake world 会编译和安装 PostgreSQL 服务器、客户端工具、库文件以及附属的应用程序和接口。
- 选项-j 4：表示开启 4 个并行编译，这可以加速编译过程。

1.2.4 安装 PostgreSQL 数据库管理系统的二进制程序

使用 postgres 用户，执行下面的命令，将刚刚编译好的 PostgreSQL 数据库管理系统二

进制程序，安装到 openEuler 操作系统上：

```
[postgres@dvsvr postgresql-14.11]$ gmake install-world
```

提示：PostgreSQL 数据库管理系统的二进制程序被安装在目录/opt/db/pg14 下

执行下面的命令，创建符号连接/opt/db/pgsql -> /opt/db/pg14：

```
[postgres@dbsvr postgresql-14.11]$ cd /opt/db
[postgres@dbsvr db]$ ln -s /opt/db/pg14 /opt/db/pgsql
[postgres@dbsvr db]$ ls -l
total 8
drwxr-xr-x. 6 postgres dba 4096 Apr 14 11:49 pg14
lrwxrwxrwx. 1 postgres dba   12 Apr 14 11:49 pgsql -> /opt/db/pg14
drwxr-xr-x. 3 postgres dba 4096 Apr 14 11:43 soft
[postgres@dbsvr db]$
```

使用 postgres 用户，执行下面的 postgres 命令（启动 PostgreSQL 数据库服务器的命令），查看 PostgreSQL 服务器的版本信息：

```
[postgres@dbsvr db]$ postgres --version
postgres (PostgreSQL) 14.11
[postgres@dbsvr db]$ /opt/db/pg14/bin/postgres --version
postgres (PostgreSQL) 14.11
[postgres@dbsvr db]$ /opt/db/pgsql/bin/postgres --version
postgres (PostgreSQL) 14.11
[postgres@dbsvr db]$
```

1.2.5 打包 PostgreSQL 服务器二进制程序

执行下面的命令，将编译后的 PostgreSQL 服务器二进制程序打包为 tar 安装包：

```
[postgres@dvsvr db]$ ls -l
total 8
drwxr-xr-x. 6 postgres dba 4096 Apr 14 09:38 pg14
lrwxrwxrwx. 1 postgres dba   12 Apr 14 09:39 pgsql -> /opt/db/pg14
drwxr-xr-x. 3 postgres dba 4096 Apr 14 09:34 soft
[postgres@dvsvr db]$ tar cf postgresql-14.11.server.tar pg14/
[postgres@dbsvr db]$ ls -lh postgresql-14.11.server.tar
-rw-r--r--. 1 postgres dba 81M Apr 15 12:21 postgresql-14.11.server.tar
[postgres@dbsvr db]$
```

生成的 tar 包文件 postgresql-14.11.server.tar 可以作为 PostgreSQL 数据库服务器二进制发行版（Linux 平台，版本号为 14.11），用于在其他计算机上安装 PostgreSQL 数据库或客户端。

1.3 初始化 PostgreSQL 数据库集簇（initdb）

在 PostgreSQL 数据库服务器上，可以使用 PostgreSQL 数据库管理系统的二进制程序 initdb，创建一个 PostgreSQL 数据库集簇。

（1） 为初始化一个 PostgreSQL 数据库集簇创建必要的目录：

```
[postgres@dbsvr db]$  mkdir -p /opt/db/userdb/u00
[postgres@dbsvr db]$  mkdir -p /opt/db/userdb/u01
[postgres@dbsvr db]$  mkdir -p /opt/db/userdb/u02
```

（2） 使用 postgres 用户，执行下面的命令，初始化一个 PostgreSQL 数据库集簇：

```
[postgres@dbsvr ~]$/opt/db/pgsql/bin/initdb -D/opt/db/userdb/u00/pgdata -EUTF8 -Upostgres -W
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF-8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

Enter new superuser password: (输入密码“dba123”)
Enter it again:               (再次输入密码“dba123”)

creating directory /opt/db/userdb/u00/pgdata ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Asia/Shanghai
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    /opt/db/pgsql/bin/pg_ctl -D /opt/db/userdb/u00/pgdata -l logfile start

[postgres@dbsvr ~]$
```

其中：

- -D 参数指定 PostgreSQL 数据库的数据目录位置（/opt/db/userdb/u00/pgdata）。

- -U 参数指定要创建的数据库超级用户（postgres）。
- -W 参数提示要为新创建的数据库超级用户输入密码。

执行下面的命令，查看刚刚创建的 PostgreSQL 数据库集簇有哪些文件和目录：

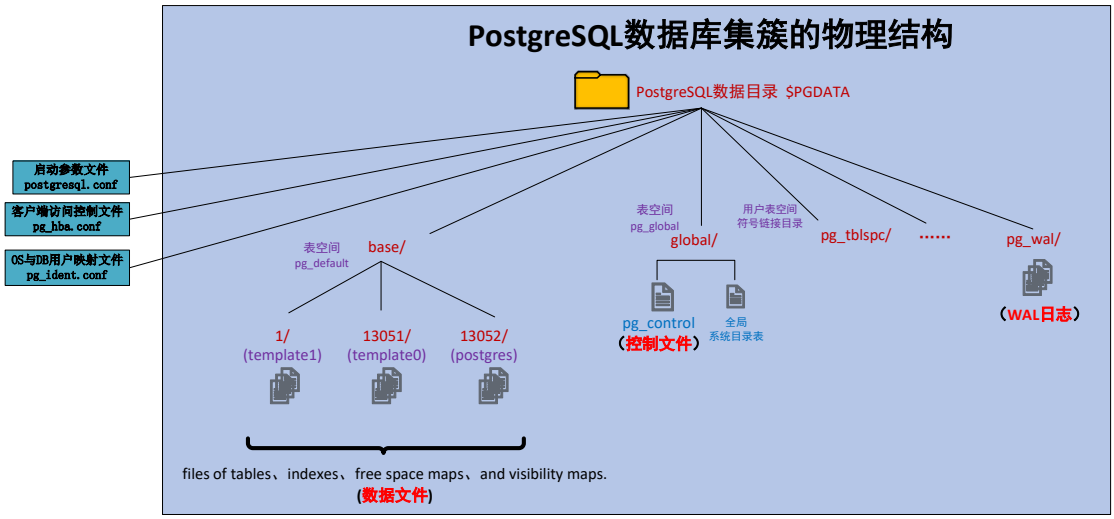
```
[postgres@dbsvr db]$ cd /opt/db/userdb/u00
[postgres@dbsvr u00]$ tree -L 1 pgdata/
pgdata/
├── base
├── global
├── pg_commit_ts
├── pg_dynshmem
├── pg_hba.conf
├── pg_ident.conf
├── pg_logical
├── pg_multixact
├── pg_notify
├── pg_replslot
├── pg_serial
├── pg_snapshots
├── pg_stat
├── pg_stat_tmp
├── pg_subtrans
├── pg_tblspc
├── pg_twophase
├── PG_VERSION
├── pg_wal
├── pg_xact
├── postgresql.auto.conf
└── postgresql.conf

17 directories, 5 files
[postgres@dbsvr u00]$
```

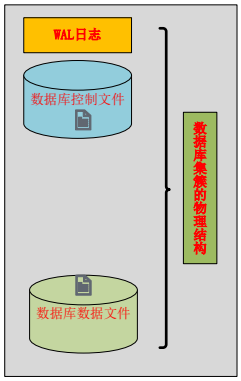
可以看到，在数据目录（$PGDATA）下，有如下的文件和子目录，如图 1-XXX 所示：

- base：表空间 pg_default 所在的目录。
- global：表空间 pg_gloabal 所在的目录，包含跨所有数据库共享的数据（PostgreSQL 实例级别的数据），如系统目录表和全局表。此外，PostgreSQL 数据库的控制文件也在该目录下。
- pg_commit_ts：存储已提交事务的时间戳数据，用于事务时间戳跟踪。
- pg_dynshmem：保存与任何特定进程无关的动态共享内存段。
- pg_hba.conf：基于主机的认证（Host-Based Authentication）配置文件，控制客户端认证和访问。

- pg_ident.conf：用于操作系统用户账号与 PostgreSQL 数据库用户帐户之间的映射的配置文件。
- pg_logical：包含逻辑复制的信息。
- pg_multixact：存储多事务状态的信息，涉及行级锁定的共享信息。
- pg_notify：存储 LISTEN 和 NOTIFY 命令的异步通知队列。
- pg_replslot：保存复制槽数据，复制槽用于 WAL 记录的持久保留。
- pg_serial：序列化事务的信息存储在这里。
- pg_snapshots：导出的快照信息存储在这个目录。
- pg_stat 和 pg_stat_tmp：存储数据库的统计信息。pg_stat 用于持久统计数据，而 pg_stat_tmp 存储临时统计数据。
- pg_subtrans：子事务的状态信息。
- pg_tblspc：存储表空间的符号链接，指向实际表空间数据的存储位置。
- pg_twophase：存储两阶段提交的事务预备文件。
- PG_VERSION：显示 PostgreSQL 服务器版本的文件。
- pg_wal：包含写前日志（WAL，之前称为 pg_xlog）文件，这些文件对恢复和复制至关重要。
- pg_xact：存储事务状态信息，如事务提交或回滚。
- postgresql.auto.conf：自动配置文件，包含由 ALTER SYSTEM 命令设置的参数。
- postgresql.conf：主要的系统参数配置文件，控制 PostgreSQL 服务器的所有主要行为和性能相关的参数。



新创建的 PostgreSQL 数据库集簇，其物理结构如图 1-XXX 所示，主要包括以下 3 个组件：数据文件、控制文件和 WAL 日志文件。



1.4 首次启动 PostgreSQL 数据库

1.4.1 PostgreSQL 数据库配置文件

严格地说，PostgreSQL 数据库配置文件不是 PostgreSQL 数据库集簇的物理组成部分。但是，PostgreSQL 数据库配置文件定制了 PostgreSQL 数据库的运行特征，对 PostgreSQL 数据库的高效运行至关重要。主要有以下三种配置文件：

- 系统配置参数文件 postgresql.conf
- 客户端访问控制文件 pg_hba.conf
- 本地操作系统用户与数据库用户的身份映射文件 pg_ident.conf

这些 PostgreSQL 数据库配置文件位于 PostgreSQL 数据库的数据目录下。

1.4.2 配置 PostgreSQL 数据库的系统参数文件

以 postgres 用户的身份，编辑/opt/db/userdb/u00/pgdata/postgresql.conf 系统参数配置文件，修改 PostgreSQL 数据库的系统参数：

```
[postgres@dvsvr ~]$ vi /opt/db/userdb/u00/pgdata/postgresql.conf
```

第 1 处修改：配置 PostgreSQL 数据库的监听 IP 和端口

找到如下的行：

```
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

#listen_addresses = 'localhost'          # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
#port = 5432                            # (change requires restart)
```

将下面 2 个参数修改为如下的值：

```
listen_addresses = '*'                  # what IP address(es) to listen on;
port = 5432                             # (change requires restart)
```

第 2 处修改：配置启动日志进程

```
logging_collector = on          #启用日志收集器（启动日志进程）。
log_directory = 'log'
```

1.4.3 配置 PostgreSQL 数据库的访问策略

以 postgres 用户的身份，编辑/opt/db/userdb/u00/pgdata/pg_hba.conf 用户访问控制文件，配置 PostgreSQL 数据库的客户端访问控制策略：

```
[postgres@dvsvr ~]$ vi /opt/db/userdb/u00/pgdata/pg_hba.conf
```

找到下面的 2 行：

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
```

在这两行的后面添加一行：

```
host    all             all             192.168.100.0/24        md5
```

添加的这一行，含义是任何来自网络 192.168.100.0/24 的主机，使用任何数据库用户（第 2 个 all），访问任何数据库（第 1 个 all）时，都需要提供数据库用户的密码，密码的加密方式为 md5。如果将此处的 md5 修改为 trust，就不需要输入密码进行身份认证了。感兴趣的读者可以测试一下。

在文件/opt/db/userdb/pgdata/pg_hba.conf 中，还有如下的两行：

```
# "local" is for Unix domain socket connections only
local   all             all                                     trust
```

这里 trust 的含义是任何数据库用户（第 2 个 all），使用本地套接字访问任何数据库（第 1 个 all），都是被信任的（不需要提供数据库用户的密码）。如果将此处的 trust 修改为 md5，本地连接 PostgreSQL 数据库，就需要输入密码进行身份认证了。感兴趣的读者可以测试一下。

1.4.4 启动 PostgreSQL 数据库

执行下面的命令，可以正常启动 PostgreSQL 数据库：

以 postgres 用户的身份，执行下面的命令，启动 PostgreSQL 数据库：

```
[postgres@dvsvr ~]$ pg_ctl -D /opt/db/userdb/u00/pgdata start
（省略了一些输出）
server started
[postgres@dbsvr ~]$ pgps
postgres  23879      1  0 12:51 ?        00:00:00 /opt/db/pg14/bin/postgres -D /opt/db/userdb/u00/pgdata
postgres  23880  23879  0 12:51 ?        00:00:00 postgres: logger
postgres  23882  23879  0 12:51 ?        00:00:00 postgres: checkpointer
postgres  23883  23879  0 12:51 ?        00:00:00 postgres: background writer
postgres  23884  23879  0 12:51 ?        00:00:00 postgres: walwriter
postgres  23885  23879  0 12:51 ?        00:00:00 postgres: autovacuum launcher
postgres  23886  23879  0 12:51 ?        00:00:00 postgres: stats collector
postgres  23887  23879  0 12:51 ?        00:00:00 postgres: logical replication launcher
[postgres@dbsvr ~]$ pstree -p 23879
postgres(23879)─┬─postgres(23880)
                ├─postgres(23882)
                ├─postgres(23883)
                ├─postgres(23884)
                ├─postgres(23885)
                ├─postgres(23886)
                └─postgres(23887)
[postgres@dbsvr ~]$
```

1.4.5 查看 PostgreSQL 系统配置文件

执行下面的 psql 命令和 SQL 语句，查看 PostgreSQL 配置文件的位置：

```
[postgres@dbsvr ~]$ psql -d postgres -U postgres
psql (14.11)
Type "help" for help.
postgres=#  SELECT name, setting
postgres-#  FROM pg_settings
postgres-#  WHERE name IN ('config_file','hba_file','ident_file');
   name     |               setting
------------+----------------------------------------
config_file |/opt/db/userdb/u00/pgdata/postgresql.conf  启动参数文件
hba_file    |/opt/db/userdb/u00/pgdata/pg_hba.conf      客户端访问控制文件
ident_file  |/opt/db/userdb/u00/pgdata/pg_ident.conf    OS 用户与 DB 用户的身份映射文件
(3 rows)
postgres=# \q
[postgres@dbsvr ~]$
```

1.4.6 查看 PostgreSQL 数据库的运行状态

执行下面的命令，查看 PostgreSQL 数据库的运行状态：

```
[postgres@dvsvr ~]$ pg_ctl -D /opt/db/userdb/u00/pgdata status
pg_ctl: server is running (PID: 23879)
/opt/db/pg14/bin/postgres "-D" "/opt/db/userdb/u00/pgdata"
[postgres@dvsvr ~]$
```

可以看出，当前 PostgreSQL 数据库正在运行中。

1.4.7 关闭 PostgreSQL 数据库

要停止 PostgreSQL 数据库，可以以 postgres 用户的身份，执行下面的命令：

```
[postgres@dbsvr ~]$ pg_ctl -D /opt/db/userdb/u00/pgdata stop
waiting for server to shut down.... done
server stopped
[postgres@dbsvr ~]$
```

1.4.8 以“Standalone Backend”方式启动 PostgreSQL 数据库

有时候，DBA 需要以“Standalone Backend”的方式，启动 PostgreSQL 数据库服务器，执行 PostgreSQL 数据库紧急维护。

在第 1 个 Linux 终端上，以 Linux 用户 postgres 的身份，执行下面的命令，以单用户的模式（Standalone Backend），启动 PostgreSQL 数据库服务器，登录访问 postgres 数据库。

```
[postgres@dbsvr ~]$ postgres --single -D /opt/db/userdb/u00/pgdata postgres

PostgreSQL stand-alone backend 14.11
backend>
```

其中：

- 命令的第 1 个 postgres 是可执行程序的名字。
- 命令的最后是第 2 个 postgres，代表的是数据库 postgres。

在第 2 个 Linux 终端上，以 Linux 用户 postgres 的身份，执行下面的 ps 命令：

```
[postgres@dbsvr ~]$ ps -ef|grep postgres|grep single|grep -v grep
postgres   24316    2258  0 12:55 pts/0    00:00:00 postgres --single -D
                                                    /opt/db/userdb/u00/pgdata postgres
[postgres@dbsvr ~]$
```

可以看到，以单用户的模式（Standalone Backend）启动 PostgreSQL 数据库服务器，只有一个进程。

在第 1 个 Linux 终端上，按“Ctrl+d”组合键，退出当用户模式。

1.5 配置 PostgreSQL 数据库运行在归档日志模式

用户现场的生产数据库，一般都运行在自动归档日志模式下。执行下面的步骤，配置 PostgreSQL 数据库运行在归档日志模式：

（1） 执行下面的命令，启动 PostgreSQL 数据库后，查看当前 PostgreSQL 数据库的归档设置：

```
[postgres@dvsvr ~]$ pg_ctl -D /opt/db/userdb/u00/pgdata start
[postgres@dvsvr ~]$ psql -d postgres -U postgres -c \
  "SELECT name,setting FROM pg_settings WHERE name like 'archive%' or name='wal_level';"
         name          |  setting
-----------------------+-----------
 archive_cleanup_command |
 archive_command         | (disabled)
 archive_mode            | off
 archive_timeout         | 0
 wal_level               | replica
(5 rows)
[postgres@dvsvr ~]$
```

可以看到，当前 PostgreSQL 数据库工作在非归档方式下（archive_mode 的值为 off）。

执行下面的步骤，配置 PostgreSQL 数据库运行在归档日志模式下：

（1） 关闭 PostgreSQL 数据库：

```
[postgres@dvsvr ~]$ pg_ctl -D /opt/db/userdb/u00/pgdata stop
```

（2） 创建用于保存归档日志的目录：

```
[postgres@dvsvr ~]$ mkdir -p  /opt/db/userdb/u02/archivelog
```

（3） 以 postgres 用户的身份，编辑文件/opt/db/userdb/u00/pgdata/postgresql.conf：

```
[postgres@dvsvr ~]$ vi  /opt/db/userdb/u00/pgdata/postgresql.conf
```

修改配置文件中关于 WAL 和归档设置的参数：

