第3章

关系数据库的管理和查询

openGauss是一个开源关系数据库管理系统，与Mulan PSL v2一起发布，内核基于华为在数据库领域多年的经验，并持续提供针对企业级场景定制的竞争功能。

目前openGauss支持的运行平台是Linux系统，而读者自己的计算机一般安装的是Windows或者macOS系统。要学习openGauss，最简单的方法是在本机上通过Docker这个虚拟平台来运行它，也可以通过申请一个云主机的方式在云端Linux主机上运行它。本章首先简单介绍Docker这个虚拟平台的作用和基本使用方法，然后分别讲解在本机和云主机上如何安装和配置openGauss数据库。

3.1 关系数据库openGauss的安装和配置方法

3.1.1 Docker平台简介

Docker平台能方便开发者进行跨平台的应用程序开发和部署，它可以解决Windows平台上的开发者想使用openGauss数据库的问题。openGauss目前不支持Windows操作系统，只能运行在Linux操作系统上。此时，通过Docker的轻量化虚拟功能，我们能够在Windows操作系统中创建一个微型的Linux容器，它很小，对系统的开销占用很小，但可以让openGauss运行。一旦我们让openGauss运行在这个Linux容器中，就可以在本地通过容器的开放端口来访问和操作openGauss数据库。与传统的虚拟机技术相比，Docker的容器技术更加轻量化，对系统资源的占用更少，运行速度一般也更快，其应用不限于数据库方面的开发，也正广泛地应用在各类应用程序的开发和部署上。Docker的好处是多方面的，本书限于篇幅，就不展开讲述了。

Docker中的核心概念是镜像和容器。镜像可以被类比为安装程序，容器可以被类比为安装好并可以随时启动运行的程序。Docker平台有一个镜像在线商城，我们可以从该商城下载需要的镜像，安装该镜像后，容器就形成了。容器可以随时被启动运行，也可以随时被停止。容器就是一个包含运行时环境的大型程序，我们即将使用的openGauss数据库就将运行在这样的容器中。

3.1.2 Docker的安装方法

1. 在Linux平台的安装方法

Docker在Linux平台上的安装非常简单，一般通过一条命令就可以安装。以Ubuntu

Linux 20.04 系统为例：

```
sudo snap install docker
```

通过该命令，就可以在线安装 Docker 平台。等待下载并安装后，应该可以看到“安装成功”字样的提示。运行下面的命令，查看所安装的 Docker 版本。

```
Docker -- version
```

笔者在写作的时候其版本为 Docker version 20.10.17。

2. 在 Windows 平台的安装方法

读者可以在 Docker 官方网站（网址详见前言二维码）下载新的 Docker Desktop 安装程序。下载完毕后，双击安装程序进行安装，建议以默认选项进行安装。如果没有遇到错误，则可以运行安装好的 Docker Desktop 程序 Docker Desktop。双击该图标启动 Docker Desktop 程序后，会发现在任务栏右下角多出了一个图标，右击该图标，会弹出一个菜单，如图 3-1 所示。

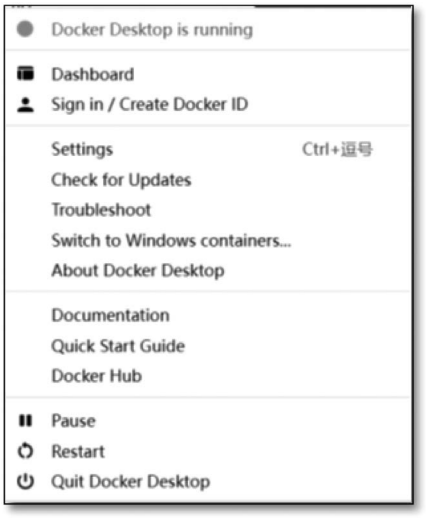


图 3-1　Docker 的系统菜单截图

如果能够在菜单最上方看到图 3-1 中的 Docker Desktop is running 字样，则说明安装成功了，否则系统会给出错误提示，请按照具体提示内容进行操作。一般的错误与 Windows 操作系统缺少某些必要的系统组件有关，如 WSL2 组件。用户可以在出错提示中找到相关链接，以了解具体的解决办法。这里就不再赘述了。

安装成功后，系统会弹出一个窗口，即 Docker Desktop 窗口，如图 3-2 所示。

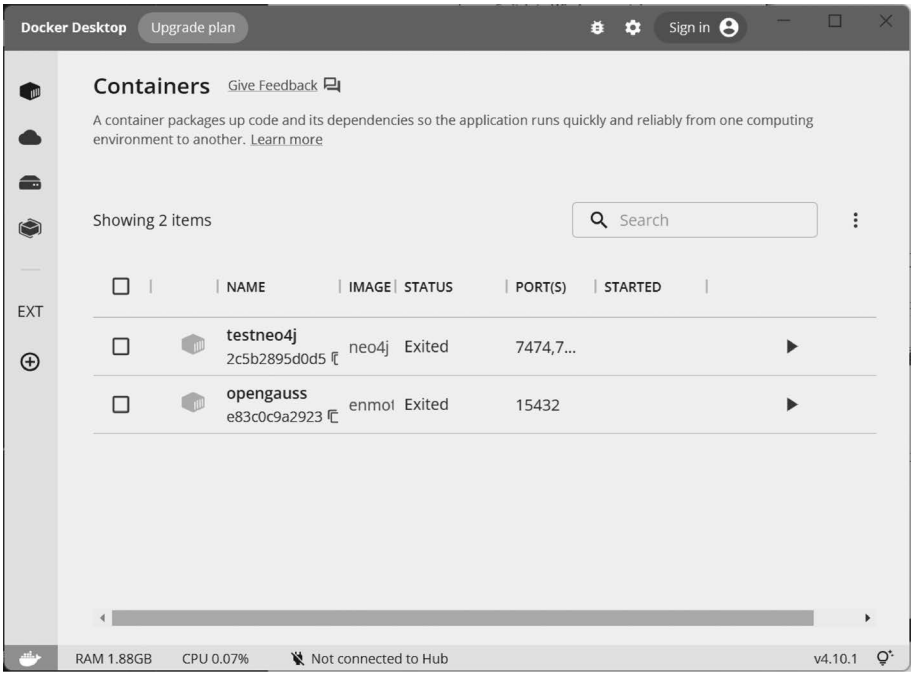


图 3-2　Docker Desktop 窗口截图

图 3-2 显示已经安装了两个容器，一个容器的名字中有 opengauss 字样，另一个容器的名字中有 neo4j 字样。这是本书将会讲解的两个数据库。对于新安装 Docker Desktop 的用户而言，这个窗口中的容器列表应该是空的。