```
#  wal_level 可以取以下的值：minimal, replica,or logical
# 修改 wal_level 的值需要重启动数据库
# 工作在归档模式下不能设置为 minimal，可以设置为除 minmal 之外的其他参数
wal_level = replica
# 修改 archive_mode 的值需要重启动数据库
archive_mode=on
# 修改 archive_command 的值不需要重启动数据库，只需要 reload
archive_command = 'cp %p /opt/db/userdb/u02/archivelog/%f'
# 修改 archive_time：归档周期，900 表示每 900 秒（15 分钟）切换一次
archive_timeout = 900
```

（4） 执行下面的命令，启动 PostgreSQL 数据库：

```
[postgres@dvsvr ~]$ pg_ctl -D /opt/db/userdb/u00/pgdata start
```

（5） 执行下面的命令和 SQL 语句，查看当前 PostgreSQL 数据库的归档设置：

```
[postgres@dvsvr ~]$ psql -d postgres -U postgres -c \
  "SELECT name,setting FROM pg_settings WHERE name like 'archive%' or name='wal_level';"
         name          |               setting
-----------------------+-----------------------------------
 archive_cleanup_command |
 archive_command         | cp %p /opt/db/userdb/u02/archivelog/%f
 archive_mode            | on
 archive_timeout         | 900
 wal_level               | replica
(5 rows)
[postgres@dvsvr ~]$
```

可以看到，PostgreSQL 数据库已经工作在归档模式下了（archive_mode 的值为 on）。

执行下面的命令，备份这个配置好的 PostgreSQL 数据库集簇：

```
[postgres@dvsvr ~]$  pg_ctl -D /opt/db/userdb/u00/pgdata stop
[postgres@dbsvr ~]$  cd /opt/db
[postgres@dbsvr db]$ tar cf userdb.tar userdb
[postgres@dbsvr db]$ ls -l userdb.tar
-rw-r--r--. 1 postgres dba 76554240 Apr 15 12:59 userdb.tar
[postgres@dbsvr db]$
```

1.6 导入大学应用数据库 universitydb

将本书的资源文件 university.sql 上传到 PostgreSQL 服务器的目录/home/postgres 下。

执行下面的命令，启动 PostgreSQL 数据库：

```
[postgres@dbsvr ~]$ pg_ctl start
```

执行下面的命令，创建数据库 universitydb，并导入大学应用测试数据集。

```
[postgres@dbsvr ~]$ psql -d postgres -U postgres -c "CREATE DATABASE universitydb;" -q
[postgres@dbsvr ~]$ psql -d universitydb -U postgres -f university.sql -q
```

执行下面的命令，备份这个导入了大学应用数据集的 PostgreSQL 数据库集簇：

```
[postgres@dvsvr ~]$  pg_ctl stop
[postgres@dbsvr ~]$  cd /opt/db/
[postgres@dbsvr db]$ tar cf userdbWithData.tar userdb
[postgres@dbsvr db]$ ls -l userdbWithData.tar
-rw-r--r--. 1 postgres dba 102369280 Apr 15 13:01 userdbWithData.tar
[postgres@dbsvr db]$
```

1.7 启动 PostgreSQL 数据库后的进程情况

启动 PostgreSQL 数据库，就是启动一个 PostgreSQL 数据库实例。PostgreSQL 数据库实例是 PostgreSQL 数据库管理系统在内存中的组件，包括内存进程、以及由这些内存进程共享的内存数据结构。必须通过 PostgreSQL 实例才能访问 PostgreSQL 数据库集簇。

以 Linux 用户 postgres 的身份，执行下面的命令，启动 PostgreSQL 数据库：

```
[postgres@dvsvr ~]$ pg_ctl start
```

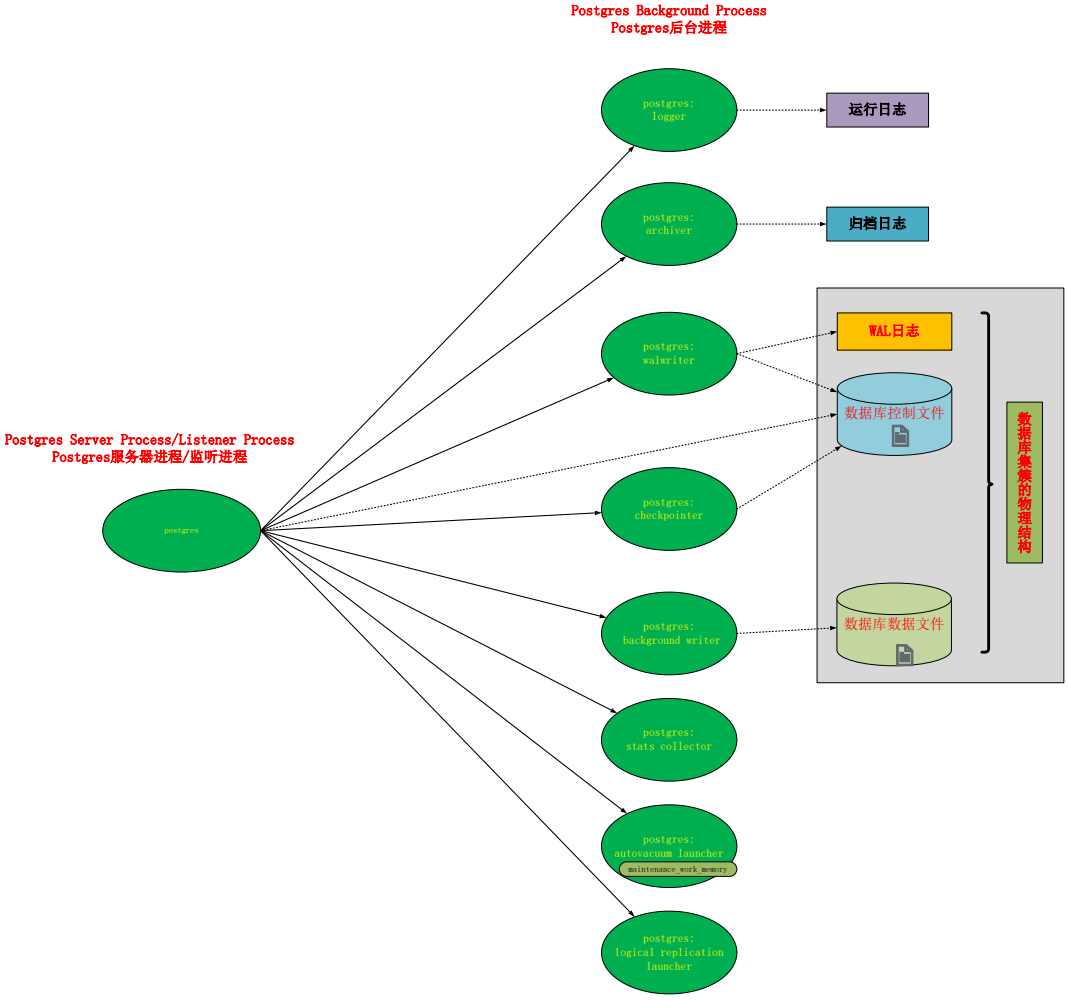
执行下面的命令，查看启动后的 PostgreSQL 数据库实例进程：

```
[postgres@dbsvr db]$ pgps
postgres   25192      1  0 13:04 ?        00:00:00 /opt/db/pg14/bin/postgres
postgres   25193  25192  0 13:04 ?        00:00:00 postgres: logger
postgres   25195  25192  0 13:04 ?        00:00:00 postgres: checkpointer
postgres   25196  25192  0 13:04 ?        00:00:00 postgres: background writer
postgres   25197  25192  0 13:04 ?        00:00:00 postgres: walwriter
postgres   25198  25192  0 13:04 ?        00:00:00 postgres: autovacuum launcher
postgres   25199  25192  0 13:04 ?        00:00:00 postgres: archiver
postgres   25200  25192  0 13:04 ?        00:00:00 postgres: stats collector
postgres   25201  25192  0 13:04 ?        00:00:00 postgres: logical replication launcher
[postgres@dbsvr db]$  pstree -p 25192
```

```
postgres(25192)─┬─┬─postgres(25193)
                ├─postgres(25195)
                ├─postgres(25196)
                ├─postgres(25197)
                ├─postgres(25198)
                ├─postgres(25199)
                ├─postgres(25200)
                └─postgres(25201)
[postgres@dbsvr db]$
```

可以看到，启动后的 PostgreSQL 数据库实例进程，如图所示。这些进程可以分为 2 类：

- postgres 服务器进程/监听（postgres Server Process/Listener）：PID 为 25192，进程名为“postgres”的进程，它是 PostgreSQL 数据库实例的所有其他进程的父进程。
- postgres 后台进程（postgres Background Process）：也被称为 PostgreSQL 辅助进程。当前有如下的后台进程：
  - postgres: logger （运行日志进程）
  - postgres: checkpointer（检查点进程）
  - postgres: background writer（后台写进程）
  - postgres: walwriter（WAL 日志写进程）
  - postgres: autovacuum launcher（自动清理启动进程）
  - postgres: archiver（归档日志进程）
  - postgres: stats collector（统计数据收集进程）
  - postgres: logical replication launcher（逻辑复制启动进程）



1.8 编译安装 PostgreSQL 客户端

1.8.1 准备一台 Linux 客户端机器

准备一台机器名为 client，IP 地址为 192.168.100.32/24，安装了 openEuler 22.03 操作系统的 Linux 计算机。

1.8.2 安装前的准备工作

首先在 PostgreSQL 客户端上，执行编译安装 PostgreSQL 服务器的准备工作（1）~（4）。

然后在 PostgreSQL 客户端上，以 postgres 用户的身份，使用 vi 编辑器，编辑 postgres 用户的环境文件/home/postgres/.bash_profile：

```
[postgres@client ~]$ vi /home/postgres/.bash_profile
```

在文件的末尾，添加如下的行：

```
export PGCLIENT=/opt/db/pgsql
export PATH=$PGCLIENT/bin:$PATH
```

接下来以 postgres 用户的身份，执行如下的命令，启用这些环境变量：

```
[postgres@client ~]$ source ~/.bash_profile
```

1.8.3 配置编译 PostgreSQL 源代码

配置编译 PostgreSQL 数据库客户端与配置编译 PostgreSQL 数据库服务器一样。

1.8.4 编译 PostgreSQL 数据库源代码

在 PostgreSQL 客户端上，使用 postgres 用户，执行下面的命令，编译安装 PostgreSQL14.11 客户端程序和接口库：

```
[postgres@dbsvr postgresql-14.11]$ make -C src/bin
[postgres@dbsvr postgresql-14.11]$ make -C src/include
[postgres@dbsvr postgresql-14.11]$ make -C src/interfaces
[postgres@dbsvr postgresql-14.11]$ make -C doc
```

make 命令的-C 选项，其含义是切换到指定的目录进行编译。

1.8.5 安装 PostgreSQL 客户端二进制程序

在 PostgreSQL 客户端上，使用 postgres 用户，执行下面的命令，安装 PostgreSQL14.11 客户端程序和接口库：

```
[postgres@client postgresql-14.11]$ make -C src/bin install
[postgres@client postgresql-14.11]$ make -C src/include install
[postgres@client postgresql-14.11]$ make -C src/interfaces install
[postgres@client postgresql-14.11]$ make -C doc install
```

提示：PostgreSQL 客户端的二进制程序被安装在目录/opt/db/pg14 下

执行下面的命令，创建符号连接/opt/db/pgsql -> /opt/db/pg14：

```
[postgres@client postgresql-14.11]$ cd /opt/db
[postgres@client db]$ ls
pg14  soft
[postgres@client db]$ ln -s /opt/db/pg14 /opt/db/pgsql
[postgres@client db]$ ls -l
total 8
drwxr-xr-x. 6 postgres dba 4096 Apr 14 11:54 pg14
```

```
lrwxrwxrwx. 1 postgres dba   12 Apr 14 11:54 pgsql -> /opt/db/pg14
drwxr-xr-x. 3 postgres dba 4096 Apr 14 11:52 soft
[postgres@client db]$
```

使用 postgres 用户，执行下面的 psql 命令（PostgreSQL 的客户端程序），查看 PostgreSQL 客户端的版本信息：

```
[postgres@client db]$ psql --version
psql (PostgreSQL) 14.11
[postgres@client db]$ /opt/db/pgsql/bin/psql --version
psql (PostgreSQL) 14.11
[postgres@client db]$ /opt/db/pg14/bin/psql  --version
psql (PostgreSQL) 14.11
[postgres@client db]$
```

1.8.6 打包 PostgreSQL 客户端二进制程序

在 PostgreSQL 客户端上，执行下面的命令，将编译后的 PostgreSQL 客户端二进制程序打包为 tar 安装包：

```
[postgres@client ~]$  cd /opt/db/
[postgres@client db]$ tar cf postgresql-14.11.client.tar pg14/
[postgres@client db]$ ls -lh postgresql-14.11.client.tar
-rw-r--r--. 1 postgres dba 36M Apr 14 11:57 postgresql-14.11.client.tar
[postgres@client db]$
```

生成的 tar 包文件 postgresql-14.11.client.tar 可以作为 PostgreSQL 客户端二进制发行版（Linux 平台，版本号为 14.11），用于在其他计算机上安装 PostgreSQL 客户端（Linux 平台）。

1.9 客户端连接访问 PostgreSQL 数据库

PostgreSQL 采用 C/S（客户机/服务器）架构。

在开始本节的学习之前，请读者执行如下的操作，构建本节的实验环境：

（1） 在 PostgreSQL 服务器上，以 postgres 用户的身份，编辑 PostgreSQL 数据库用户访问控制文件/opt/db/userdb/u00/pgdata/pg_hba.conf：

```
[postgres@dvsvr ~]$ vi /opt/db/userdb/u00/pgdata/pg_hba.conf
```

找到如下的两行：

```
# "local" is for Unix domain socket connections only
```

```
local   all             all                                     trust
```

将第 2 行修改为如下：

```
local   all             all                                     md5
```

修改后的含义是：使用本地套接字连接 PostgreSQL 数据库，需要输入 md5 加密的密码进行身份认证。

（2） 在 PostgreSQL 服务器上，执行下面的命令，重新启动 PostgreSQL 数据库：

```
[postgres@dbsvr db]$ pg_ctl restart
```

1.9.1 使用 IP 协议远程连接 PostgreSQL 服务器

如图 1-XX 所示，位于 IP 地址为 192.168.100.32/24 的“客户端进程”，可以执行下面的命令，使用 Internet 协议（IP 协议），向位于 IP 地址为 192.168.100.31/24 的“PostgreSQL 服务器进程”，发出访问数据库 universitydb 的服务请求（步骤①）：

```
[postgres@client ~]$ psql -d universitydb -U postgres -h 192.168.100.31 -p 5432
Password for user postgres:
```

此时，在 PostgreSQL 客户端（192.168.100.32/24）和 PostgreSQL 服务器（192.168.100.31/24）之间，创建了一个**连接（Connection）**。

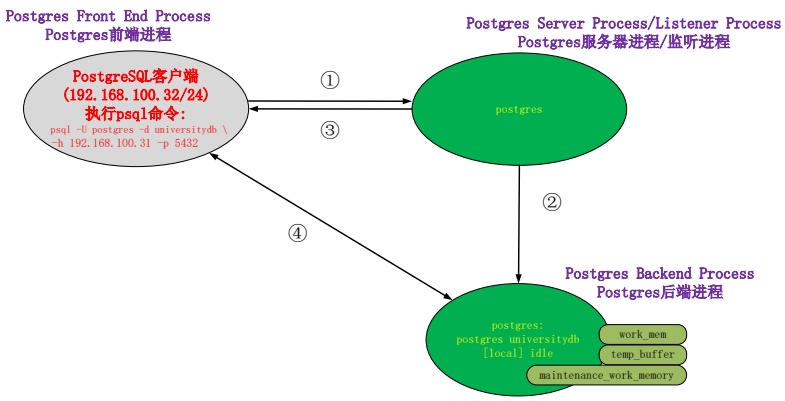
在 PostgreSQL 客户端，输入数据库用户 postgres 的密码：

```
Password for user postgres:（输入数据库用户 postgres 的密码“dba123”）
```

一旦通过了用户身份认证，“PostgreSQL 服务器进程”将为“客户端进程”创建一个“PostgreSQL 后端进程”（步骤②），并将这个新创建的“PostgreSQL 后端进程”的连接信息，告知给“客户端进程”（步骤③）。“客户端进程”使用获得的“PostgreSQL 后端进程”的连接信息，创建与该“PostgreSQL 后端进程”的会话（Session）连接（步骤④）：

```
psql (14.11)
Type "help" for help.
universitydb=#
```

之后，“客户端进程”可以向 “PostgreSQL 后端进程”发出 SQL 语句，由“PostgreSQL 后端进程”，代表“客户端进程”，在 PostgreSQL 服务器上，执行 SQL 语句，并将执行结果返回给“客户端进程”。



执行下面的 SQL 语句，查看 PostgreSQL 客户端的 IP 地址：

```
universitydb=# SELECT inet_client_addr() AS client_ip;
   client_ip
----------------
 192.168.100.32
(1 row)
universitydb=#
```

执行下面的命令，查看 PostgreSQL 客户端上当前会话的前端进程的 PID：

```
universitydb=# \! ps |grep psql
  28185 pts/0    00:00:00 psql
universitydb=#
```

执行下面的 SQL 语句，查看 PostgreSQL 服务器的 IP 地址：

```
universitydb=# SELECT inet_server_addr() AS server_ip;
   server_ip
----------------
 192.168.100.31
(1 row)
universitydb=#
```

执行下面的 SQL 语句，查看 PostgreSQL 服务器上代表当前会话的后端进程的 PID：

```
universitydb=# SELECT pg_backend_pid();
 pg_backend_pid
----------------
          28857
(1 row)
universitydb=#
```

在 PostgreSQL 服务器上，打开另外一个 Linux 终端，执行下面的命令：

```
[postgres@dbsvr ~]$ pgps
postgres   28842      1  0 14:37 ?        00:00:00 /opt/db/pg14/bin/postgres
postgres   28843  28842  0 14:37 ?        00:00:00 postgres: logger
postgres   28845  28842  0 14:37 ?        00:00:00 postgres: checkpointer
postgres   28846  28842  0 14:37 ?        00:00:00 postgres: background writer
postgres   28847  28842  0 14:37 ?        00:00:00 postgres: walwriter
postgres   28848  28842  0 14:37 ?        00:00:00 postgres: autovacuum launcher
```