3.1.3 在 Docker 中拉取 openGauss 数据库镜像

在笔者写作时，华为公司并没有给 openGauss 发布官方的镜像，但由于 openGauss 是开源的，因此相关企业制作了比较好的镜像。我们可以从 Docker 平台下载这个镜像。镜像的下载又常被称为“拉取”，后面我们一般用拉取这个词。

如果是在 Linux 操作系统中，那么只需在命令行中运行下面的语句就可以了：

```
docker pull enmotech/opengauss:latest
```

其中，pull 的意思是拉取镜像，后面的字符串“enmotech/opengauss”是这个镜像的名字，“:latest”的意思是拉取新版本的镜像。

如果是在 Windows 操作系统中，用户可以仿照上面的方法，打开一个命令行窗口，输入同样的语句就可以了。如果没有显示出错提示，则可以打开 Docker Desktop 窗口，切换到 Images（镜像）标签页（单击窗口左侧代表镜像的图标 Images），此时可以看到拉取的镜像显示在列表中，如图 3-3 所示。

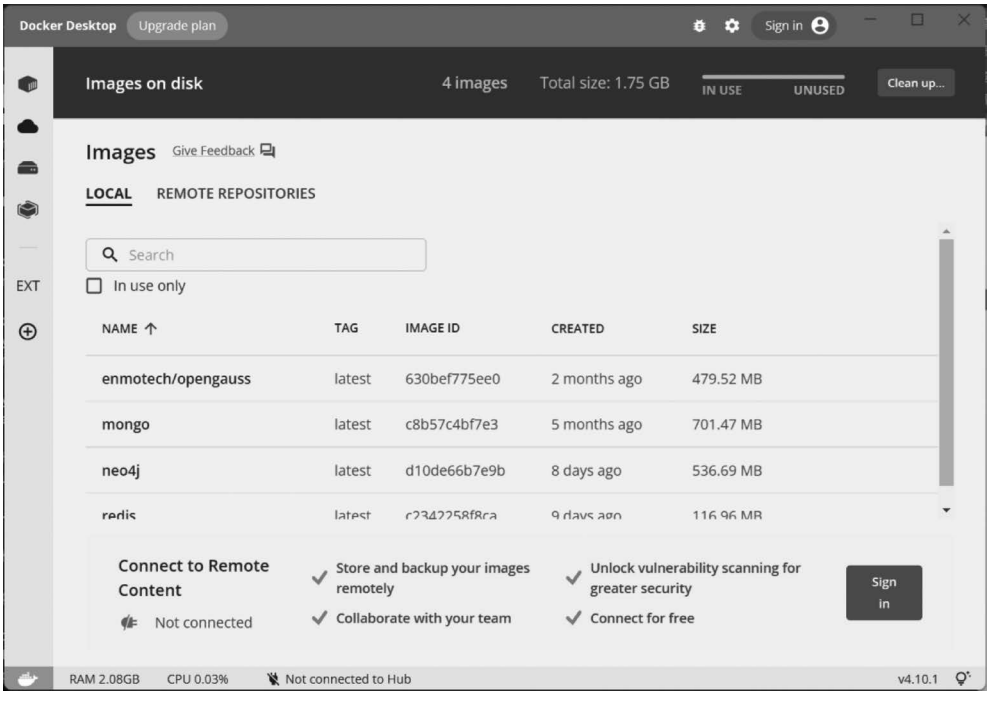


图 3-3 Docker 镜像列表截图

从图 3-3 中可见，作者除已经拉取 openGauss 数据库的镜像外，还拉取了 MongoDB、Neo4j 和 Redis 等数据库的镜像，这些数据库都将在本书中讲解。

3.1.4 安装运行 openGauss 容器

一旦拉取了 openGauss 的镜像，就可以安装并运行对应的容器了。容器一旦被启动，我们就可以操作 openGauss 的数据库系统来练习关系数据库的使用了。

在 Linux 操作系统中，可以在命令行输入下面的语句：

```
docker run --name opengauss --privileged=true -d -p 15432:5432 enmotech/opengauss:latest
```

其中，docker run 是指要安装并启动运行一个容器。“--name opengauss”是指容器的名字是 opengauss。必须给容器起个名字，以便对这个容器进行启停操作。“--privileged＝true -d”是与容器的运行权限有关的参数，暂时不需要理解，照着写就行。“-p 15432:5432”用于指定这个容器在运行时内部和外部通信的端口，其中 5432 是容器内部的通信端口，15432 是这个容器对外的通信端口。这个容器就像一个独立运行的虚拟主机，运行在这个主机内部的程序，都通过 5432 这个内部端口发送信息；而运行在容器外面的程序，如以后要在 Windows 平台上开发的 Python 程序，就得通过 15432 这个外部端口来跟容器内部的程序（比如 openGauss）通信。“enmotech/opengauss:latest”表示要安装并运行的容器名称。

如果这条命令没有返回出错信息，那么恭喜你，你的 openGauss 数据库容器已经在运行了，在这个容器中有一个微型的 Linux 系统，里面安装并运行着 openGauss 和它所需的所有依赖软件。可以通过编程的方法，通过 15432 这个外部端口来访问容器中的 openGauss 数据库（见 3.4 节），也可以通过带有图形界面的专用软件来访问容器中的 openGauss 数据库（见 3.3 节）。

在 Windows 操作系统中，也可以仿照上面的做法，打开一个命令行窗口，输入上面的指令。如果指令运行成功，可以打开 Docker Desktop 窗口，切换到 Containers（容器）标签页（单击窗口左侧代表容器的图标 Containers），此时可以看到安装的容器显示在列表中，如图 3-4 所示。

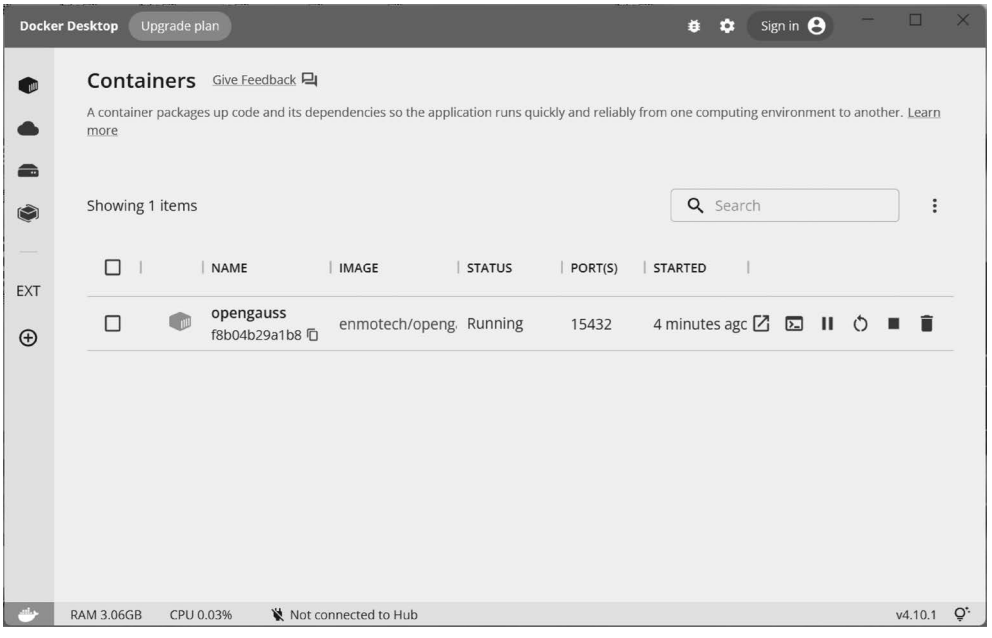


图 3-4　Docker 中的容器列表截图

图 3-4 中显示了一个名为 opengauss 的容器，状态是 Running，对外端口是 15432。这说明该容器已经成功运行了。

可以随时关闭容器的运行，方法是单击容器列表项右侧的关闭按钮 ■。如果要再次启动容器，则单击容器列表项右侧的运行按钮 ▶。这个运行按钮在容器关闭状态时会显示出来。

除通过图形界面的方式外，还可以通过命令行方式来启动和关闭容器。比如启动一个已

经安装好的容器，方法如下：

```
docker start opengauss
```

这条语句启动了一个安装好的名为 opengauss 的容器。如果要关闭一个运行中的容器，方法如下：

```
docker stop opengauss
```

这条语句关闭了一个正在运行的名为 opengauss 的容器。

3.2 关系数据库 openGauss 的基本设置

虽然通过前面的操作，该容器已经在运行了，但里面运行的只是一个 openGauss 管理程序。由于 openGauss 默认的权限限制，目前无法从容器外部通过编程方式或者图形化程序来访问 openGauss 数据库。

在开启关系数据库之旅前，必须花点时间熟悉 openGauss 的基本操作，包括创建用户、分配权限、显示现有数据库、显示用户权限等。这些基本操作需要以命令行方式运行，所以无论是在 Linux 还是在 Windows 平台上，做法是类似的。

3.2.1 进入容器内部

要想对 openGauss 数据库进行配置，必须先进入容器内部，可以用命令行方式，如果是在 Windows 操作系统中，也可以用图形界面的方式。首先来看如何用命令行的方式进入一个正在运行中的容器（确保要进入的容器已经启动运行了），方法是在命令行窗口中输入这条 Docker 语句：

```
docker exec -it opengauss bash
```

其中，docker exec 的意思是进入一个运行中的容器；“-it”的意思是进入容器后开启一个交互式的会话环境，以便通过输入命令来配置程序；opengauss 是指要进入一个名为 opengauss 的容器；bash 也是必须写的参数，具体意思暂不必深究。

如果这条语句运行成功，可以看到命令行的提示符改变了，会呈现类似图 3-5 的形式。

```
root@f8b04b29a1b8:/#
```

图 3-5 Docker 容器命令行截图

其中，root 是指在这个基于 Linux 的容器内部，当前的用户名是 root；“@”符号后面的一串字符是当前容器中虚拟主机的名字，基本不用这个名字；“#”字符后面就是闪烁的光标，等待输入命令，以便在容器中运行。这说明已经进入容器内部的 Linux 环境了。如果想退出容器，则可在“#”字符后输入 exit 命令。

如果是在 Windows 操作系统下运行 Docker 的，也可以通过 Docker Desktop 窗口程序进入容器。方法是在容器标签页中，在一个已经运行的容器右侧寻找一个按钮 ，单击这个按钮，系统就会弹出一个命令行窗口，并自动进入容器，而且可以看到“#”提示符，等待输入指令。

3.2.2 登录 openGauss 数据库

在这个包含 openGauss 的容器内部，运行着一个微型的 Linux 操作系统，这个操作系统已

经内置了一个管理员用户 root，还内置了一个专门用于操作 openGauss 的用户 omm。首先用一条 Linux 命令从默认的 root 用户切换到这个 omm 用户，以便以 omm 的身份进行后续的 openGauss 操作。切换到 omm 用户的方法如下：

```
su - omm
```

注意："-"左侧和右侧各有一个空格。

一旦切换成功，会看到 Linux 的命令行提示符变成类似图 3-6 的样子。

其中，omm 代表现在的身份是 omm 这个用户；"@"后面的字符串代表主机名；"$"是新的命令提示字符，可以在这个字符后面输入命令。

```
omm@f8b04b29a1b8:~$
```

图 3-6　Docker 容器中的数据库命令行截图

一旦切换到 omm 这个用户，就做好了访问 openGauss 数据库的准备了。

openGauss 为用户提供了一个名为 gsql 的数据库管理程序。通过它，可以在 Linux 环境中查看现有的 openGauss 数据库列表，并且选择登录特定的数据库。要列出现有的数据库，只需要在命令行中输入这样的语句：

```
gsql -p 5432 -l
```

其中，gsql 是数据库管理程序的名字；"-p"的意思是查找在容器内部 5432 这个端口上提供服务的数据库，这个端口正是在启动容器时的命令中指定的容器内部端口（详见 3.1.4 节）；"-l"是指列出现有数据库。

这个语句的运行结果如图 3-7 所示。

```
                                List of databases
   Name    | Owner | Encoding | Collate | Ctype | Access privileges
-----------+-------+----------+---------+-------+-------------------
omm        | omm   | UTF8     | C       | C     |
postgres   | omm   | UTF8     | C       | C     |
template0  | omm   | UTF8     | C       | C     | =c/omm           +
           |       |          |         |       | omm=CTc/omm
template1  | omm   | UTF8     | C       | C     | =c/omm           +
           |       |          |         |       | omm=CTc/omm
(4 rows)
```