```
postgres   28849   28842  0 14:37 ?        00:00:00 postgres: archiver
postgres   28850   28842  0 14:37 ?        00:00:00 postgres: stats collector
postgres   28851   28842  0 14:37 ?        00:00:00 postgres: logical replication launcher
postgres   28857   28842  0 14:37 ?        00:00:00 postgres: postgres universitydb 192.168.100.32(45152) idle
[postgres@dbsvr ~]$
```

1.9.2 使用 Socket 本地连接 PostgreSQL 服务器

如图 1-XX 所示，位于 PostgreSQL 数据库服务器上的“客户端进程”，可以使用 Unix 本地套接字协议（Socket），向“PostgreSQL 服务器进程”，发出访问数据库 universitydb 的服务请求（步骤①）。

```
[postgres@dbsvr db]$ psql -d universitydb -U postgres
Password for user postgres:
```

此时，位于 PostgreSQL 服务器上的 PostgreSQL 客户端进程与 PostgreSQL 服务器进程之间，创建了一个连接（Connection）。

此时我们输入数据库用户 postgres 的正确密码：

```
Password for user postgres:（输入数据库用户 postgres 的密码“dba123”）
```

一旦通过了用户身份认证，“PostgreSQL 服务器进程”将为“客户端进程”创建一个“PostgreSQL 后端进程”（步骤②），并将这个新创建的“PostgreSQL 后端进程”的连接信息，告知给“客户端进程”（步骤③）。“客户端进程”使用获得的“PostgreSQL 后端进程”的连接信息，创建与该“PostgreSQL 后端进程”的会话（Session）连接（步骤④）：

```
psql (14.11)
Type "help" for help.
universitydb=#
```

之后，“客户端进程”可以向“PostgreSQL 后端进程”发出 SQL 语句，由“PostgreSQL 后端进程”，代表“客户端进程”，在 PostgreSQL 服务器上，执行 SQL 语句，并将执行结果返回给“客户端进程”。

执行下面的 SQL 语句，查看 PostgreSQL 服务器上代表当前会话的后端进程的 PID：

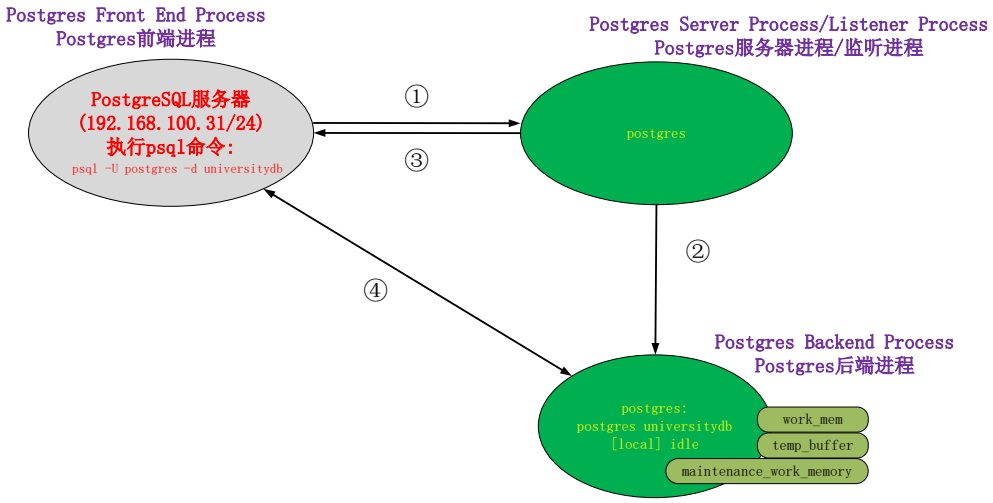
```
universitydb=# SELECT pg_backend_pid();
 pg_backend_pid
----------------
          28904
(1 row)
universitydb=#
```

执行下面的命令，查看 PostgreSQL 服务器上，当前会话的前端进程的 PID：

```
universitydb=# \! ps |grep psql
  28901 pts/1    00:00:00 psql
universitydb=#
```

在 PostgreSQL 服务器上，打开另外一个 Linux 终端，执行下面的命令：

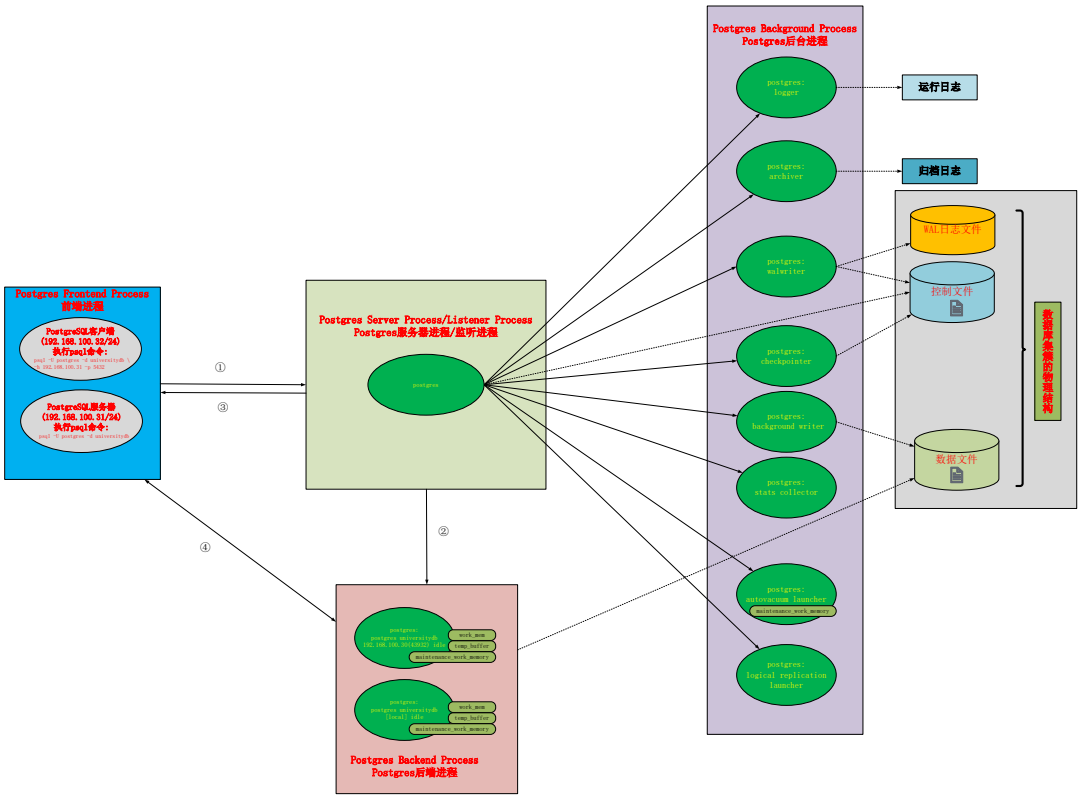
```
[postgres@dbsvr ~]$ pgps
postgres   28842      1  0 14:37 ?        00:00:00 /opt/db/pg14/bin/postgres
postgres   28843  28842  0 14:37 ?        00:00:00 postgres: logger
postgres   28845  28842  0 14:37 ?        00:00:00 postgres: checkpointer
postgres   28846  28842  0 14:37 ?        00:00:00 postgres: background writer
postgres   28847  28842  0 14:37 ?        00:00:00 postgres: walwriter
postgres   28848  28842  0 14:37 ?        00:00:00 postgres: autovacuum launcher
postgres   28849  28842  0 14:37 ?        00:00:00 postgres: archiver
postgres   28850  28842  0 14:37 ?        00:00:00 postgres: stats collector
postgres   28851  28842  0 14:37 ?        00:00:00 postgres: logical replication launcher
postgres   28857  28842  0 14:37 ?        00:00:00 postgres: postgres universitydb 192.168.100.32(45152) idle
postgres   28904  28842  0 14:39 ?        00:00:00 postgres: postgres universitydb [local] idle
[postgres@dbsvr ~]$
```



1.10 PostgreSQL 数据库的进程分类

客户端进程连接 PostgreSQL 数据库，访问 PostgreSQL 数据库集簇中的数据库，如图 1-XXX 所示，涉及以下四类进程：

- Postgres 前端进程
- Postgres 服务器进程
- Postgres 后端进程
- Postgres 后台进程



① Postgres 前端进程（客户端进程）向数据库服务器的 Postgres 服务器进程发起连接请求。

② 通过身份认证后，Postgres 服务器进程将派生 1 个新的 Postgres 后端进程。

③ Postgres 服务器进程将连接 Postgres 后端进程的信息发送给 Postgres 前端进程（客户端进程）。

④ 在 Postgres 前端进程（客户端进程）和 Postgres 后端进程之间建立会话。在会话上，Postgres 前端进程（客户端进程）将要执行的 SQL 语句发送给 Postgres 后端进程；Postgres 后端进程负责解析 SQL 语句、生成 SQL 语句的执行计划、执行 SQL 语句；将执行结果发送回 PostgreSQL 前端进程（客户端进程）。

图 1-XXX 中的虚线箭头方向表示有该进程读写对应的文件。

1.11 PostgreSQL 数据库对象及其 OID

PostgreSQL 数据库中的一切皆对象。更准确地说，PostgreSQL 数据库对象，指的是

PostgreSQL 数据库集簇对象，经常被简称为“数据库对象”，它们可以是：数据库用户、表空间、数据库、表、视图、索引、系列、存储过程或函数、触发器、……。

可以将这些数据库对象分为 2 类：

- 有些 PostgreSQL 数据库集簇对象，如表、索引、视图、系列、函数、过程等，可以属于某个数据库下的某个模式，它们被称为**模式对象（Schema Object）**；
- 另外一些 PostgreSQL 数据库集簇对象，如表空间、数据库、用户等，不能属于一个模式，它们被称为**非模式对象（non-Schema Object）**。

执行下面的命令，恢复有大学应用测试数据集的 PostgreSQL 数据库集簇：

```
[postgres@dbsvr ~]$  cd /opt/db
[postgres@dbsvr db]$ pg_ctl stop
[postgres@dbsvr db]$ rm -rf userdb
[postgres@dbsvr db]$ tar xf userdbWithData.tar
[postgres@dbsvr db]$ pg_ctl start
```

每一个数据库对象都有一个唯一的**对象标识（Object IDentifier，OID）**。可以使用 oid2name 命令或 SQL 语句来查看数据库对象的标识（OID）。

使用 oid2name 命令查看所有数据库的名字及其 oid、默认表空间信息：

```
[postgres@dbsvr ~]$ oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  13052      postgres  pg_default
  13051     template0  pg_default
      1     template1  pg_default
  16384  universitydb  pg_default
[postgres@dbsvr ~]$
```

使用 oid2name 命令查看所有表空间的名字及其 oid 信息：

```
[postgres@dbsvr ~]$ oid2name -s
All tablespaces:
   Oid  Tablespace Name
-----------------------
  1663       pg_default
  1664        pg_global
[postgres@dbsvr ~]$
```

也可以使用 SQL 语句，通过查询 PostgreSQL 的系统表，来获取数据库对象的 OID：

```
[postgres@dbsvr ~]$ psql -d universitydb -U postgres
```

```
universitydb=# -- 查看数据库用户 postgres 的对象标识（OID）
universitydb=# SELECT oid, rolname FROM pg_roles WHERE rolname = 'postgres';
 oid | rolname
-----+----------
  10 | postgres
(1 row)
universitydb=# -- 查看数据库 universitydb 的对象标识（OID）
universitydb=# SELECT oid, datname FROM pg_database WHERE datname='universitydb';
  oid  |   datname
-------+--------------
 16384 | universitydb
(1 row)
universitydb=# -- 查看表空间 pg_default 的对象标识（OID）
universitydb=# SELECT oid, spcname FROM pg_tablespace WHERE spcname='pg_default';
 oid  |  spcname
------+------------
 1663 | pg_default
(1 row)
universitydb=# -- 查看表 instructor 的信息
universitydb=# \d instructor
                  Table "public.instructor"
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
 id        | character varying(5)  |           | not null |
 dept_name | character varying(20) |           |          |
 name      | character varying(20) |           | not null |
 salary    | numeric(8,2)          |           |          |
Indexes:
    "instructor_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
  "fk_sys_c0011280" FOREIGN KEY (dept_name) REFERENCES department(dept_name) ON UPDATE RESTRICT ON DELETE RESTRICT
Referenced by:
  TABLE "teaches" CONSTRAINT "fk_sys_c0011287" FOREIGN KEY (id) REFERENCES instructor(id) ON UPDATE RESTRICT ON DELETE RESTRICT
  TABLE "advisor" CONSTRAINT "fk_sys_c0011297" FOREIGN KEY (id) REFERENCES instructor(id) ON UPDATE RESTRICT ON DELETE RESTRICT

universitydb=# -- 查看表 instructor 的对象标识（OID）
universitydb=# SELECT oid, relname FROM pg_class WHERE relname = 'instructor';
  oid  |  relname
-------+------------
 16397 | instructor
(1 row)
universitydb=# -- 查看索引 instructor_pkey 的对象标识（OID）
universitydb=# SELECT oid, relname FROM pg_class WHERE relname = 'instructor_pkey';
  oid  |     relname
-------+-----------------
 16426 | instructor_pkey
(1 row)
universitydb=# \q
[postgres@dbsvr ~]$
```

1.12 数据库

多个“**数据库（Database）**”的集合被称为 **PostgreSQL 数据库集簇（Database Cluster）**。一个 PostgreSQL 数据库集簇有多个数据库；一个数据库只能属于一个 PostgreSQL 数据库集簇。

1.12.1 PostgreSQL 数据库集簇有哪些数据库

执行下面的命令，恢复刚刚创建并配置好的 PostgreSQL 数据库集簇：

```
[postgres@dbsvr ~]$  cd /opt/db
[postgres@dbsvr db]$ pg_ctl stop
[postgres@dbsvr db]$ rm -rf userdb
[postgres@dbsvr db]$ tar xf userdb.tar
[postgres@dbsvr db]$ pg_ctl start
```

执行下面的命令，查看初始创建的 PostgreSQL 数据库集簇，有哪些数据库：

```
[postgres@dbsvr ~]$ psql -d postgres -U postgres -c "\l" -q
                                 List of databases
    Name     |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------+----------+----------+-------------+-------------+-----------------------
 postgres    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 template1   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
(3 rows)
[postgres@dbsvr ~]$
```

可以看到，初始创建的 PostgreSQL 数据库集簇，有 3 个数据库，其中：

- 模板数据库 TEMPLATE1 是在 initdb 命令的 bootstrap 模式运行阶段创建的、最原始的模板数据库。
- 模板数据库 TEMPLATE0 是根据模板数据库 TEMPLATE1 创建的。模板数据库 template1 会被用户修改，因此创建模板数据库 template0，作为最初始的备份。
- 数据库 postgres 也是根据模板数据库 TEMPLATE1 创建的，作为一个可以被用户访问的初始数据库。PostgreSQL 数据库管理员，通过连接访问 postgres 数据库，来管理 PostgreSQL 数据库。

1.12.2 使用 SQL 语句 CREATE DATABASE 创建数据库

（1） 执行下面的 psql 命令和 SQL 语句，创建大学应用数据库 universitydb：

```
[postgres@dbsvr ~]$ psql -d postgres -U postgres
postgres=# CREATE DATABASE universitydb;
CREATE DATABASE
postgres=# \q
[postgres@dvsvr ~]$
```

（2） 将本书的资源文件 university.sql 上传到服务器的目录/home/postgres 下。

（3） 执行下面的命令，导入大学应用测试数据集：

```
[postgres@dbsvr ~]$ psql -d universitydb -U postgres -f university.sql -q
```

（4） 执行下面的命令，查看当前 PostgreSQL 数据库集簇中的数据库：

```
[postgres@dbsvr ~]$ psql -d postgres -U postgres -c "\l" -q
                                  List of databases
    Name     |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------+----------+----------+-------------+-------------+-----------------------
 postgres    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 template1   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 universitydb | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(4 rows)
[postgres@dbsvr ~]$
```

可以看到，现在 PostgreSQL 数据库集簇有 4 个数据库了，其中的 universitydb 数据库，是由用户新创建的大学应用数据库。

1.12.3 使用命令 createdb 创建数据库

执行命令 createdb 创建数据库 mycollegedb：

```
[postgres@dvsvr ~]$ createdb  -U postgres  --owner=postgres mycollegedb
[postgres@dbsvr ~]$ psql -d postgres -U postgres -c"\l" -q
                                  List of databases
    Name     |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------+----------+----------+-------------+-------------+-----------------------
 mycollegedb | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 template1   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 universitydb | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
```

```
(5 rows)
[postgres@dbsvr ~]$
```

1.12.4 模板数据库

模板数据库可以用于创建一个新的、跟该模板数据库一模一样的数据库。

执行下面的步骤，可以定制一个用户的模板数据库 mycollegedb：

（1） 执行下面的命令，将大学应用数据集导入数据库 mycollegedb：

```
[postgres@dvsvr ~]$ psql -d mycollegedb -U postgres -f university.sql -q
```

（2） 执行下面的命令和 SQL 语句，将数据库 mycollegedb 修改为一个模板数据库：

```
[postgres@dvsvr ~]$ psql -d postgres -U postgres
postgres=# ALTER DATABASE mycollegedb  WITH ALLOW_CONNECTIONS false  IS_TEMPLATE true ;
ALTER DATABASE
postgres=#
```

至此，我们定制了一个自己的模板数据库 mycollegedb。

请读者注意，模板数据库不应该随便被用户修改。通过禁止用户连接到模板数据库（在 ALTER DATABASE 语句中使用了 WITH ALLOW_CONNECTIONS false 子句），可以保证这一点。

（3） 现在，可以基于模板数据库 mycollegedb 创建产品数据库 newuniversitydb 了：

```
postgres=# CREATE DATABASE newuniversitydb TEMPLATE mycollegedb ;
CREATE DATABASE
postgres=# \q
[postgres@dvsvr ~]$ psql -d newuniversitydb -U postgres
newuniversitydb=# \dt
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | advisor    | table | postgres
（省略了一些输出）
(11 rows)
newuniversitydb=# SELECT * FROM student ;
  id   | dept_name  |   name   | tot_cred
-------+------------+----------+----------
 00128 | Comp. Sci. | Zhang    |      102
（省略了一些输出）
(13 rows)
newuniversitydb=#
```

可以看到，新创建的 newuniversitydb 数据库，拥有与模板数据库 mycollegedb 一样的表和数据。

1.12.5 删除数据库

无法删除一个正在连接的数据库。

打开一个 Linux 终端（我们将其命名为“Linux 终端#1”），执行下面的 psql 命令：