图 3-7　openGauss 数据库列表截图

可见，这个镜像的提供商已经内置了 4 个数据库 omm、postgres、template0 和 template1。其中，postgres 这个数据库是一个默认数据库，无须用户名和密码就可以登录，适合我们进行学习和演练。所以，后面我们将登录这个数据库，并在这个数据库中进行关系数据库的学习。至于这个数据库的名字 postgres，恰好是著名的开源数据库 PostgreSQL 的名字，华为的 openGauss 就是脱胎于 PostgreSQL，这里也显示出华为向前辈致敬的意思。

一旦查到了现有数据库，而且明确要登录 postgres 这个数据库，就可以继续用 gsql 这个数据库管理程序来实现数据库登录。方法是：

```
gsql -d 数据库名 -p 端口号 -U 用户名 -W 密码 -r
```

这里给出了登录数据库的语法，其中 gsql 是数据库管理程序的名字，后面有多个参数，分别指定了数据库名、端口号、用户名和密码。由于 postgres 这个默认数据库不需要用户名和密码就可以登录，因此可以用下面的简化方式实现登录：

```
gsql -d postgres -p 5432
```

一旦登录成功，会发现命令行提示符变为类似下面的样子：

```
openGauss=#
```

这说明已经登录 openGauss 数据库了，后面就可以通过 SQL 和一些专用指令对数据库进行管理操作了。

3.2.3 对数据库进行基本的用户和权限编辑

在登录数据库后，在提示符“openGauss=#”后面输入的所有 SQL 指令都需要以英文的分号“;”结尾，否则指令不会被运行。

当用前面的方法登录数据库后，默认的登录角色是 omm。这个角色的权限是很大的。到底它有什么权限，除 omm 外，这个数据库还有哪些可登录的用户？这可以通过一条简单的专用指令来查询，方法是在命令提示符后面输入：

```
\du
```

这个专用查询指令会返回数据库的用户清单和每个用户的权限，如图 3-8 所示。

```
openGauss=# \du
                                   List of roles
 Role name |                            Attributes
-----------+------------------------------------------------------------
 gaussdb   | Sysadmin                                                   | {}
 omm       | Sysadmin, Create role, Create DB, Replication, Administer audit, Monitoradmin, Ope
```

图 3-8 openGauss 数据库用户列表截图

从图 3-8 中可见，omm 这个角色的权限很大，比如具有系统管理员(sysadmin)权限，可以创建新的角色(create role)、创建数据库(create DB)等。正是由于它的权限太大，因此只适合用来对数据库进行管理，而非进行日常的数据操作。如果想用 omm 这个用户对 openGauss 数据库进行远程访问，比如通过容器外部的 Python 程序或者可视化管理工具来访问，那么是比较困难的，因为系统对远程使用这个 omm 用户进行了限制。为了方便读者后续的学习，建议创建一个新用户。

1. 创建新用户

可以用以下语句创建新用户：

```
create user 用户名 password "密码";
```

其中，create user 的意思是创建用户。用户名一般是字母开头的字符串，password 后面跟着的是带英文双引号的密码。注意：密码必须是 8 位的(包括大小写字母和数字)，而且这个英文双引号是不能缺少的，语句后面的英文分号也是不可缺少的。

可以仿照这个语法创建一个名为 stud 的新用户，其密码为“Study@2023”。

```
create user stud password "Study@2023";
```

2. 赋予这个用户充分的权力

今后将使用这个新创建的用户从容器外部访问 openGauss 数据库，并进行诸如创建表、删除表、新增和查询数据等操作，因此需要的权限还是比较多的。需要给这个新用户添加访问默认表空间 pg_default 的权限，由于所有的数据表都要依附在某个表空间中，因此，如果没有这个权限，就无法建表。进行权限分配的方法如下：

```
grant create on tablespace pg_default to stud;
```

该语句是把在表空间 pg_default 的创建操作的权限赋给用户 stud。一旦拥有这个权限，用户 stud 就可以在数据库中创建表、删除表，建立视图，建立存储过程了。

3.3　基于图形化的关系数据库管理工具的使用


视频讲解

openGauss 数据库提供了官方支持的图形化的数据库管理工具 Data Studio，它可以让用户在 Windows 操作系统中，以直观的方式来访问运行在远程主机或容器中的 openGauss 数据库，可以通过 Data Studio 运行各种 SQL 语句来实现对数据库的操作。下面以 Windows 11 操作系统为例来介绍 Data Studio 的安装和使用方法。

3.3.1　配置运行环境

图形化的数据库管理工具 Data Studio 是基于 Java 语言的，所以需要先安装 Java 语言的开发环境 JDK(Java Development Kit)。首先在 Java 的官方网站下载新的 JDK 安装包。在笔者写作的时候，Java 语言是归属 Oracle 公司所有的，其开发环境 JDK 的新版本是 20，其下载链接详见前言二维码。