```
[postgres@dvsvr ~]$ psql -d newuniversitydb -U postgres
newuniversitydb=#
```

打开另外一个 Linux 终端（我们将其命名为“Linux 终端#2”），执行下面的 psql 命令和 SQL 语句，删除 newuniversitydb 数据库：

```
[postgres@dvsvr ~]$  psql -d postgres -U postgres
postgres=# DROP DATABASE newuniversitydb ;
ERROR:  database "newuniversitydb" is being accessed by other users
DETAIL:  There is 1 other session using the database.
postgres=#
```

可以看到，此时还有 1 个用户连接访问 newuniversitydb 数据库，因此无法删除 newuniversitydb 数据库！

返回到“Linux 终端#1”，执行元命令“\q”，退出 psql：

```
newuniversitydb=# \q
[postgres@dvsvr ~]$
```

返回“Linux 终端#2”，再次执行删除 newuniversitydb 数据库的 SQL 语句：

```
postgres=# DROP DATABASE newuniversitydb ;
DROP DATABASE
postgres=#
```

可以看到，这一次 newuniversitydb 数据库被删除掉了！

无法删除一个模板数据库。在“Linux 终端#2”，执行下面的 SQL 语句，删除数据库 mycollegedb：

```
postgres=# DROP DATABASE mycollegedb ;
ERROR:  cannot drop a template database
postgres=#
```

可以看到，我们无法删除一个模板数据库！

要删除模板数据库 mycollegedb，首先需要将其更改为一个普通的数据库，然后再将其删除：

```
postgres=# ALTER DATABASE mycollegedb IS_TEMPLATE false;
ALTER DATABASE
postgres=# DROP DATABASE mycollegedb ;
DROP DATABASE
postgres=# \q
```

```
[postgres@dbsvr ~]$
```

1.13 模式和模式对象

数据库（Database）中可以包含许多数据库对象（Database Object），我们可以将这些数据库对象进行逻辑分组，每个分组就是一个**模式（Schema）**，其中的数据库对象被称为**模式对象（Schema Object）**。

1.13.1 存储模式信息的系统表

在 PostgreSQL 数据库中，模式对象及其所属的模式信息是通过系统表 pg_namespace 来管理和查看的。pg_namespace 表存储了有关数据库中所有模式的信息。每个模式本身也是一个模式对象，并且其信息也存储在这个表中。

执行下面的 psql 命令和 SQL 语句，查看 universitydb 数据库中有哪些模式：

```
[postgres@dbsvr ~]$ psql -d universitydb -U postgres
universitydb=# SELECT nspname,          -- 模式的名称
universitydb-#        nsp.oid,          -- 模式的 OID
universitydb-#        nsp.nspowner,     -- 属主的 OID
universitydb-#        rolname           -- 属主的角色名
universitydb-# FROM pg_namespace nsp, pg_authid au
universitydb-# WHERE nsp.nspowner = au.oid;
      nspname       |  oid  | nspowner | rolname
--------------------+-------+----------+----------
 pg_toast           |    99 |       10 | postgres
 pg_catalog         |    11 |       10 | postgres
 public             |  2200 |       10 | postgres
 information_schema | 12686 |       10 | postgres
(4 rows)
universitydb=# SELECT nspname AS schema_name,
universitydb-#        oid AS schema_oid,
universitydb-#        pg_get_userbyid(nspowner) AS owner_name,
universitydb-#        nspowner AS owner_oid
universitydb-# FROM pg_namespace;
    schema_name     | schema_oid | owner_name | owner_oid
--------------------+------------+------------+-----------
 pg_toast           |         99 | postgres   |        10
 pg_catalog         |         11 | postgres   |        10
 public             |       2200 | postgres   |        10
 information_schema |      12686 | postgres   |        10
(4 rows)
universitydb=#
```

可以看到，当前 universitydb 数据库中，有 4 个模式：

- public：每个数据库都有名叫“public”的模式，如果你创建表时没有指定模式，那么将会在模式 public 下创建该表。
- pg_catalog：pg_catalog 是 PostgreSQL 中每个数据库特有的模式，它包含了该数据库的系统目录（系统表），存储了数据库的元数据：表、视图、索引、数据类型、函数以及其他数据库对象的定义。PostgreSQL 数据库的这种设计，确保了数据库之间的独立性，每个数据库都是自包含的（有自己独立的一套系统目录（系统表）），使得数据库管理更灵活，数据隔离更有效。
- information_schema：是 ANSI SQL 标准的模式，包含了关于其他数据库对象的信息。它为数据库对象提供了一个只读视图，可以用来查询数据库的结构和元数据。
- pg_toast：用于存储大对象和超长行的“压缩”数据。pg_toast 模式自动处理这些数据，使得用户无需手动管理大量数据。

执行下面的 psql 命令和元命令，查看数据库 universitydb 有哪些模式：

```
[postgres@dbsvr ~]$ psql -d universitydb -U postgres
universitydb=# \dn
  List of schemas
  Name  |  Owner
--------+----------
 public | postgres
(1 row)
universitydb=#
```

可以看到，使用元命令“\dn”，只显示 public 模式，省略了其他 3 个模式。

1.13.2 在一个数据库中创建模式

对于一个关于大学应用的数据库，可以为各个行政机构创建一个模式，如为教务处创建模式 academicOffice，为人事处创建模式 humanResources。下面使用 SQL 语句 CREATE SCHEMA 来创建这两个模式：

```
universitydb=# CREATE SCHEMA academicOffice;
CREATE SCHEMA
universitydb=# CREATE SCHEMA humanResources;
CREATE SCHEMA
postgres=#
```

可以使用 psql 客户端的元命令“\dn”，来查看当前数据库中有哪些模式：

```
universitydb=# \dn
      List of schemas
      Name      |  Owner
----------------+----------
 academicoffice | postgres
 humanresources | postgres
 public         | postgres
(3 rows)
universitydb=#
```

1.13.3 模式是命名空间

模式也称为命名空间，一个数据库的某个模式下的数据库对象不能重名，但是一个数据库的不同模式下的数据库对象可以重名。

可以使用下面的方法来标识一个模式对象：

数据库名.模式名.对象名

因为 PostgreSQL 数据库的每个用户连接只能访问一个数据库，因此在标识一个模式对象时，可以省略数据库名：

模式名.对象名

同一数据库的不同模式下，可以创建同名的数据库对象（如表）。执行下面的 SQL 语句，分别在 universitydb 数据库中的模式 academicoffice、humanresources 和 public 中，创建测试表 person，并插入 1 条数据：

```
universitydb=# CREATE TABLE public.person(name varchar(20) PRIMARY KEY);
universitydb=# INSERT INTO  public.person VALUES('张三');
universitydb=# CREATE TABLE academicoffice.person(name varchar(20) PRIMARY KEY);
universitydb=# INSERT INTO  academicoffice.person VALUES('李四');
universitydb=# CREATE TABLE humanresources.person(name varchar(20) PRIMARY KEY);
universitydb=# INSERT INTO  humanresources.person VALUES('王五');
```

执行下面的 SQL 语句，查看这些新建的表 person 的对象标识（OID）：

```
universitydb=# SELECT pg_class.oid, pg_class.relname
universitydb-# FROM pg_class
universitydb-# JOIN pg_namespace ON pg_class.relnamespace = pg_namespace.oid
universitydb-# WHERE pg_class.relname = 'person' AND
universitydb-#       pg_namespace.nspname = 'public';
  oid  |  relname
-------+------------
 16630 | instructor
(1 row)
```

```
universitydb=# SELECT pg_class.oid, pg_class.relname
universitydb-# FROM pg_class
universitydb-# JOIN pg_namespace ON pg_class.relnamespace = pg_namespace.oid
universitydb-# WHERE pg_class.relname = 'person' AND
universitydb-#       pg_namespace.nspname = 'academicoffice';
  oid  |  relname
-------+------------
 16635 | instructor
(1 row)
universitydb=# SELECT pg_class.oid, pg_class.relname
universitydb-# FROM pg_class
universitydb-# JOIN pg_namespace ON pg_class.relnamespace = pg_namespace.oid
universitydb-# WHERE pg_class.relname = 'person'  AND
universitydb-#       pg_namespace.nspname = 'humanresources';
  oid  |  relname
-------+------------
 16640 | instructor
(1 row)
universitydb=#
```

可以看到，这三个名为 person 的表，对象标识不一样！

执行下面的 SQL 语句，可以看出数据库 universitydb 的不同模式下的同名表 person 的内容：

```
universitydb=# SELECT * FROM public.person;
col                          （使用模式名.表名来指定一个表）
------
 张三
(1 row)
universitydb=# SELECT * FROM academicoffice.person;
name                         （使用模式名.表名来指定一个表）

------
 李四
(1 row)
universitydb=# SELECT * FROM universitydb.humanresources.person;
name                         （使用数据库名.模式名.表名来指定一个表）
------
 王五
(1 row)
universitydb=#
```

1.13.4 模式搜索路径 SEARCH_PATH

一个数据库可以有多个模式，如果用户 SQL 中的数据库对象名前面没有指定模式名的话，那么将按照系统配置参数 SEARCH_PATH 指定的搜索顺序，定位诸如表、索引、系列

等数据库对象。

```
universitydb=# SHOW SEARCH_PATH;
   search_path
-----------------
 "$user", public
(1 row)
universitydb=#
```

配置参数 SEARCH_PATH 的默认值是“$user”,public，其含义是：查找一个数据库对象时，将首先在与数据库用户同名的模式下查找；如果不存与数据库用户同名的模式，或者在这个同名模式下没有找到这个指定名字的数据库对象，则转到模式 public 下去查找；如果在 public 模式也找不到，则向用户返回错误信息。下面的例子证明了这一点：

```
universitydb=# -- 创建与登录用户 postgres 同名的模式 postgres。
universitydb=# CREATE SCHEMA postgres;
universitydb=# -- 在模式 postgres 中创建表 person，并插入一条记录。
universitydb=# CREATE TABLE postgres.person(name varchar(20) PRIMARY KEY);
universitydb=# INSERT INTO  postgres.person VALUES('赵六');
universitydb=# -- 查看不指定模式名时的 person 表。
universitydb=# SELECT * FROM person;
 name                    说明：由于存在与连接的数据库用户 postgres 同名的模式 postgres，
------                         因此访问的表是 postgres.person。
 赵六
(1 row)
universitydb=# -- 删除与连接的数据库用户 postgres 的同名模式 postgres
universitydb=# DROP SCHEMA postgres CASCADE;
NOTICE:  drop cascades to table person
DROP SCHEMA
universitydb=# -- 查看不指定模式名时的 person 表。
universitydb=# SELECT * FROM person;
name                    说明：由于删除了与数据库用户同名的模式，
------                         因此访问的表是 public.person。
 张三
(1 row)
universitydb=# --删除 public 模式下的表 person。
universitydb=# DROP TABLE public.person;
DROP TABLE
universitydb=# SELECT * FROM person;
ERROR:  relation "person" does not exist
LINE 1: SELECT * FROM person;
                      ^
universitydb=#
(无法在模式搜索路径下找到任何名字为 mytable 的表，因此报错!)
```

如果在不指定模式名时，想要访问的表位于模式 humanresources 之下，可以进行如下的设置：

```
universitydb=# -- 设置参数 SEARCH_PATH 的值为模式 humanresources。
universitydb=# SET SEARCH_PATH TO humanresources;
SET
universitydb=# SHOW SEARCH_PATH;
  search_path
--------------
 humanresources
(1 row)
universitydb=# SELECT * FROM person;
 name
------
 王五
(1 row) （由于当前的搜索模式为 humanresources，因此访问的表是 humanresources.person。）
universitydb=# \q
[postgres@dbsvr ~]$
```

1.13.5 PostgreSQL 数据库集簇的逻辑结构

PostgreSQL 数据库集簇的逻辑层次结构是：数据库集簇、数据库、模式和模式对象（如图 5-2 所示）。

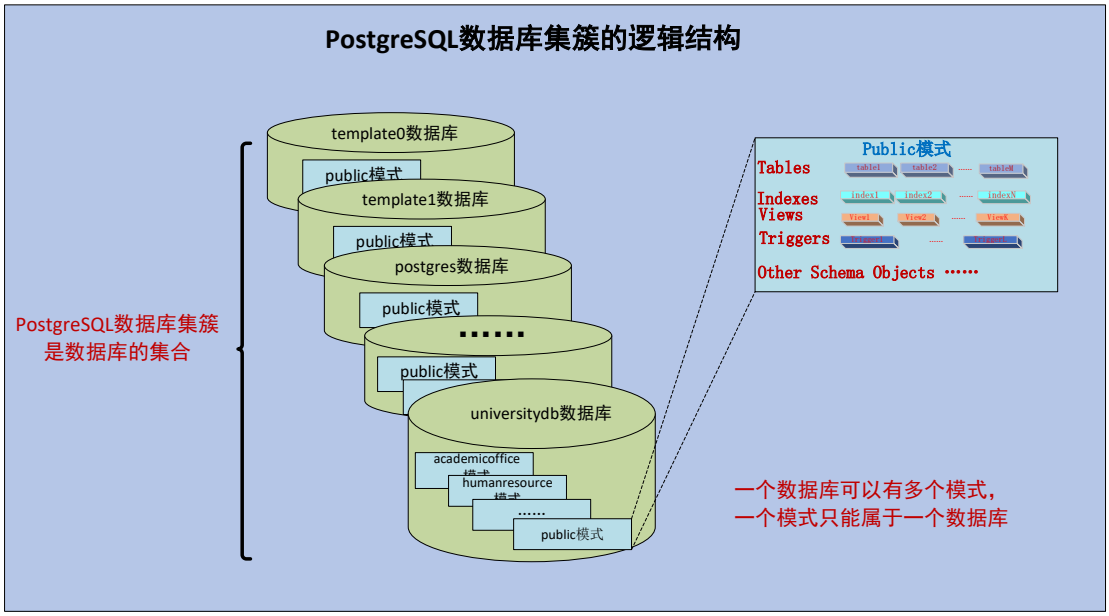


图 5- PostgreSQL 数据库集簇的逻辑结构图

数据库集簇（Database Cluster）是多个用户数据库（Database）的集合。事实上，PostgreSQL 的数据库集簇与数据库之间是一对多的联系：一个 PostgreSQL 数据库集簇中有多个数据库；一个数据库只能属于一个 PostgreSQL 数据库集簇。一个数据库包含多个模式（Schema），一个模式只能属于一个数据库。在一个模式里有许多模式对象（Schema Object），一个模式对

象只能属于一个模式。

这正如在一个大柜子中，有很多抽屉，每个抽屉里有很多收纳盒，收纳盒里有很多物品。数据库集簇相对于大柜子，数据库相当于抽屉，模式相当于抽屉里的收纳盒，模式对象相当于放在收纳盒里的物品。

1.14 表空间

表空间（TABLESPACE）是数据库模式对象的物理容器。

执行下面的命令，恢复有大学应用测试数据集的 PostgreSQL 数据库集簇：

```
[postgres@dbsvr ~]$  cd /opt/db
[postgres@dbsvr db]$ pg_ctl stop
[postgres@dbsvr db]$ rm -rf userdb
[postgres@dbsvr db]$ tar xf userdbWithData.tar
[postgres@dbsvr db]$ pg_ctl start
```

1.14.1 表空间是一个操作系统目录

PostgreSQL 的表空间实际上是一个操作系统目录。如果一个数据库在某个表空间中保存有数据库对象，那么在这个代表表空间的操作系统目录下，会有一个以该数据库的 OID 为名字的子目录。

执行下面的命令，查看当前 PostgreSQL 数据库集簇有哪些表空间：

```
[postgres@dbsvr ~]$ psql -d postgres -U postgres
postgres=# \db
       List of tablespaces
    Name    |  Owner   | Location
------------+----------+----------
 pg_default | postgres |
 pg_global  | postgres |
(2 rows)
postgres=# \q
[postgres@dbsvr db]$
```

可以看到，当前 PostgreSQL 数据库集簇有 2 个表空间：pg_deafult 和 pg_global，其中：

- 表空间pg_default是系统默认的表空间，位于数据目录$PGDATA下的base子目录：

```
[postgres@dbsvr ~]$    cd /opt/db/userdb/u00/
[postgres@dbsvr u00]$  cd pgdata/base/
[postgres@dbsvr base]$ ls -l
total 24
drwx------. 2 postgres dba  4096 Apr 15 20:22 1
drwx------. 2 postgres dba  4096 Apr 15 12:47 13051
```

```
drwx------. 2 postgres dba  4096 Apr 15 21:09 13052
drwx------. 2 postgres dba 12288 Apr 15 20:22 16384
[postgres@dbsvr base]$ oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  13052       postgres  pg_default
  13051      template0  pg_default
      1      template1  pg_default
  16384   universitydb  pg_default
[postgres@dbsvr base]$
```

在代表表空间 pg_default 的目录下,有 4 个以数据库的 OID 为名字的子目录，表示在表空间 pg_default 中，存储了这 4 个数据库的数据。

执行下面的查询，查看表空间 pg_default 存储了哪些数据库的数据：

```
[postgres@dbsvr base]$ psql -d postgres -U postgres
postgres=# SELECT t.spcname AS "Tablespace",
postgres-#        array_to_string(
postgres(#                    ARRAY(
postgres(#                            (
postgres(#                              SELECT datname
postgres(#                              FROM pg_database
postgres(#                              WHERE oid IN (
postgres(#                                SELECT pg_tablespace_databases(t.oid)
postgres(#                                             AS datoid
postgres(#                              )
postgres(#                              ORDER BY 1
postgres(#                            )
postgres(#                           ),
postgres(#                         ','
postgres(#              ) AS "Database(s)"
postgres-#  FROM pg_tablespace t
postgres-#  WHERE t.spcname != 'pg_global'
postgres-#  ORDER BY 1;
 Tablespace |               Database(s)
------------+-----------------------------------------
 pg_default | postgres,template0,template1,universitydb
(1 row)
postgres=# \q
[postgres@dbsvr base]$
```

可以看到，表空间 pg_default 中存储了 postgres、template0、template1 和 universitydb 等数据库的数据。

- 表空间 pg_global 是一个特殊的表空间，位于数据目录$PGDATA 下的 global 子目录，用于存储那些跨数据库共享的全局系统表。

```
[postgres@dbsvr ~]$   cd /opt/db/userdb/u00/
[postgres@dbsvr u00]$ cd pgdata/global/
```