下载 x64 MSI Installer，即 64 位版的安装程序，如图 3-9 所示。

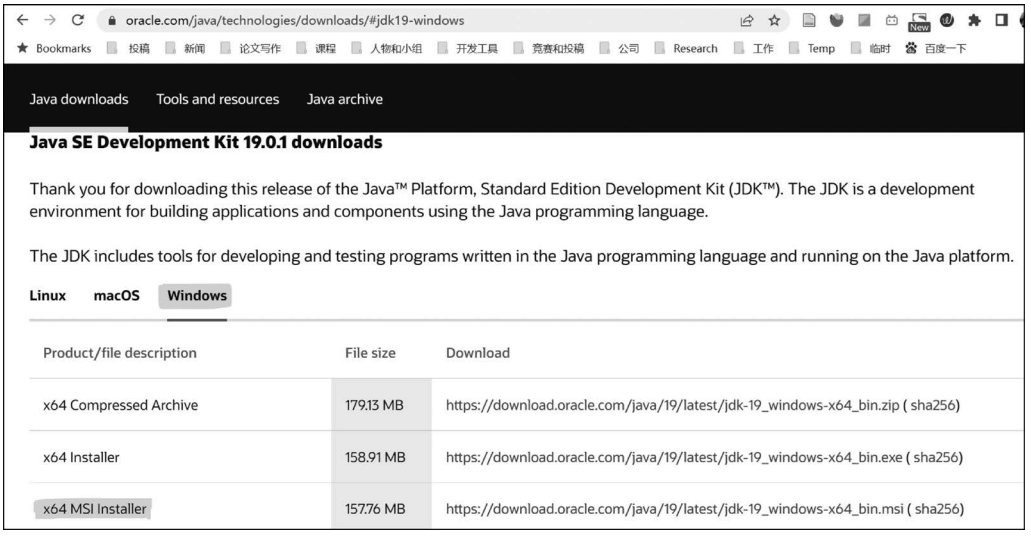


图 3-9　JDK 下载官方网站截图

下载后运行这个安装程序，建议按照默认的选项进行安装。

完成 JDK 的安装后，就可以下载 Data Studio 了，请到 openGauss 的官方网站下载(详见

前言二维码）。

在笔者写作本书的时候，其新版本是 3.1.0，如图 3-10 所示。

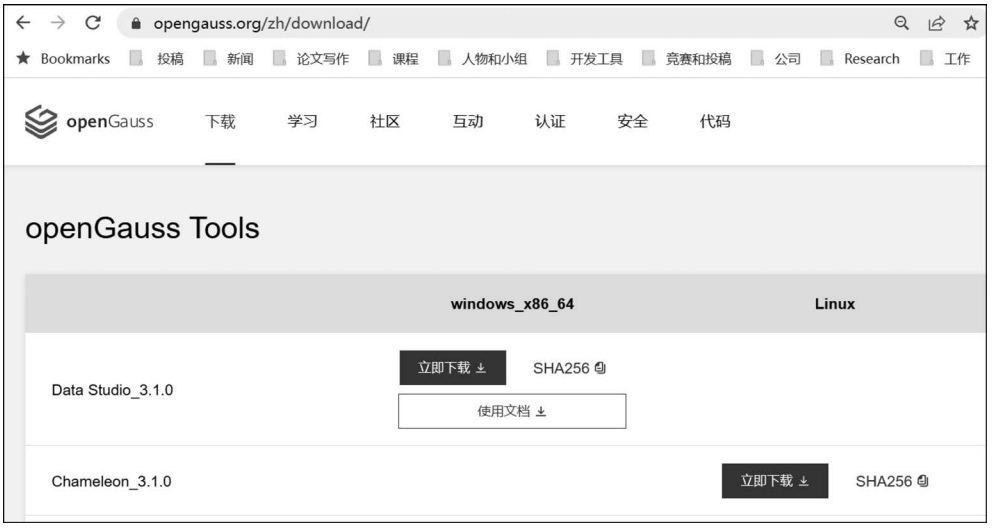


图 3-10　Data Studio 下载官方网站截图

单击其中的下载链接，就可以得到一个 ZIP 压缩包，名为 DataStudio_win_64.zip。将这个压缩包解压缩，在其中可以看到可执行文件 Data Studio.exe，如图 3-11 所示。双击它就可以启动 Data Studio。

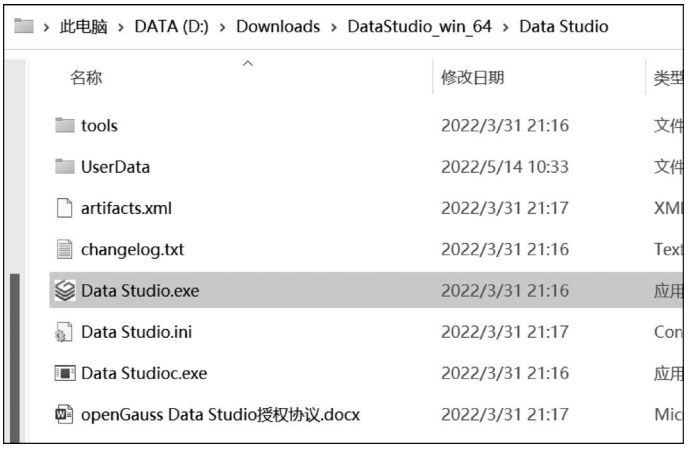


图 3-11　Data Studio 安装压缩包中的文件内容

3.3.2　通过 Data Studio 连接 openGauss 数据库

双击 Data Studio.exe 如果一切正常，会看到数据库连接界面，询问要连接哪个主机中的哪个数据库，以及用户名和密码，如图 3-12 所示。

图 3-12 左侧是曾用过的数据库连接信息列表，如果是首次运行，那么这个列表应该是空的。图 3-12 右侧是新连接的必填信息。假设已经在本机上按照 3.1 节和 3.2 节的叙述安装了基于 Docker 的 openGauss 数据库，并且设置了用户 stud。那么就可以把相关信息填写到右侧的文本框中。具体填写方法如图 3-13 所示。