```
[postgres@dbsvr global]$ ls -l
total 564
-rw-------. 1 postgres dba  8192 Apr 15 21:09 1213
-rw-------. 1 postgres dba 24576 Apr 15 12:47 1213_fsm
-rw-------. 1 postgres dba  8192 Apr 15 16:32 1213_vm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 1214
-rw-------. 1 postgres dba 24576 Apr 15 12:47 1214_fsm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 1214_vm
-rw-------. 1 postgres dba 16384 Apr 15 12:47 1232
-rw-------. 1 postgres dba 16384 Apr 15 12:47 1233
-rw-------. 1 postgres dba  8192 Apr 15 12:47 1260
-rw-------. 1 postgres dba 24576 Apr 15 12:47 1260_fsm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 1260_vm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 1261
-rw-------. 1 postgres dba 24576 Apr 15 12:47 1261_fsm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 1261_vm
-rw-------. 1 postgres dba  8192 Apr 15 13:03 1262
-rw-------. 1 postgres dba 24576 Apr 15 12:47 1262_fsm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 1262_vm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 2396
-rw-------. 1 postgres dba 24576 Apr 15 12:47 2396_fsm
-rw-------. 1 postgres dba  8192 Apr 15 12:47 2396_vm
-rw-------. 1 postgres dba 16384 Apr 15 12:47 2397
-rw-------. 1 postgres dba 16384 Apr 15 13:03 2671
-rw-------. 1 postgres dba 16384 Apr 15 13:03 2672
-rw-------. 1 postgres dba 16384 Apr 15 12:47 2676
-rw-------. 1 postgres dba 16384 Apr 15 12:47 2677
-rw-------. 1 postgres dba 16384 Apr 15 12:47 2694
-rw-------. 1 postgres dba 16384 Apr 15 12:47 2695
-rw-------. 1 postgres dba 16384 Apr 15 16:32 2697
-rw-------. 1 postgres dba 16384 Apr 15 16:32 2698
-rw-------. 1 postgres dba     0 Apr 15 12:47 2846
-rw-------. 1 postgres dba  8192 Apr 15 12:47 2847
-rw-------. 1 postgres dba     0 Apr 15 12:47 2964
-rw-------. 1 postgres dba  8192 Apr 15 12:47 2965
-rw-------. 1 postgres dba     0 Apr 15 12:47 2966
-rw-------. 1 postgres dba  8192 Apr 15 12:47 2967
-rw-------. 1 postgres dba     0 Apr 15 12:47 3592
-rw-------. 1 postgres dba  8192 Apr 15 12:47 3593
-rw-------. 1 postgres dba     0 Apr 15 12:47 4060
-rw-------. 1 postgres dba  8192 Apr 15 12:47 4061
-rw-------. 1 postgres dba     0 Apr 15 12:47 4175
-rw-------. 1 postgres dba  8192 Apr 15 12:47 4176
-rw-------. 1 postgres dba     0 Apr 15 12:47 4177
-rw-------. 1 postgres dba  8192 Apr 15 12:47 4178
-rw-------. 1 postgres dba     0 Apr 15 12:47 4181
-rw-------. 1 postgres dba  8192 Apr 15 12:47 4182
-rw-------. 1 postgres dba     0 Apr 15 12:47 4183
-rw-------. 1 postgres dba  8192 Apr 15 12:47 4184
-rw-------. 1 postgres dba     0 Apr 15 12:47 4185
-rw-------. 1 postgres dba  8192 Apr 15 12:47 4186
```

```
-rw-------. 1 postgres dba     0 Apr 15 12:47 6000
-rw-------. 1 postgres dba  8192 Apr 15 12:47 6001
-rw-------. 1 postgres dba  8192 Apr 15 12:47 6002
-rw-------. 1 postgres dba     0 Apr 15 12:47 6100
-rw-------. 1 postgres dba  8192 Apr 15 12:47 6114
-rw-------. 1 postgres dba  8192 Apr 15 12:47 6115
-rw-------. 1 postgres dba  8192 Apr 15 21:14 pg_control
-rw-------. 1 postgres dba   512 Apr 15 12:47 pg_filenode.map
-rw-------. 1 postgres dba 23840 Apr 15 20:22 pg_internal.init
[postgres@dbsvr global]$
```

可以使用下面的命令，来查看该目录下 OID 值对应的数据库对象名字：

```
[postgres@dbsvr global]$ oid2name -d postgres -U postgres -o 1213
From database "postgres":
  Filenode     Table Name
----------------------------
      1213  pg_tablespace
[postgres@dbsvr global]$ oid2name -d postgres -U postgres -o 1260
From database "postgres":
  Filenode  Table Name
----------------------
      1260   pg_authid
[postgres@dbsvr global]$ oid2name -d postgres -U postgres -o 1262
From database "postgres":
  Filenode   Table Name
-----------------------
      1262  pg_database
[postgres@dbsvr global]$
```

以上列出的是系统表。

- 在表空间 pg_global 中存储的信息，不是特定于某个数据库的，而是对整个 PostgreSQL 数据库集簇的所有数据库，都是可见和可用的。如角色（用户）信息、表空间信息等。这些信息需要跨多个数据库可用，因此不能放在特定于数据库的表空间中。
- 不可直接操作：与用户定义的表空间不同，pg_global 和其他内置表空间（如 pg_default）不能通过标准的 SQL 命令进行删除或更改。这些表空间的管理和维护由 PostgreSQL 系统自身负责。实际上，直接修改这些内部系统表可能会导致数据库不稳定或不可用。
- 表空间 pg_global 所在的$PGDATA/global 子目录，还有控制文件：

```
[postgres@dbsvr ~]$ cd /opt/db/userdb/u00
[postgres@dbsvr u00]$ cd pgdata/global/
```

```
[postgres@dbsvr global]$ ls -l pg_control
-rw-------. 1 postgres dba 8192 Apr 15 14:57 pg_control
[postgres@dbsvr global]$
```

1.14.2 表空间和数据库的关系

下面，我们将在操作系统目录/opt/db/userdb/u01/pgdata/userts/data_ts 下新建一个名叫 data_ts 的表空间，并查看表空间 data_ts 的 OID：

```
[postgres@dbsvr ~]$ psql -d postgres -U postgres
postgres=# \! mkdir -p /opt/db/userdb/u01/pgdata/userts/data_ts
postgres=# CREATE TABLESPACE data_ts LOCATION '/opt/db/userdb/u01/pgdata/userts/data_ts';
CREATE TABLESPACE
postgres=# SELECT oid, spcname FROM pg_tablespace WHERE spcname = 'data_ts';
  oid  | spcname
-------+---------
 16505 | data_ts
(1 row)
postgres=#
```

下面我们来观察一下这个新建的表空间 data_ts：

```
postgres=# \! ls -l /opt/db/userdb/u00/pgdata/pg_tblspc
total 0
lrwxrwxrwx. 1 postgres dba 40 Apr 15 16:31 16505 -> /opt/db/userdb/u01/pgdata/userts/data_ts
postgres=# \! oid2name -s
All tablespaces:
    Oid  Tablespace Name
-----------------------
   1663       pg_default
   1664        pg_global
  16505          data_ts
postgres=#
```

可以看到，当我们新创建表空间 data_ts 时，将会在 PostgreSQL 数据库的数据目录 /opt/db/userdb/u00/pgdata 的子目录 pg_tblspc 下，创建一个以表空间 data_ts 的 OID 值“16505”为名字的符号链接，它指向了表空间实际存储的目录/opt/db/userdb/u01/pgdata/userts/data_ts。

继续执行下面的命令，查看操作系统目录/opt/db/userdb/u01/pgdata/userts/data_ts 的信息：

```
postgres=# \! ls -l /opt/db/userdb/u01/pgdata/userts/data_ts
total 4
drwx------. 2 postgres dba 4096 Apr 15 16:31 PG_14_202107181
postgres=# \! ls -l /opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181
total 0
postgres=#
```

可以看到，在创建表空间 data_ts 时，在指定的目录/opt/db/userdb/u01/pgdata/userts/data_ts 下，

会创建一个名叫 PG_14_202107181 的子目录，它是表空间 data_ts 实际对应的操作系统目录。由于目前还未在这个新创建的表空间中存储任何数据，因此该目录现在还是一个空目录！

在表空间 data_ts 为数据库 postgres 创建一个表 my_large_table，然后继续观察表空间 data_ts：

```
postgres=# -- 在数据库 postgres 中创建表 my_large_table
postgres=# CREATE TABLE my_large_table ( id SERIAL PRIMARY KEY,
postgres-#                                data char(1024))  TABLESPACE data_ts;
CREATE TABLE
postgres=# -- 查看表 my_large_table 的 OID
postgres=# SELECT oid, relname FROM pg_class WHERE relname = 'my_large_table';
  oid  |    relname
-------+---------------
 16507 | my_large_table
(1 row)
postgres=# \! oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  13052       postgres  pg_default   数据库 postgres 的 OID 是 13052
  13051      template0  pg_default
      1      template1  pg_default
  16384   universitydb  pg_default
postgres=# \! ls -l /opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181
total 4
drwx------. 2 postgres dba 4096 Apr 15 16:43 13052
postgres=# \! ls -l /opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052
total 8
-rw-------. 1 postgres dba    0 Apr 15 16:43 16507
-rw-------. 1 postgres dba    0 Apr 15 16:43 16511
-rw-------. 1 postgres dba 8192 Apr 15 16:43 16512
postgres=# \! oid2name -d postgres -o 16507
From database "postgres":
  Filenode      Table Name
--------------------------
     16507  my_large_table
postgres=# \! oid2name -d postgres -o 16511
From database "postgres":
  Filenode      Table Name
--------------------------
     16511  pg_toast_16507
postgres=# \! oid2name -d postgres -o 16512
From database "postgres":
  Filenode           Table Name
-------------------------------
     16512  pg_toast_16507_index
postgres=#
```

说明：

创建表 my_large_table 时，表的 data 列是一个宽字符列，因此为其自动创建了一个 TOAST（The Oversized-Attribute Storage Technique）表，其 OID 是 16511。TOAST 表用于存储大型字段数据，使得主表能够更有效地管理它的行。此外，还为 TOAST 表创建了一个索引，其 OID 是 16512。

可以看到，在表空间 data_ts 为数据库 postgres 创建一个表 my_large_table，将在代表表空间 data_ts 的操作系统目录/opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181 下，创建一个以数据库 postgres 的 OID 的值（OID=13052）为名字的子目录，在这个子目录下，以文件的形式，存储了表 my_large_table，文件名是表的 OID 值 16507。

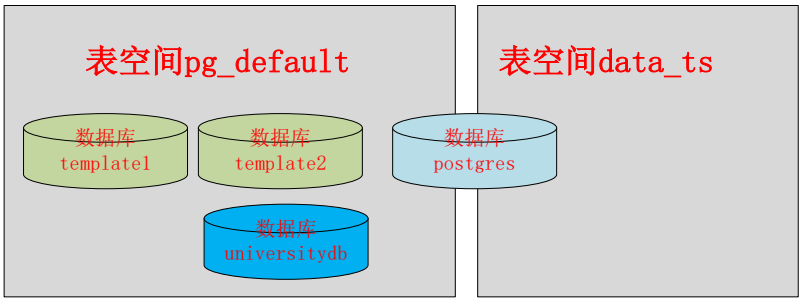
执行下面的 SQL 语句，查看数据库 postgres 的数据，存储在哪些表空间里：

```
postgres=# SELECT DISTINCT t.spcname
postgres-# FROM pg_tablespace t
postgres-# WHERE t.oid IN (
postgres(#     -- 查询数据库默认表空间
postgres(#     SELECT dattablespace
postgres(#     FROM pg_database
postgres(#     WHERE datname = 'postgres'
postgres(#     UNION
postgres(#     -- 查询数据库中对象所使用的表空间
postgres(#     SELECT c.reltablespace
postgres(#     FROM pg_class c
postgres(#     JOIN pg_namespace n ON c.relnamespace = n.oid
postgres(#     WHERE n.nspname NOT IN ('pg_catalog', 'information_schema')
postgres(#     AND c.reltablespace != 0
postgres(#     UNION
postgres(#     -- 查询索引所使用的表空间
postgres(#     SELECT i.indexrelid::regclass::oid
postgres(#     FROM pg_index i
postgres(#     JOIN pg_class c ON i.indrelid = c.oid
postgres(#     JOIN pg_namespace n ON c.relnamespace = n.oid
postgres(#     WHERE n.nspname NOT IN ('pg_catalog', 'information_schema')
postgres(# )
postgres-# AND t.spcname != 'pg_global'
postgres-# ORDER BY t.spcname;
  spcname
------------
 data_ts
 pg_default
(2 rows)
postgres=#
```

可以看到，数据库 postgres 的数据，存储在 data_ts 和 pg_default 这两个表空间中。

总结以上，在 PostgreSQL 数据库中，一个数据库的模式对象可以存储在多个表空间中

（数据库 postgres 的数据存储在表空间 pg_default 和 data_ts 中）；一个表空间可以存储多个数据库的数据（表空间 pg_default 存储了模板数据库 template0 和 template1、数据库 universitydb 的全部数据，还有数据库 postgres 的一部分数据。）



1.14.3 观察表空间中表的物理存储

数据库 postgres 的表 my_large_table 存储在表空间 data_ts。

执行下面的命令和 SQL 语句，查看表 my_large_table 的 OID：

```
postgres=# SELECT oid, relname FROM pg_class WHERE relname = 'my_large_table';
  oid  |    relname
-------+---------------
 16507 | my_large_table
(1 row)
postgres=# \! oid2name -o 16507
From database "postgres":
  Filenode      Table Name
------------------------
     16507  my_large_table
postgres=# \! ls -l /opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052
total 8
-rw-------. 1 postgres dba    0 Apr 15 16:43 16507
-rw-------. 1 postgres dba    0 Apr 15 16:43 16511
-rw-------. 1 postgres dba 8192 Apr 15 16:43 16512
postgres=#
```

创建一个存储过程 fill_my_large_table(n INT)，为表 my_large_table 插入 n 条数据：

```
postgres=# -- 创建一个存储过程，往表 my_large_table 插入 n 行数据
postgres=# CREATE OR REPLACE PROCEDURE fill_my_large_table(n INT)
postgres-# LANGUAGE plpgsql
postgres-# AS $$
postgres$# BEGIN
postgres$#     FOR i IN 1..n LOOP
```

```
postgres$#         -- 使用LPAD填充字符以满足char(1024)的要求
postgres$#         INSERT INTO my_large_table (data) VALUES (LPAD('', 1024, 'a'));
postgres$#     END LOOP;
postgres$# END;
postgres$# $$;
CREATE PROCEDURE
postgres=#
```

执行下面的命令，插入 40 万条记录后，查看表 my_large_table 的大小：

```
postgres=# --插入40万条
postgres=# CALL fill_my_large_table(400000);
CALL
postgres=# -- 查看有40万条数据时表my_large_table的大小
postgres=# SELECT pg_size_pretty(pg_total_relation_size('my_large_table'));
 pg_size_pretty
----------------
 455 MB
(1 row)
postgres=# \! ls -lh /opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052
total 447M
-rw-------. 1 postgres dba 447M Apr 16 03:26 16507
-rw-------. 1 postgres dba 136K Apr 16 03:26 16507_fsm
-rw-------. 1 postgres dba 8.0K Apr 16 03:26 16507_vm
-rw-------. 1 postgres dba    0 Apr 16 03:02 16511
-rw-------. 1 postgres dba 8.0K Apr 16 03:02 16512
postgres=#
```

可以看到，有 40 万条记录的表 my_large_table：

- 文件/opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052/16507 是主数据文件，大小约为 447MB。
- 文件/opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052/16507_fsm 是 FSM（Free Space Map，自由空间映射图）文件，用于跟踪表中的空闲空间，以便 PostgreSQL 可以有效地插入新的行或更新现有行，其大小目前为 136KB。
- VM 文件/opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052/16386_vm 是 VM（Visibility Map，可见性映射图）文件，用于跟踪表的哪些页面是全部可见的，这有助于加速索引扫描和减少全表扫描的需要，其大小目前为 8KB。

执行下面的命令，继续插入 200 万条记录后，再查看表 my_large_table 的大小：

```
postgres=# --插入200万条
postgres=# CALL fill_my_large_table(2000000);
CALL
postgres=# -- 查看有240万条数据时表my_large_table的大小
postgres=# SELECT pg_size_pretty(pg_total_relation_size('my_large_table'));
```

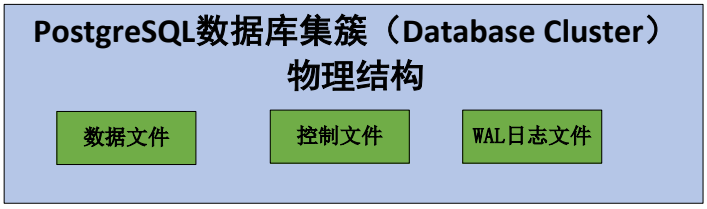
```
 pg_size_pretty
----------------
 2731 MB
(1 row)
postgres=# \! ls -lh /opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052
total 2.7G
-rw-------. 1 postgres dba 1.0G Apr 15 20:27 16507
-rw-------. 1 postgres dba 1.0G Apr 15 20:27 16507.1
-rw-------. 1 postgres dba 631M Apr 15 20:28 16507.2
-rw-------. 1 postgres dba 696K Apr 15 20:27 16507_fsm
-rw-------. 1 postgres dba  16K Apr 15 20:26 16507_vm
-rw-------. 1 postgres dba    0 Apr 15 16:43 16511
-rw-------. 1 postgres dba 8.0K Apr 15 16:43 16512
total 2.7G
postgres=#
```

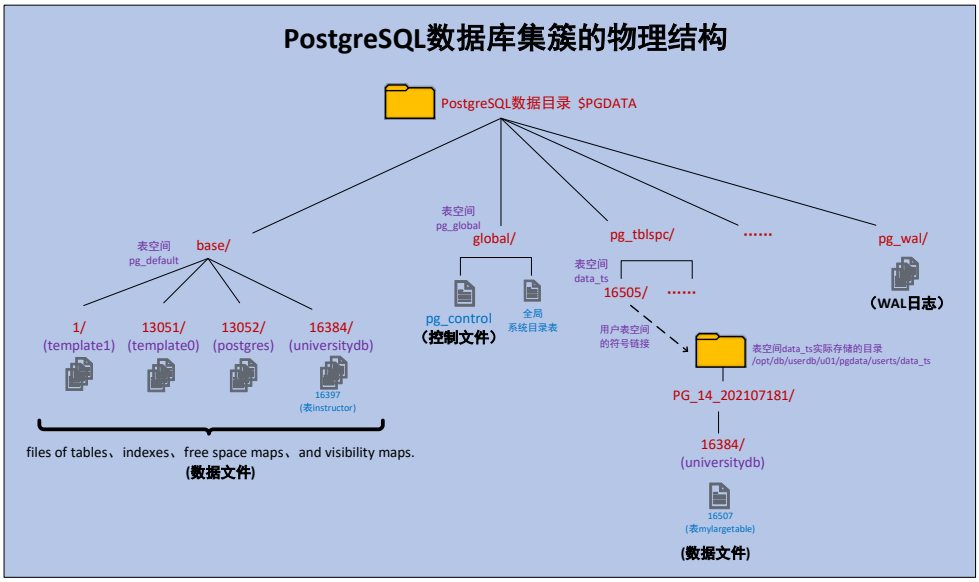
可以看到：

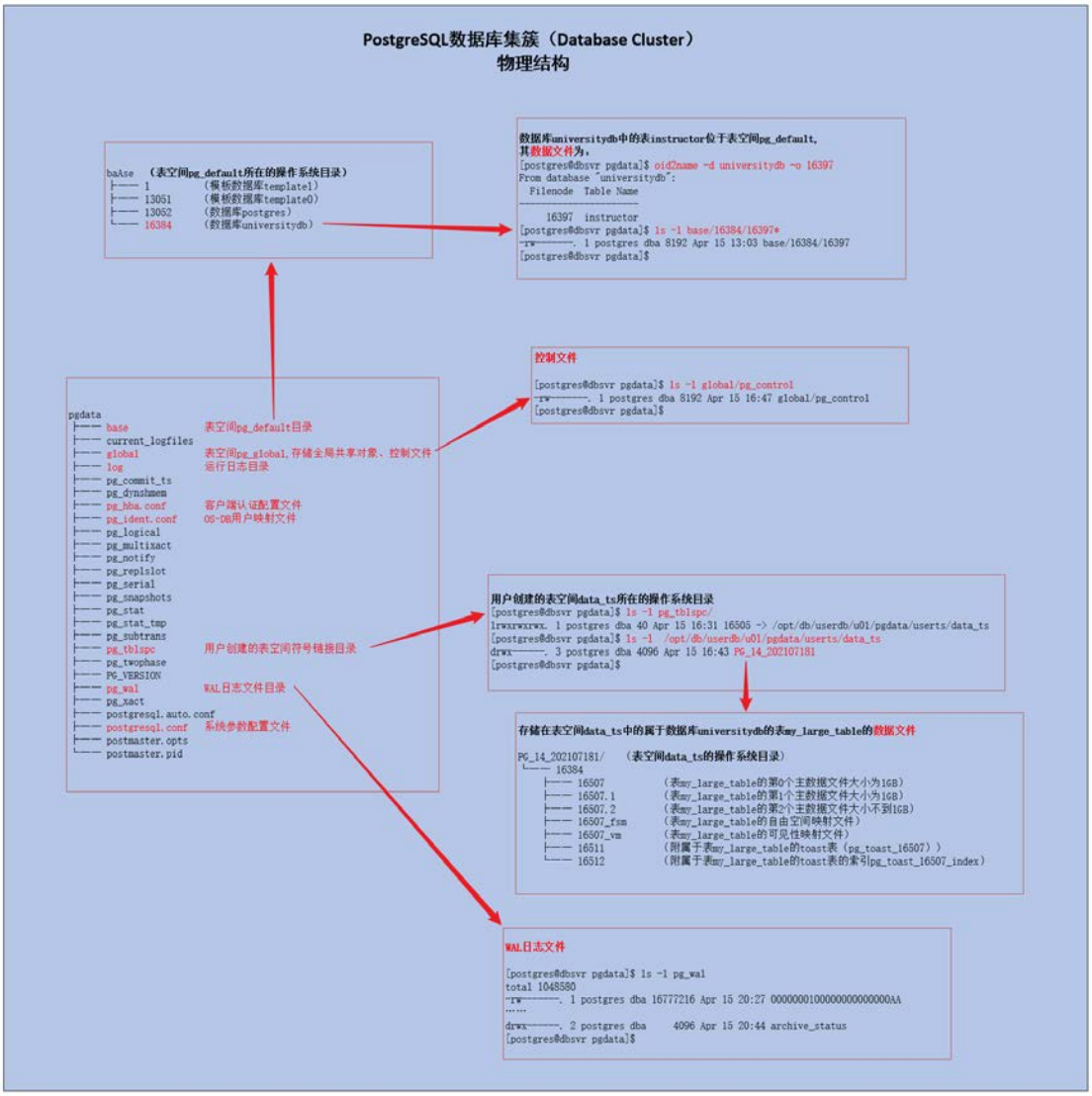
- 有 240 万条记录的表 my_large_table，占用了 2731MB 存储空间，主数据文件有 3 个：2 个为 1GB 大小，另外 1 个为 631MB。

1.14.4 PostgreSQL 数据库集簇的物理结构

到目前为止，PostgreSQL 数据库集簇的物理结构如下图所示：







1.14.5 观察删除一个表

删除 1 个表，将删除这个表对应的所有文件，同时还会删除该表的其他附属文件。下面的示例演示了这一点。

执行下面的命令，删除表 my_large_table，表 my_large_table 对应的操作系统文件将被删除掉：

```
postgres=# DROP TABLE my_large_table;
DROP TABLE
postgres=# \! ls -lh /opt/db/userdb/u01/pgdata/userts/data_ts/PG_14_202107181/13052
total 0
```

```
-rw-------. 1 postgres dba 0 Apr 15 21:06 16507
-rw-------. 1 postgres dba 0 Apr 15 21:06 16511
-rw-------. 1 postgres dba 0 Apr 15 21:06 16512
```

说明：表 my_large_table 的所有数据文件大小都置为 0 了，但是数据文件暂时还未被删除，需要稍等一会儿（可能是几分钟），直到执行了下一次检查点操作，表对应的文件才会才会被删除！

如果想马上删除表对应的数据文件，可以手工发出一个检查点。

```
postgres=# CHECKPOINT;
CHECKPOINT
postgres=# \! ls -lh /opt/db/userdb/u00/pgdata/userts/data_ts/PG_14_202107181/13052
total 0
postgres=#
```

1.14.6 删除表空间

不能删除一个存储有数据的表空间。下面的测试证明了这一点：

```
postgres=# -- 在数据库 postgres 中创建表 my_large_table
postgres=# CREATE TABLE my_large_table ( id SERIAL PRIMARY KEY,
postgres-#                               data char(1024))  TABLESPACE data_ts;
CREATE TABLE
postgres=# DROP TABLESPACE data_ts;
ERROR:  tablespace "data_ts" is not empty
postgres=#
```

执行下面的 SQL 语句，查看表空间 data_ts 目前保存了哪些数据库的数据：

```
postgres=# SELECT d.datname
postgres-# FROM pg_database d, ( SELECT pg_tablespace_databases(oid) AS datoid
postgres(#                       FROM pg_tablespace t
postgres(#                       WHERE t.spcname='data_ts'
postgres(#                      ) t
postgres-# WHERE t.datoid = d.oid;
 datname
----------
 postgres
(1 row)
postgres=#
```

可以看到，表空间 data_ts 保存了数据库 postgres 的数据。

连接到数据库 postgres 上，执行下面的 SQL 语句，查看在表空间 data_ts 中有哪些表：

```
postgres=# -- 查看数据库在某个特定表空间中有哪些表
postgres=# -- 查看数据库 postgres 在表空间 data_ts 中有哪些表
postgres=# SELECT n.nspname AS schema_name,
postgres-#        c.relname AS table_name
postgres-# FROM pg_class c JOIN pg_namespace n ON c.relnamespace = n.oid
postgres-# WHERE c.reltablespace = (  SELECT oid
```

```
postgres(#                              FROM pg_tablespace
postgres(#                              WHERE spcname = 'data_ts'
postgres(#                             )
postgres-#       AND c.relkind = 'r';
 schema_name |   table_name
-------------+----------------
 public      | my_large_table
(1 row)
postgres=#
```

可以看到，在表空间 data_ts 中，有 postgres 数据库的表 my_large_table。

执行下面的 SQL 语句，将表移动到表空间 pg_default:

```
postgres=# ALTER TABLE my_large_table SET TABLESPACE pg_default ;
ALTER TABLE
postgres=#
```

执行下面的 SQL 语句，再次尝试删除表空间 data_ts：

```
postgres=# DROP TABLESPACE data_ts;
DROP TABLESPACE
postgres=#  \! ls -l /opt/db/userdb/u00/pgdata/pg_tblspc
total 0
postgres=# \! rmdir /opt/db/userdb/u01/pgdata/userts/data_ts
postgres=# \q
[postgres@dbsvr ~]$
```

可以看到，这一次我们成功地删除了表空间 data_ts。请注意，删除表空间时，还需要将为表空间 data_ts 提前创建的操作系统目录也删除，避免在后期，对 PostgreSQL 数据库服务器的运维，造成混乱（一堆的目录，都不知道干什么用的，也不知道能不能删除这些目录！）。

1.15 系统配置文件

严格地说，PostgreSQL 数据库配置文件不是 PostgreSQL 数据库集簇的物理组成部分。但是，PostgreSQL 数据库配置文件定制了 PostgreSQL 数据库的运行特征，对 PostgreSQL 数据库的高效运行至关重要。主要有以下三种配置文件：

- 系统配置参数文件 postgresql.conf
- 客户端访问控制文件 pg_hba.conf
- 本地操作系统用户与数据库用户的身份映射文件 pg_ident.conf

执行下面的 SQL 语句，查看 PostgreSQL 配置文件的位置：

```
postgres=#  SELECT name, setting
postgres-#  FROM pg_settings
postgres-#  WHERE name IN ('config_file','hba_file','ident_file');
```