其中，“名称”是给这个连接起的名字，因为这是连接本地的一个数据库，所以笔者起名为

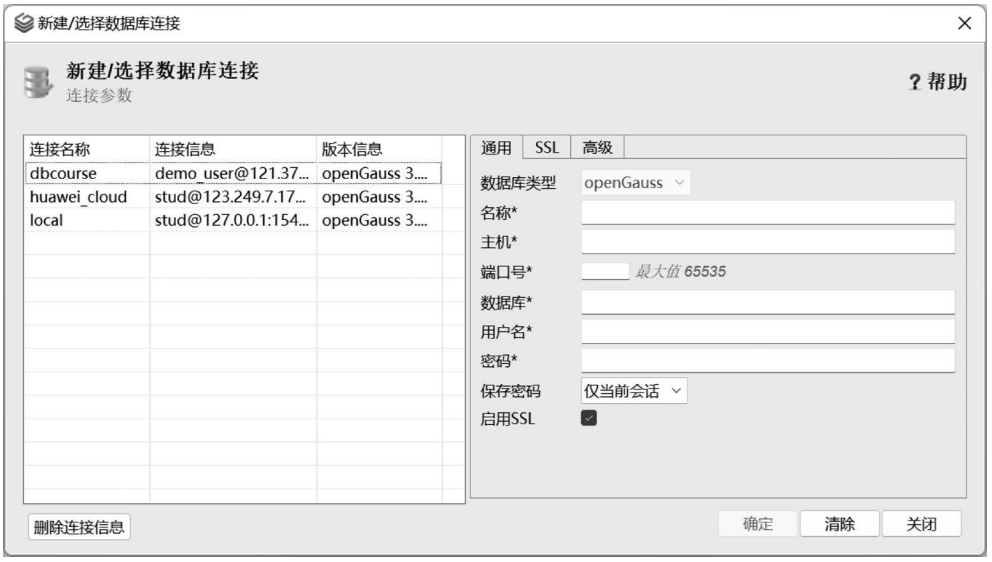


图 3-12　Data Studio 的“新建/选择数据库连接”界面

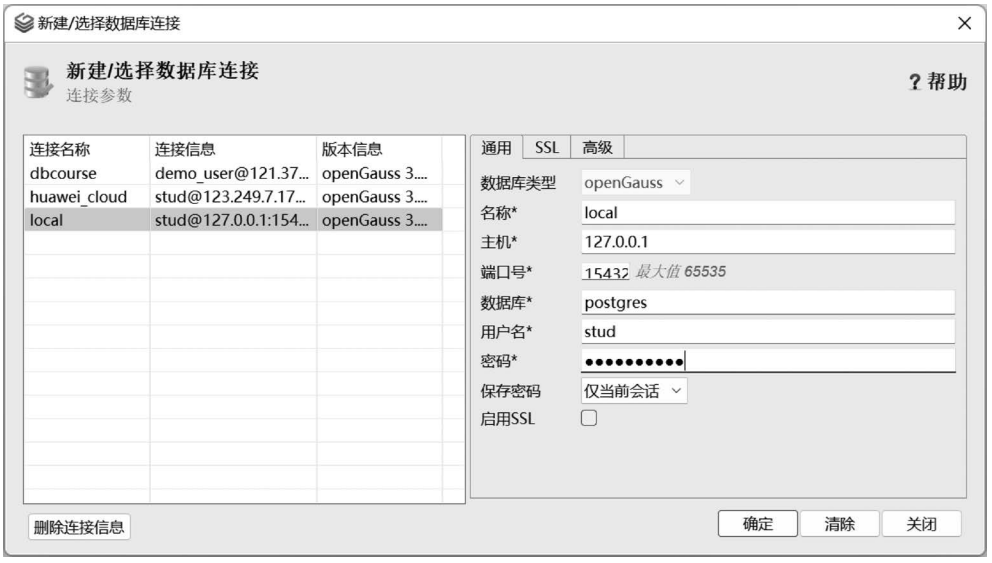


图 3-13　在 Data Studio 中输入 openGauss 连接信息

local。“主机”是数据库所在主机的 IP 地址，因为使用的是连接本机，所以使用本机的默认 IP 地址 127.0.0.1。“端口号”是指数据库对外的服务端口，在 3.1.4 节中设置其对外端口是 15432，所以就填这个数值。“数据库”是指要连接的具体数据库的名字，因为 openGauss 系统中可以有多个数据库，根据 3.2.2 节中的叙述，知道要连接的数据库名字是 postgres，所以这里就填它。“用户名”就是在 3.2.3 节中新建的用户 stud。“密码”就是在 3.2.3 节中创建新用户时设置的密码，如果当时是按照书中操作的话，应该是 Study@2023。将“保存密码”设置为“仅当前会话”，作用是在 Data Studio 没关闭之前，一直记住这个登录密码。当长时间没有操作，导致与数据库的连接断开时，一旦用户恢复操作，Data Studio 能够记住这个密码实现自动重新连接。“启用 SSL”这个复选框被取消勾选了，原因是此处仅以学习为目的，不需要 SSL 这样的通信保密措施，这些措施会增加配置数据库的难度，将来要部署实际的数据库系统时，可以启用该选项，但这超出了本书的范畴。

单击“确定”按钮后，可能会看到系统弹出的警告框，提示没有启用 SSL，有安全隐患。单击“继续”按钮忽略这个警告。

一切顺利的话，应该可以看到 Data Studio 的主界面，如图 3-14 所示。

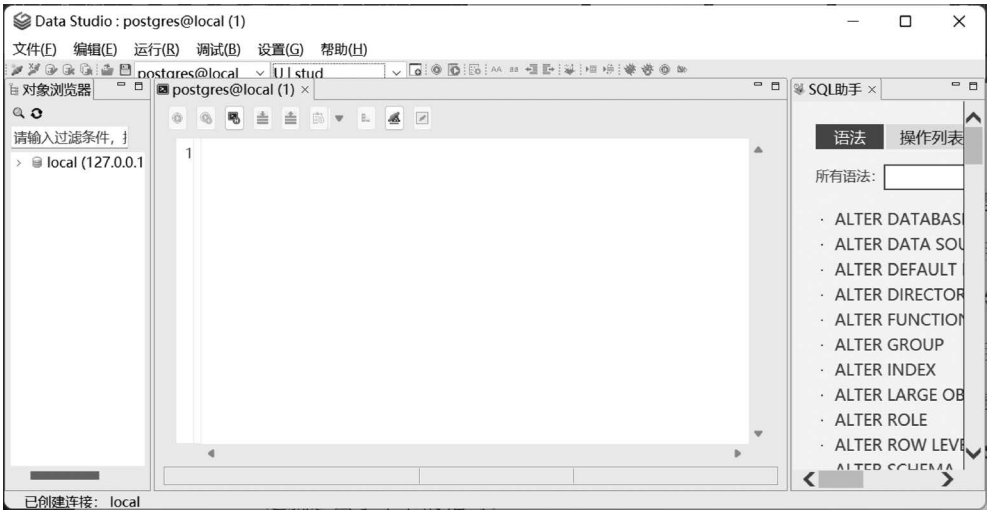


图 3-14　连接数据库成功后的 Data Studio 的主界面

其中，界面上方是菜单和工具栏区；左侧是“对象浏览器”面板；中间是输入 SQL 指令的面板；右侧是“SQL 助手”，会呈现一些有关 SQL 语句的帮助信息。

目前，“对象浏览器”中只有一个项目 local(127.0.0.1)，表示名字是 local 的数据库连接，连接的主机 IP 是 127.0.0.1，这与在前文中的连接设置是一致的。双击它，或者单击它左侧的 › 符号，即可展开这个数据库连接的细节，将会看到一个树状结构，如图 3-15 所示。

双击“数据库(2)”或者单击其左侧的 › 符号，以便展开数据库的细节信息，如图 3-16 所示。

可以看到有两个数据库，分别是 omm 和 postgres。前者有个红色的 ✕ 图标，意思是无法访问这个数据库，而在 postgres 前面有个绿色的对号图标，意思是可以访问。这是对的，因为这个 stud 用户是在 postgres 数据库中创建的(见 3.2.3 节)，当然只能访问这个数据库。

双击 postgres 选项，或者单击其左侧的 › 符号，以便展开其细节信息，如图 3-17 所示。

图 3-15　“对象浏览器”列表

图 3-16　展开后的数据库列表

图 3-17　查看数据库的信息

在 postgres 数据库下面有两种模式，分别是系统模式和用户模式。所谓模式，可以理解为目录，一个数据库可以有很多目录。因为每个数据库可以有多个用户，每个用户最好在属于自己的目录中工作，创建属于自己的表，存储自己的数据，这样多个用户可以使用同一个数据