```
    name     |               setting
-------------+-------------------------------------
config_file |/opt/db/userdb/pgdata/postgresql.conf 启动参数文件
hba_file    |/opt/db/userdb/pgdata/pg_hba.conf     客户端访问控制文件
ident_file  |/opt/db/userdb/pgdata/pg_ident.conf   操作系统用户与数据库用户的身份映射文件
(3 rows)
postgres=#
```

1.16 PostgreSQL 服务器的体系结构图

图 5-1 是 PostgreSQL 服务器的体系结构图。

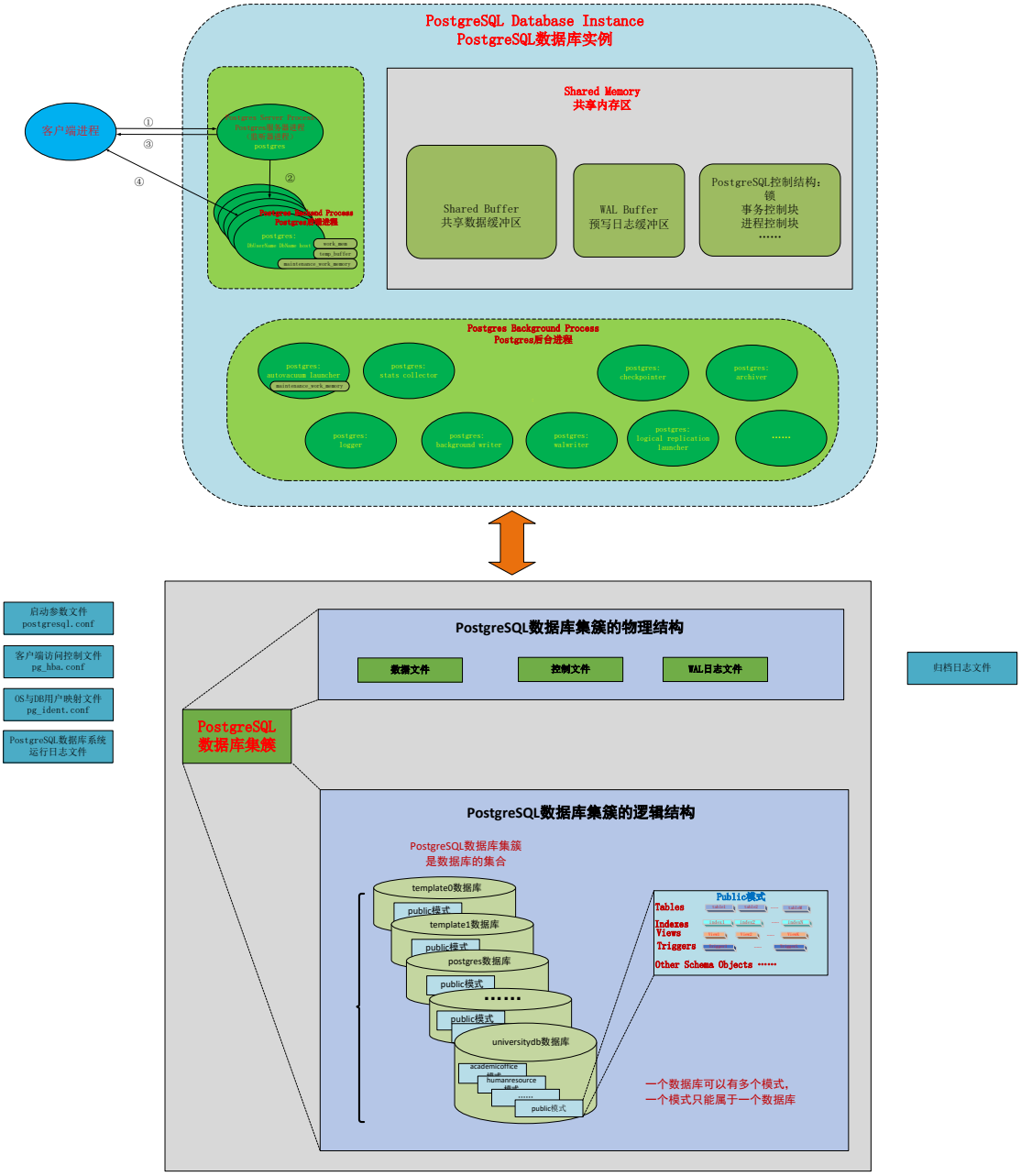
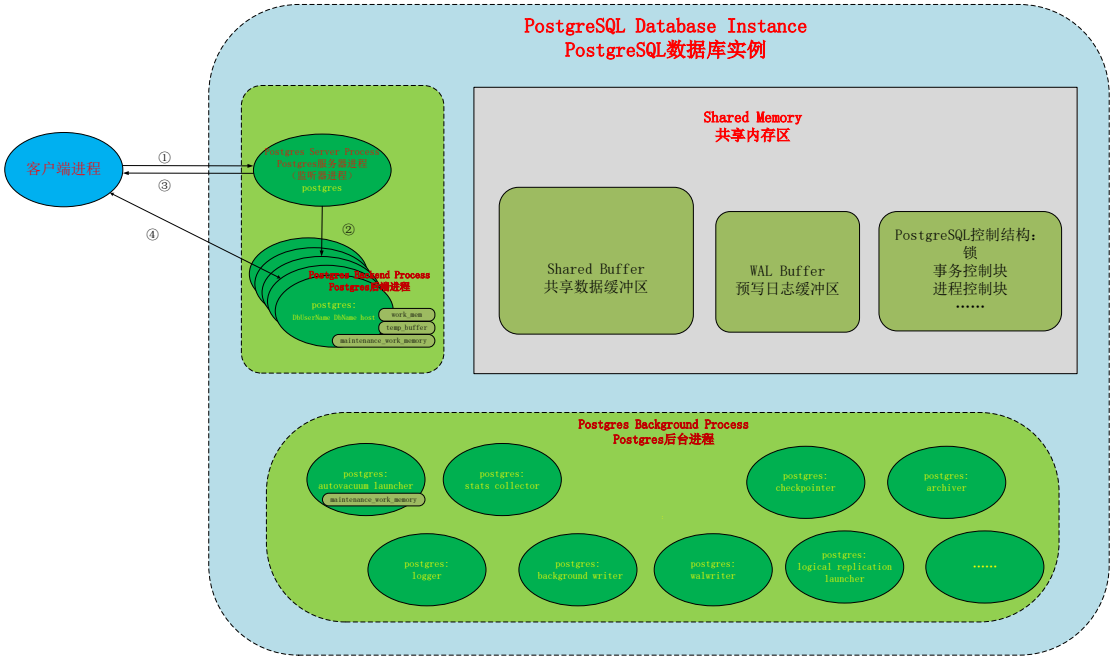


图 5-1　PostgreSQL 服务器的体系结构图

“PostgreSQL 数据库服务器”由 2 部分组成：PostgreSQL 数据库实例和 PostgreSQL 数据库集簇。可以认为，PostgreSQL 数据库服务器是数据库管理系统（DBMS）。

1.17 PostgreSQL 数据库实例

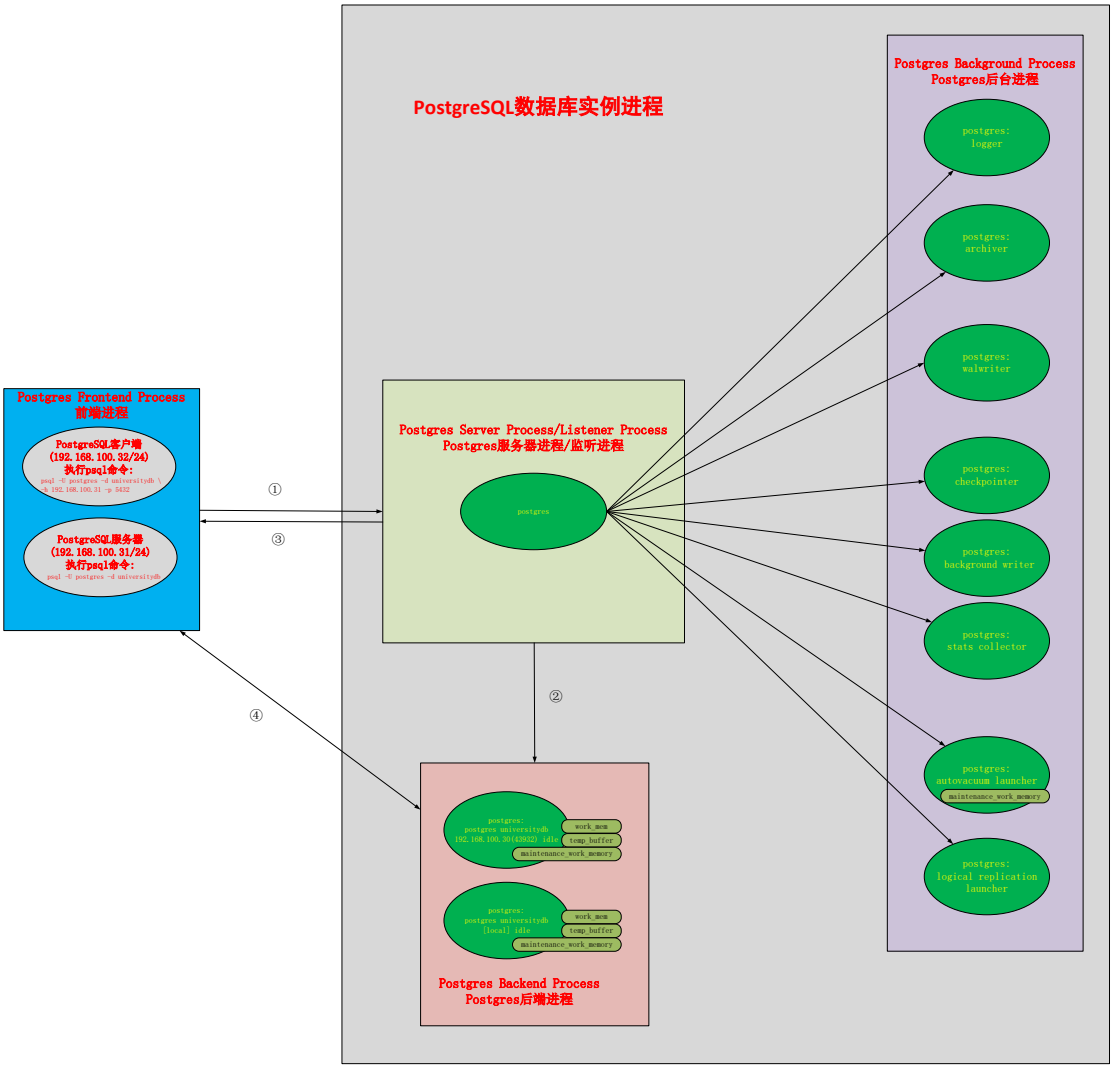
PostgreSQL 数据库实例是 PostgreSQL 服务器在内存中的组件，包括一些内存进程、以及这些进程共享的内存数据结构，如图 1-xxx 所示。



PostgreSQL 数据库服务器是一个多用户数据库管理系统。必须启动一个 **PostgreSQL 数据库实例（Database Instance）**，才能允许多个用户并行地访问驻留在硬盘上的 PostgreSQL 数据库集簇。实际上，启动 PostgreSQL 数据库，就是使用 PostgreSQL 二进制程序，启动一个在内存中运行的 PostgreSQL 数据库实例。

1.17.1 PostgreSQL 数据库实例进程

如图 1-XXX 所示，我们把 PostgreSQL 服务器进程/监听进程、后端进程和后台进程等三类进程，统称为**“PostgreSQL 数据库实例进程”**。



1、postgres 服务器进程/监听进程

启动 PostgreSQL 数据库实例，首先会启动一个名叫“postgres”的进程（PID=2173），它是 PostgreSQL 数据库服务器的 postgres 服务器进程/监听进程，负责完成如下的任务：

（1） 负责启动 PostgreSQL 数据库实例。

- 分配共享内存区，例如数据缓冲区、日志缓冲区等。
- 初始化 PostgreSQL 数据库的内存控制结构。
- 注册信号处理函数。

（2） 启动 PostgreSQL 数据库实例的其它 postgres 后台进程。

（3） PostgreSQL 数据库实例启动完成后，postgres 进程将成为一个监听进程，在接收到客户端的连接请求并通过安全认证后，为客户端分配一个 postgres 后端进程。

（4） 在后端进程出现错误时，负责执行恢复操作（redo 操作）。

（5） 负责关闭 PostgreSQL 数据库。

2、后台进程（Background Process）

PostgreSQL 数据库的后台进程，通常不直接参与 SQL 语句的处理，它们在后台运行，辅助服务进程更高效地处理客户请求，因此它们也被称为后台辅助进程。后台进程具有不同的生命周期：有些后台进程（如 background writer）在服务器启动时自动创建，随服务器关闭而退出；有些后台进程，如 archiver（归档日志进程），是根据系统的配置参数 archive_mode 的值是 on 还是 off，来决定是否启动；还有一些后台进程，如 autovacuum worker（自动清理工作者进程），它是由 autovacuum launcher（自动清空操作启动进程）根据需要启动的，完成任务后就自动退出了。

（1） background writer（后台写进程）

PostgreSQL 的后台写进程（background writer）负责将数据缓冲区中的脏页面（即修改过的页面）写入到磁盘数据文件中。后台写进程把数据缓冲区中的脏页面写回到磁盘的数据文件中，则服务进程在需要数据缓存块时减少写操作，从而加快系统的响应时间。后台写进程随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。

后台写进程周期性把数据缓冲区的脏页面写回到磁盘的数据文件中，写的速度不能太快，也不能太慢。写得太快，会增加系统的 I/O，例如如果缓冲区的数据页面更新了多次，如果每次更新都写到磁盘上，会增加系统的 I/O 负担。如果写得太慢，则服务进程需要缓冲区页面时，可能需要自己将脏页面写到磁盘，从而增加系统的响应时间。

在 PostgreSQL 的参数配置文件 postgresql.conf 中有多个以”bgwriter_”开头的系统配置参数，配置了后台写进程的行为，例如：

- 参数 bgwriter_delay 设置后台写进程的启动周期，默认值是 200ms（毫秒）；
- 参数 bgwriter_lru_maxpages 设置每次从内存写出到数据文件的最大页数，默认值为 100。

（2） walwriter（WAL 日志写进程）

PostgreSQL 的 WAL 日志写进程（walwriter）负责将共享内存区中的 WAL 缓冲区写入到磁盘上的 WAL 日志文件中。

当 SQL 语句对数据进行更新时，更新操作在数据缓冲区中进行，同时会将数据变更记录写到 WAL 缓冲区。当事务提交或 WAL 缓冲区充满到一定程度时，WAL 缓冲区的日志项需要写到磁盘上的 WAL 日志文件中。后台日志写进程周期性地运行，将大量的随机写，改善为 WAL 日志的顺序写，减少了磁盘 I/O 操作，从而大大地提高了系统的响应性能。

日志写进程也是随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。用户可以通过修改系统配置参数 wal_writer_delay 来设置该进程的写日志间隔。

（3） checkpointer（检查点进程）

PostgreSQL 的检查点进程（checkpointer）负责处理系统的检查点操作。数据库的检查点是一个事件，当该事件发生时，数据库缓存中的脏缓存块将全部写入数据文件，同时在 WAL 日志文件写入一条日志记录，并对数据库控制文件进行更新。可以看出，检查点是一个非常耗 I/O 资源的动作。

如果 PostgreSQL 数据库服务器非正常关闭，则下次系统重新启动时需要进行数据库系统恢复（Recovery）。恢复的过程如下：首先从控制文件中读取最后一个检查点位置，然后在 WAL 日志中找到最后的检查点，接下来从这里开始进行系统重做（REDO），直到 WAL 日志结束。

检查点进程也是随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。

如果检查点执行间隔太大，当 PostgreSQL 数据库系统崩溃时，会增加系统恢复的时间；反之，如果检查点执行间隔太小，则会增加系统的 I/O，降低系统的吞吐量。用户可以通过修改系统配置参数 checkpoint_timeout 来设置检查点进程执行检查点操作的间隔时间。

（4） stats collector（统计数据收集进程）

PostgreSQL 的统计信息收集进程（stats collector）负责收集 SQL 执行过程中的统计信息，例如在表或索引上进行了进行了多少次增、删、改操作，磁盘的读写次数，元组的读写数量等。这些信息可以帮助数据库管理员 DBA 进行系统性能诊断，找出可能存在的性能瓶颈。

统计信息收集进程也是随数据库服务一起启动，并且在数据库服务的运行过程中一直存在。

收集统计信息会给系统增加负荷，系统配置文件中有以”track_”开头的配置参数，可以

配置它收集哪些信息。

（5） autovacuum launcher（自动清空启动进程）

PostgreSQL 数据库采用多版本并发控制策略，当事务对表中元组进行删除操作时，并不会立刻从表中直接删除这些数据行，而是在元组上打上删除标记。当事务对表中元组进行更新操作时，采用的是删除旧元组，插入新元组的策略，因此在数据库中会保留元组的多个版本。如果没有其他并发事务需要读这些旧元组时，则可以将它们清除，否则数据库占用的存储会越来越大。

PostgreSQL 的自动清理启动进程（autovacuum launcher），负责向主进程 postgres 请求创建自动清理工作进程（autovacuum worker）。那些过时的、并发事务不再需要的 MVCC 旧元组，将由自动清理工作进程负责清除。清理工作结束后，自动清理工作进程将自动退出。

（6） archiver 日志归档进程

数据库的 WAL 日志文件记录了数据库中所有对数据页面的更新，一旦系统出现故障，可以使用 WAL 日志进行故障恢复。系统在运行过程中会一直源源不断地产生日志，当系统执行了一个完整的检查点后，系统故障恢复就不再需要该检查点之前的 WAL 日志了，因此为了防止日志文件的个数不断增长，PostgreSQL 会重用早期不再需要的 WAL 日志文件（修改了 WAL 日志文件的名字，感觉像删除了原来的日志文件）。

但是，如果出现了磁盘介质的故障，则需要使用数据库的备份和自备份以来的所有 WAL 日志，来恢复一个完整的数据库文件。因此用户生产环境的 PostgreSQL 数据库，通常运行在归档模式下，即在 WAL 日志文件被重用之前，将其拷贝到一个特定的地方，这样在将来进行介质故障恢复时，可以使用这些归档的 WAL 日志。

PostgreSQL 的归档日志进程（archiver）负责将 WAL 日志文件拷贝到归档日志目录。数据库在归档模式下运行，才会启动该进程，系统配置参数 archive_mode 设置 PostgreSQL 是否运行在归档模式下，默认是 off（运行在非归档方式）。

（7） logger （运行日志进程）

PostgreSQL 的运行日志进程（logger）负责收集和管理数据库服务器的日志信息。日志信息对于数据库的监控、问题诊断和性能分析至关重要。可以在 PostgreSQL 数据库的系统

配置文件 postgresql.conf 中修改关于日志的参数，来定制这些日志信息。通过维护一个高效和可靠的日志记录系统，logger 进程帮助数据库管理员追踪和分析数据库行为，及时发现和解决问题。

（8） logical replication launcher（逻辑复制启动进程）

PostgreSQL 的逻辑复制启动进程（logical replication launcher）是负责管理逻辑复制功能的后台进程。逻辑复制允许数据库将数据更改从一个系统复制到另一个系统，但与物理复制不同，它允许不同版本的 PostgreSQL 之间的数据复制，以及更细粒度的控制，如只复制选定的表或仅复制数据更改。逻辑复制启动进程监控逻辑复制插槽的创建和管理，以及复制订阅者的管理。当设置了逻辑复制，该进程负责启动和管理必要的工作进程来处理数据的捕捉、传输和应用。这使得逻辑复制启动进程在实现高可用性、灾难恢复和数据分发方面发挥了关键作用，特别是在需要跨版本复制或更复杂的数据分发策略时。

3、后端进程（Backend Process）

客户端用户进程需要访问 PostgreSQL 数据库服务器时，首先需要向服务器发出服务请求，PostgreSQL 数据库服务器上的 postgres 服务器进程/监听收到连接请求并完成用户身份认证后，将为发出请求的用户进程，在 PostgreSQL 服务器上，创建一个后端进程（Backend Process），负责完成这个用户进程的所有的数据库请求任务。当用户进程断开会话连接时，用户进程所对应的后端进程也将自动退出。

后端进程主要完成以下任务：

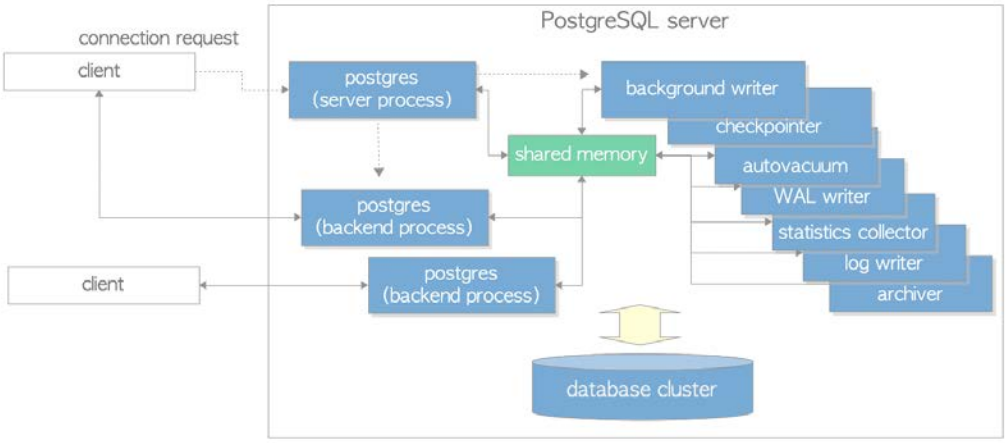
- 解析并执行用户所提交的 SQL 语句。
- 在数据库缓存中查找用户所访问的数据，如果没有，从数据文件中读取。
- 根据所执行的 SQL 语句类别，更新数据或将数据返回给用户进程。

1.17.2 PostgreSQL 数据库实例的内存结构

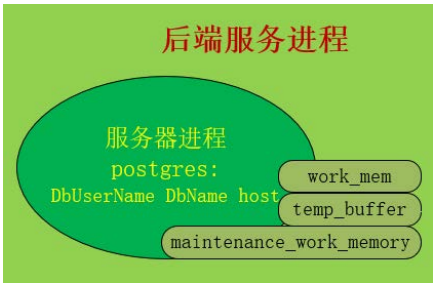
PostgreSQL 数据库使用的内存总体上可以分为两部分：

- **系统全局区（Shared Global Area，SGA）**：SGA 是一组共享内存结构，其中包含一个 PostgreSQL 数据库实例的数据和控制信息。SGA 由 postgres 服务器进程、所

有的后端进程和后台进程共享，如图 1-xxx 所示。它使用的是操作系统的共享内存。



- **程序全局区（Program Global Area, PGA）：**每个 postgres 后端进程都有自己的 PGA，用于存储进程私有数据和控制信息的内存区域，如图 1-xxx 所示，用于内部操作例如排序和哈希操作等可使用的工作内存。

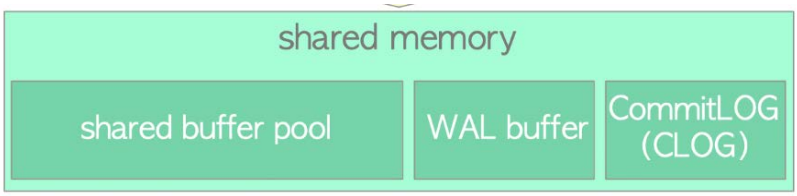


有些 postgres 后台进程，如 autovacuum launcher，会使用 maintainance memory，作为自己的 PGA 内存，如图 1-xxx 所示。

1、系统全局区（SGA）

系统全局区（SGA）是 PostgreSQL 实例的主要组成部分，它保存 PostgreSQL 数据库系统所有进程的共享信息。PostgreSQL 数据库服务器使用系统全局区来实现后台进程间的数据共享。在数据库服务器启动时，为 SGA 分配内存，在终止时，SGA 区释放。

系统全局区主要由数据缓冲区（Shared buffers）、日志缓冲区（WAL buffers）和进程共享的控制结构，例如锁表、进程控制块、事务控制块等组成。



数据缓冲区是系统全局区中最大的一块共享内存，保存最近从数据文件中读取的数据块，以提高数据库的处理性能。理论上来说，数据缓冲区越大越好，但是 PostgreSQL 是运行在操作系统之上，操作系统也是有文件缓存的，因此建议该值配置为系统可用内存的 25%~40%。数据缓冲区大小由系统配置参数 shared_buffers 确定。shared_buffers 的默认值是 128M，通常不能满足生产数据库的需求，DBA 需要根据系统的实际硬件情况和系统负载设置该值。

日志缓冲区用于缓存在对数据进行修改操作的操作过程中生成的 REDO 记录。该缓冲区是一个顺序使用的，循环的结构，当写满时，再从头部开始。由参数 wal_buffers 定义大小，较大的缓冲区可以减少写日志的磁盘 I/O，但是事务提交时日志记录一定要写盘，因此，日志缓冲区不需要太大。wal_buffers 默认值-1，将自动配置 WAL 缓冲区的大小范围：

- 等于 shared_buffers 的 1/32 的尺寸（大约 3%）；
- 不小于 64kB
- 不大于 WAL 日志段的尺寸。

参数 wal_buffers 的默认配置，在大部分情况下，都能满足生产数据库的需要。

系统全局区中的其它数据结构，例如锁表、进程控制块、事务控制块等，都是 PostgreSQL

数据库内部使用的控制结构，以保证多个数据库进程正确运行。

2、程序全局区（PGA）

每个数据库进程，无论是后端进程还是后台进程，工作时都需要自己的内存空间，例如postgres 后端进程在处理 SQL 时需要进行内部排序、创建哈希表、对临时表的访问，后台进程 autovacuum launcher 需要进行数据清理等。这些内存，由进程根据需要向操作系统申请和释放，并由该进程独享。

因为数据库是多用户共享的系统，单个 postgres 后端进程不能耗尽系统的所有资源，因此 PostgreSQL 对每个 postgres 后端进程使用的内存资源是有控制的：

- **maintenance_work_mem**：执行维护性操作时使用的内存。
- **temp_buffer**：访问临时表数据所使用的缓冲区。
- **work_mem**：事务执行内部排序或 Hash 表写入临时文件之前使用的内存缓冲区。

postgres 后端进程在处理 SQL 语句时，经常会有排序或创建哈希表这些非常需要内存的操作，对于单个进程来说，这些操作如果能在内存中进行，性能会更好，但是单个进程不能耗尽所有的内存资源以影响其它进程的正常运行，PostgreSQL 使用配置参数 Work_mem 来设定每个进程可以使用内存数量。Work_mem 设置过小，会造成排序和哈希操作变成外存操作，极大降低系统性能。Work_mem 设置过大，则如果多个进程同时进行排序或哈希时，又会造成系统的内存资源耗尽。Work_mem 的设置需要同时考虑每个进程处理 SQL 的复杂度、系统的并发数以及可用的内存资源总量。

数据库中临时表的数据不是所有 postgres 后端进程共享的，因此后端进程在访问临时表时不使用 SGA 中的数据缓存区，而是把临时表的数据读到自己进程内部的临时缓存区中。配置参数 temp_buffer 设置每个 postgres 后端进程的临时缓冲区可以使用的最大内存空间，默认值是 8M。如果 postgres 后端进程需要访问比较大的临时表时，可以重新设置该值来提升性能。

数据库在做系统维护操作时，例如 VACUUM、CREATE INDEX、REINDEX 等操作时，需要读入大量的数据，在执行这些操作时把配置参数 maintenance_work_mem 设置为较大的值，可以加速这些操作的执行。

进程对私有内存的使用是根据需要逐渐向系统请求分配的，并不是一开始就分配所有的

内存。以上参数的设置值，是这些操作可以使用的最大内存量。这些参数都是会话级的，可以为每个数据库会话单独设置这些参数的值。

1.18 PostgreSQL 数据库的数据字典

数据库管理系统的数据字典，存储数据库中数据的元信息，如表定义，函数定义，系统的统计信息、用户权限信息等。PostgreSQL 数据库管理系统会使用数据字典信息对 SQL 语句进行分析、重写和优化；数据库管理员或用户，可以通过查询数据字典来了解数据库中的数据情况。

PostgreSQL 数据库的数据字典，主要由**系统表**、**系统视图**和**动态视图**构成。通过这些系统表和视图，PostgreSQL 为数据库管理员和开发者提供了强大的工具集来管理、监控和优化数据库的运行和性能。

1.18.1 系统表

系统表：系统表是 PostgreSQL 元数据的核心，有**全局系统表（位于 global 表空间）**和**特定数据库的系统表（位于特定数据库的 pg_catlog 模式）**。

常见的系统表包括：

- pg_class：包含了关于表、索引、视图、序列等数据库对象的信息。
- pg_attribute：存储了表中列的信息，例如列的数据类型和顺序。
- pg_proc：包含了存储过程和函数的信息。
- pg_type：存储了数据库中所有数据类型的信息。
- pg_database：包含了数据库实例中所有数据库的信息。
- pg_namespace：存储了架构（命名空间）的信息，用于组织数据库中的对象。
- pg_index：存储了索引的信息，例如索引覆盖的列和它所属的表。
- pg_constraint：包含了关于数据库中所有表约束的信息，包括主键、外键和检查约束等。
- pg_trigger：存储了触发器的信息。
- pg_tablespace：存储了表空间的信息，表空间用于定义数据库文件的物理存储位置。
- pg_extension：包含了已安装扩展的信息。

- pg_foreign_table：存储了外部表的信息，外部表用于访问存储在 PostgreSQL 外部的数据。
- pg_foreign_server：包含了外部服务器的信息，外部服务器表示可以访问外部数据的服务器。
- pg_foreign_data_wrapper：存储了外部数据封装器的信息，外部数据封装器用于定义访问外部数据的接口。

1.18.2 系统视图

PostgreSQL 提供了多个系统视图，这些视图简化了对系统表数据的查询，使得获取数据库元数据更为方便。这些系统视图覆盖了从用户和权限、表和列的定义，到数据库性能统计等多个方面的信息。以下是一些主要的系统视图及其描述：

- 基本信息视图
  - pg_tables：列出数据库中的所有表，包括表名和所属架构。
  - pg_views：显示数据库中所有视图的信息，包括视图的名称和定义。
  - pg_indexes：展示数据库中所有索引的详细信息，包括索引所属的表和索引的定义。
  - pg_sequences：列出数据库中所有序列的信息。
- 安全和权限相关视图
  - pg_roles：列出数据库中定义的所有角色（用户和组）。
  - pg_group：展示数据库用户组的信息。
  - pg_user：提供了数据库用户的信息。
- 性能监控相关视图
  - pg_stat_activity：显示当前数据库活动，包括执行的查询和连接的状态。
  - pg_stat_user_tables：提供用户表的访问统计信息。
  - pg_stat_user_indexes：展示用户定义的索引的使用情况和性能统计。
  - pg_statio_user_tables：提供用户表的 I/O 统计信息。
  - pg_statio_user_indexes：显示用户索引的 I/O 统计信息。
- 信息模式（INFORMATION_SCHEMA）视图

  INFORMATION_SCHEMA 提供了一组 SQL 标准的视图，这些视图对于获取

跨不同数据库系统的元数据信息非常有用。这些视图隐藏了底层系统表的复杂性，提供了一种标准化的方式来查询数据库对象的元数据。

- information_schema.tables：提供关于数据库中表的信息，符合 SQL 标准。
- information_schema.columns：列出表中的列及其数据类型等信息。
- information_schema.constraints：显示数据库中的约束，包括主键、外键等。

- 扩展和外部数据相关视图
  - pg_available_extensions：列出数据库可以安装的扩展。
  - pg_extension：显示已安装扩展的信息。
  - pg_foreign_table：展示外部表的信息。
  - pg_foreign_server：列出配置的外部服务器信息。
  - pg_foreign_data_wrapper：提供外部数据封装器的信息。

以上视图提供了一个从高层次概述数据库对象和性能的窗口，使得数据库的管理和监控工作更加直观和高效。利用这些视图，数据库管理员和开发人员可以方便地访问数据库的元数据，进行诊断和性能优化等工作。

1.18.3 动态视图

动态视图基于从数据库内存结构生成的虚拟表，不是存储在数据库中的常规表，其数据取决于数据库和实例的状态。由于动态性能视图在数据库运行过程中会不断动态更新，所以不需要保证视图的读一致性。

动态视图（也称为管理视图）主要用于监控和诊断数据库的性能和健康状况。PostgreSQL 提供了几个动态视图来帮助数据库管理员和开发者获取运行时信息，如：

- pg_stat_activity：显示了当前数据库中所有活动会话的信息，包括每个会话正在执行的查询。
- pg_stat_user_tables：提供了用户表的访问和性能统计信息。
- pg_stat_user_indexes：显示用户索引的使用情况和统计。
- pg_statio_user_tables：显示用户表的 I/O 统计信息。
- pg_statio_user_indexes：显示用户索引的 I/O 统计信息。
- pg_indexes：显示数据库中的索引。
- pg_views：显示数据库中的视图。

- pg_locks：显示当前持有的锁，帮助识别锁竞争问题。

我们可以使用动态视图来查看数据库的当前活动情况，如系统和会话参数、内存使用和分配、CPU 和 IO 使用情况、文件状态、会话和事务进度等。这些视图在进行数据库性能分析和问题诊断时非常有用。通过查询这些视图，可以获得关于数据库当前状态和历史性能的深入洞察。

从存储结构上看，PostgreSQL 的系统表与普通的用户表相同，只是用户不能直接通过 INSERT、UPDATE、DELETE 语句，修改这些系统表和系统视图，只能查询它们的信息。

在执行 DDL 语句时，PostgreSQL 数据库会更新系统表来记录数据库对象的定义；执行 vacuum 或 analyze 命令时，系统会更新系统表或系统视图中的统计信息。

PostgreSQL 数据库的数据字典信息，保存在模式 pg_catalog 和模式 information_schema 中。

1.18.4 pg_catalog 模式

PostgreSQL 数据库的系统表和系统视图，被组织在 pg_catalog 模式中，它们存储了关于数据库本身的元数据，包括数据库中的表、列、数据类型、函数、索引以及权限等信息。

基本上，pg_catalog 包含了 PostgreSQL 数据库的“数据字典”。执行下面的 SQL 语句，可以查询 pg_catalog 模式中的系统表和系统视图的列表：

```
postgres=# SELECT tablename FROM pg_catalog.pg_tables WHERE schemaname = 'pg_catalog';
       tablename
------------------------
 pg_statistic
（省略了输出）
 (62 rows)
postgres=#
```

可以看到，pg_catalog 模式中系统表和系统视图，均以“pg_”开头命名。

以下是数据库用户经常查询的系统表和系统视图：

- pg_attribute：存储 KingbaseES 中表或索引的属性信息。
- pg_class：存储 KingbaseES 中表或索引等数据库对象的信息。
- pg_database：存储 KingbaseES 中数据库的信息。
- pg_index：存储关于索引的信息。

- pg_namespace：存储 KingbaseES 中数据库模式的信息。
- pg_proc：存储 KingbaseES 中函数或存储过程信息。
- pg_tablespace：存储 KingbaseES 中表空间信息。
- pg_user 和 pg_roles：存储关于用户和角色的信息。

动态视图基于从数据库内存结构生成的虚拟表，不是存储在数据库中的常规表，其数据取决于数据库和实例的状态。由于动态性能视图在数据库运行过程中会不断动态更新，所以不需要保证视图的读一致性。

以下是数据库用户经常查询的动态视图::

- pg_stat_activity：提供关于当前活动会话的信息。
- pg_stat_user_tables：显示用户表的访问统计。
- pg_stat_user_indexes：显示用户索引的使用情况和统计。
- pg_statio_user_tables：显示用户表的 I/O 统计信息。
- pg_statio_user_indexes：显示用户索引的 I/O 统计信息。
- pg_tables：显示数据库中的表。
- pg_indexes：显示数据库中的索引。
- pg_views：显示数据库中的视图。
- pg_locks：显示当前持有的锁。

我们可以使用动态视图来查看数据库的当前活动情况，如系统和会话参数、内存使用和分配、CPU 和 IO 使用情况、文件状态、会话和事务进度等。

1.18.5 information_schema 模式

PostgreSQL 数据库实现了 ANSI（American National Standards Institute）标准定义的 information_schema 模式，它用于提供关于数据库的元数据信息，包括表、列、数据类型、视图、约束等。information_schema 提供了一个标准化的方式来查询数据库的结构信息，这意味着同样的查询可以在支持 SQL 标准的任何数据库系统上运行，而无需担心数据库的内部实现细节。

ANSI 标准的 information_schema 模式具有以下特点：

- 标准化：information_schema 遵循 SQL 标准，因此其结构和用法在所有遵循标准

的数据库系统中都是一致的。这样，开发人员可以使用相同的查询语句来获取不同数据库系统的结构信息。

- 只读视图：information_schema 包含的是一系列只读视图，这些视图提供了数据库对象的信息。由于它们是只读的，因此你不能通过修改这些视图来改变数据库结构。
- 跨数据库兼容性：使用 information_schema 可以让数据库的查询更加可移植。当你正从一个数据库系统迁移到另一个时，使用 information_schema 可以简化查询的迁移过程，因为你不需要重写这些查询来适应新系统的系统表。

information_schema 模式包含了许多视图，下面是其中的一些重要视图：

- TABLES：提供数据库中所有表的信息，包括表名、表类型（基表或视图）、所属模式等。
- COLUMNS：提供表中所有列的详细信息，包括列名、数据类型、是否允许 NULL、默认值等。
- VIEWS：列出数据库中所有视图的信息，包括视图名和视图定义。
- TABLE_CONSTRAINTS：显示表的约束信息，如主键、外键、唯一约束等。
- KEY_COLUMN_USAGE：提供约束键和列之间的映射关系信息。

information_schema 在多种场景下非常有用，包括：

- **数据库审计：**了解数据库的结构，包括表、列和约束的使用情况。
- **动态生成 SQL：**在编写能够适应不同数据库结构的应用程序时，可以查询 information_schema 来动态地生成 SQL 语句。
- **数据库迁移和文档化：**获取数据库的结构信息，帮助在不同的数据库系统之间迁移或记录数据库的结构。

执行下面的 SQL 语句，可以查询 information_schema 模式中的系统表和系统视图的列表：

```
postgres=# SELECT table_name FROM information_schema.tables
postgres=# WHERE table_schema = 'information_schema';
             table_name
------------------------------------
```

```
 information_schema_catalog_name
（省略了输出）
(69 rows)
postgres=#
```

1.18.6 数据字典查询示例

例 1：使用动态视图 pg_tables，列出当前访问的数据库的所有的表：

```
universitydb=# SELECT schemaname, tablename
universitydb-# FROM pg_tables
universitydb-# WHERE schemaname NOT IN ('pg_catalog', 'information_schema');
   schemaname    | tablename
-----------------+-----------
 public          | time_slot
 public          | department
 public          | course
 public          | instructor
 public          | section
 public          | classroom
 public          | teaches
 public          | student
 public          | takes
 public          | advisor
 public          | prereq
 academicoffice | person
 humanresources | person
(13 rows)
universitydb=#
```

列出当前访问的数据库的所有的表的表名、OID、模式名、属主和表空间

```
universitydb=# SELECT
universitydb-#   c.oid AS "OID",
universitydb-#   c.relname AS "Table Name",
universitydb-#   n.nspname AS "Schema Name",
universitydb-#   a.rolname AS "Owner",
universitydb-#   coalesce(ts.spcname, 'pg_default') AS "Tablespace"
universitydb-# FROM
universitydb-#   pg_class c
universitydb-#   JOIN pg_namespace n ON c.relnamespace = n.oid
universitydb-#   JOIN pg_authid a ON c.relowner = a.oid
universitydb-#   LEFT JOIN pg_tablespace ts ON c.reltablespace = ts.oid
universitydb-# WHERE
universitydb-#   c.relkind = 'r'  -- r 表示普通表
universitydb-#   AND n.nspname NOT IN ('pg_catalog', 'information_schema');
  OID  | Table Name |  Schema Name   |  Owner   | Tablespace
-------+------------+----------------+----------+------------
 16415 | time_slot  | public         | postgres | pg_default
```

```
 16394 | department | public         | postgres | pg_default
 16391 | course     | public         | postgres | pg_default
 16397 | instructor | public         | postgres | pg_default
 16403 | section    | public         | postgres | pg_default
 16388 | classroom  | public         | postgres | pg_default
 16412 | teaches    | public         | postgres | pg_default
 16406 | student    | public         | postgres | pg_default
 16409 | takes      | public         | postgres | pg_default
 16385 | advisor    | public         | postgres | pg_default
 16400 | prereq     | public         | postgres | pg_default
 16634 | person     | academicoffice | postgres | pg_default
 16639 | person     | humanresources | postgres | pg_default
(13 rows)
universitydb=#
```

查看系统当前所有会话的信息：

```
universitydb=# \x
Expanded display is on.
universitydb=# SELECT  pid,datname AS database_name,usename AS username,
universitydb-#         application_name,client_addr AS client_address,
universitydb-#         client_hostname,client_port,backend_start,xact_start,
universitydb-#         query_start,state_change,state,query
universitydb-# FROM pg_stat_activity;
（省略了许多输出）
-[ RECORD 3 ]----+--------------------------------
pid              | 90065
database_name    | universitydb
username         | postgres
application_name | psql
client_address   |
client_hostname  |
client_port      | -1
backend_start    | 2024-04-16 15:28:15.965085+08
xact_start       | 2024-04-16 15:59:42.963795+08
query_start      | 2024-04-16 15:59:42.963795+08
state_change     | 2024-04-16 15:59:42.963798+08
state            | active
query            | SELECT                        +
                 |   pid,                        +
                 |   datname AS database_name,    +
                 |   usename AS username,         +
                 |   application_name,            +
                 |   client_addr AS client_address,+
                 |   client_hostname,             +
                 |   client_port,                 +
                 |   backend_start,               +
                 |   xact_start,                  +
                 |   query_start,                 +
                 |   state_change,                +
```

```
                    |   state,                          +
                    |   query                           +
                    | FROM                              +
                    |   pg_stat_activity;
（省略了许多输出）
universitydb=# \q
[postgres@dbsvr ~]$
```

1.19 SQL 语句的执行过程

SQL 语句的正确执行需要数据库服务器的多个组件互相配合，例如数据缓存区、日志缓冲区、后台进程等。用户执行 SQL 语句首先需要与数据库实例建立连接，这时数据库服务器就会创建一个专用的服务进程为该用户服务。

根据不同类型的 SQL 语句，服务进程使用不同的组件：

- SELECT 语句需要把结果集返回给用户进程
- DML 语句需要日志系统
- COMMIT 语句必须保证一个事务可以恢复

有些数据库后台进程并不参与 SQL 处理，只是为了提高数据库性能。

SELECT 语句执行过程如图 4-所示：

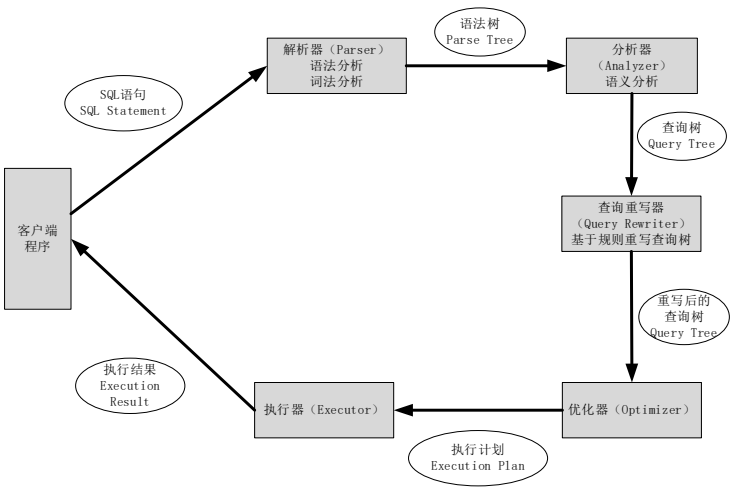


图 4- SQL 语句执行过程图

当客户端用户进程发出SQL语句，后端服务进程接收到SQL语句后，调用解析器（Parser）进行语法分析和词法分析，生成一棵语法树；语法树又被转交给分析器（Analyzer），进行

语义分析，生成查询树；查询树被转交给查询重写器，基于规则进行查询重写，生成了一棵重写后的查询树；接下来将重写后的查询树转交给优化器（Optimizer），生成一个高度优化的执行计划；执行计划转交给执行器（Executor），产生最终结果，由后端服务进程将查询结果转交给客户端进程。

DML 语句的执行过程与 SELECT 语句的执行过程基本类似，区别在于 DML 语句不需要向客户端用户进程返回数据，不过需要后端服务进程向客户端用户进程返回 DML 语句的执行是否成功的信息。

下面是执行 DML 语句的关键步骤：

（1） 服务进程接收到 SQL 语句后，如果是 DML，看需要的数据是否在缓冲区，如果在，直接执行（3），否则执行（2）。

（2） 从数据库文件中将数据调入数据缓冲区。

（3） 完成缓冲区中数据的修改。

（4） 在 REDO 日志中生成该操作的日志，记录数据的修改，写到日志缓冲区

（5） 给用户进程返回 SQL 语句的执行情况，成功还是失败，修改的数据元组数。

如果服务进程接收到 COMMIT 语句，则生成一条事务提交日志记录写到日志缓存区，并把该日志之前所有的日志都写到日志文件，给用户进程返回 SQL 语句的执行情况。