库，而不必担心互相影响。

现在以 stud 用户的身份登录数据库，所以此时的模式是“用户模式”。双击“用户模式”选项，或者单击其左侧的 › 符号，以便展开其细节信息，如图 3-18 所示。

可以看到，在用户模式下有 public 和 stud 两个具体的用户模式，此处选择 stud 模式。stud 字样旁边的(0)是指当前这个模式下还没有任何表，创建新表后，这个括号中的数字就会更新了。

双击 stud 选项，或者单击其左侧的 › 符号，以便展开其细节信息，如图 3-19 所示。

图 3-18　数据库的用户模式

图 3-19　数据库详情列表

在 stud 这个用户模式下有很多子项目，每个子项目的左边都有 › 符号，说明它们都是目录，都可能还有子项目，可以继续展开。

现在已经完成了通过 Data Studio 连接 openGauss 数据库的任务，下面就可以通过 SQL 指令来操作数据库了。

3.3.3　通过 Data Studio 和 SQL 语句操作 openGauss 数据库

在 Data Studio 中输入和运行 SQL 语句是很直观和简单的。首先，在位于主窗体中央的编辑窗口中输入一条 SQL 语句，如图 3-20 所示。

刚刚输入的 SQL 指令是以 CREATE TABLE 开始的，你还记得是什么作用吗？顾名思义，这是创建表的语句。如果记不清了，可以查看本书的 2.4.5 节。具体而言，这条语句的作用是：创建一个名为 Companies 的数据表。该表包含的列(字段)有 id、name、address、email 和 phone。int、varchar(50)和 text 是数据类型，它们表示可以在该字段中存储哪些数据。

此时，观察到左侧“对象浏览器”窗口中 stud 下方有“普通表(0)”，意思是此时还没有表。如果成功运行了这条语句，括号中的数字应该会变成 1。此外，在右侧的“SQL 助手”面板中可以看到已经自动出现了 CREATE TABLE 字样，以及对这条 SQL 语句的帮助信息，这体现了 Data Studio 试图给开发者提供主动的帮助。

写完 SQL 语句后，怎么让它运行呢？别急，请仔细看代码编辑窗口上方是不是有一串小按钮，单击最左边的按钮 ◉ 就可以执行当前 SQL 指令，并在当前编辑窗口显示运行结果；单击第二个按钮 ◉ 可以弹出一个新窗口来执行当前 SQL 指令，并在新的编辑窗口中显示运行

图 3-20　Data Studio 中的 SQL 语句编辑窗体

结果；单击第三个按钮 可以弹出一个窗体显示已经运行过的 SQL 命令的历史记录。对于开发者而言，第一个按钮是最常用的。下面单击它，看看会不会新建一个表。如果顺利，单击第一个按钮 ，这条 SQL 语句就会被运行，给数据库创建一个新的表，如图 3-21 所示。

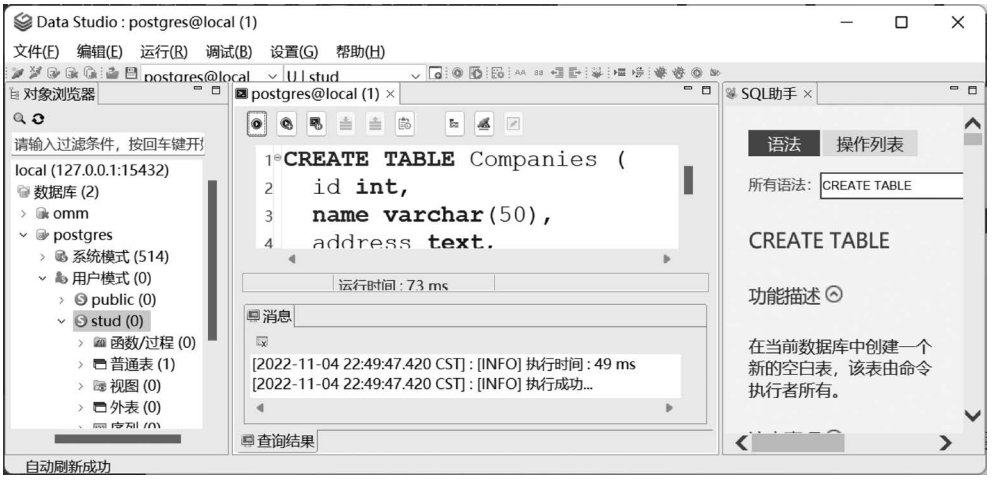


图 3-21　Data Studio 中运行 SQL 语句后的消息窗体

从图 3-21 中可以看出，在 SQL 编辑框下方出现了一个“消息”面板，里面显示了“执行成功”字样；在左侧的“对象浏览器”面板的 stud 下方，“普通表”右侧的括号里，原本的数字 0 已经变成了 1，这说明新表已经被成功创建。

此时，美中不足的是“对象浏览器”面板中用户模式 stud 右边的括号中的数字还是 0。按照前面的说法，此时 stud 模式下已经有了数据表，就不应该是 0 了，这是什么原因呢？

不要急，因为 Data Studio 还没来得及更新所有的界面信息，如果你想让它花点时间彻底更新界面信息，很简单，只需要单击“对象浏览器”面板上方的刷新图标 ，就是图中放大镜右侧的循环状的箭头。单击刷新图标后，就应该看到界面刷新的动作，并且看到更新后的界面，如图 3-22 所示。

从图 3-22 中可以看出，“对象浏览器”面板中的“用户模式”和 stud 字样后面的括号中，数字都从 0 变为 1 了。随着今后对数据库的扩充操作，这个数字还会不断增长。

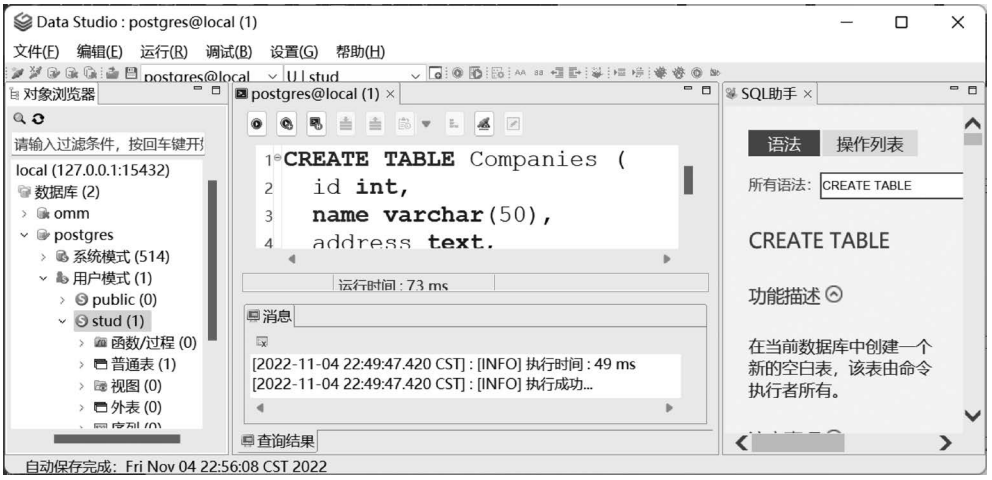


图 3-22　Data Studio 中刷新后的界面

双击“对象浏览器”面板中用户模式 stud 下方的“普通表”，或者单击它左侧的 › 符号，会看到其中具体包含哪些表格，如图 3-23 所示。

可见目前只有一个普通表，名字是 companies。细心的同学可能会发现在 SQL 语句中写的是首字母大写的 Companies，但在普通表列表中却显示的是首字母小写的 companies。这是因为 SQL 对于语法关键字和表名是不区分大小写的，因此两者对于数据库而言是一样的。

在“对象浏览器”面板中继续展开普通表下的 companies 表，展开的方式是双击 companies，或者单击其左侧的 › 符号。表的内部结构如图 3-24 所示。

可见，表中包括 3 类主要的结构信息，分别是列、约束和索引。刚才的 SQL 语句中只是指定了新表的列，并没有指定任何约束和索引信息，因此只能继续展开“列”这个项目，会看到在列中包含的具体信息，如图 3-25 所示。

图 3-23　表格列表

图 3-24　表格详情

图 3-25　“列”清单

图 3-25 中显示了 address、email、id、name 和 phone 这 5 个列(字段)，这与前面的 SQL 语句的内容是一致的，只是它是按照列名的首字符排序显示的，因此先后顺序与在 SQL 语句中的不太相同，但这对于表结构和存放数据而言是没有关系的。

现在，尝试再运行一条 SQL 指令，给这个表添加一个主键约束。首先，选中并删除 SQL 编辑框中现有的指令，然后输入如图 3-26 所示的新指令。

单击“运行”按钮，应该会看到运行成功字样。这样我们就给这个新表添加了一个主键约束，指定 id 列为主键。此时，我们在“对象浏览器”面板中可以看到 companies 这个表的“约束”项中新增了一个约束 companiespk，这与刚才的 SQL 指令内容是一致的，如图 3-27 所示：

```
postgres@local (1) ×
1 ALTER TABLE companies
2 ADD CONSTRAINT companiesPK PRIMARY KEY(id)
```

图 3-26　输入的改变表结构的 SQL 语句

图 3-27　“列”详情

表是有了，但还缺少数据，下面来添加几条记录。请回忆在 2.4.8 节中学习的相关 SQL 指令，结合这个新表的结构，输入如图 3-28 所示的指令。

```
postgres@local (1) ×
1 INSERT INTO companies(address,email,id,name,phone)
2 VALUES
3 ('麓山南路332号','xxx@feichaung.com',1,'飞创公司','1887766332'),
4 ('建设路32号','xxx@meiming.com',1,'美名公司','1387766332'),
5 ('光明路33号','xxx@leishen.com',1,'雷神公司','19877663 32'),
6 ('矿工路132号','xxx@tianfeng.com',1,'天风公司','1317766332')
```

图 3-28　使用 SQL 语句添加数据①

单击“运行”按钮后，会弹出错误信息，提示主键不允许重复。下面来检查这段代码，会发现新增的 4 条记录中，每个 id 字段的值都是 1，而 id 字段已经被我们设置为主键了，是必须非空且唯一的。解决方法很简单，就是修改后面 3 条记录的 id 字段值，如图 3-29 所示。

```
postgres@local (1) ×
1 INSERT INTO companies(address,email,id,name,phone)
2 VALUES
3 ('麓山南路332号','xxx@feichaung.com',1,'飞创公司','1887766332'),
4 ('建设路32号','xxx@meiming.com',2,'美名公司','1387766332'),
5 ('光明路33号','xxx@leishen.com',3,'雷神公司','1987766332'),
6 ('矿工路132号','xxx@tianfeng.com',4,'天风公司','1317766332')
```

图 3-29　使用 SQL 语句添加数据②

再次运行，应该就可以看到运行成功字样。

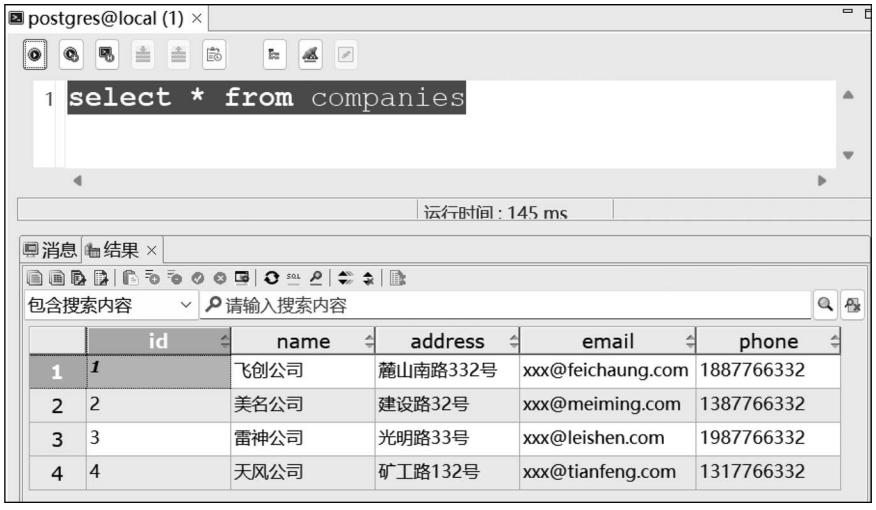
现在我们的数据库中终于保存了实际数据，让我们查询一下吧。删除前面的代码，并输入新的查询代码，按照 2.4.1 节讲解的简单查询语句，可以这样写：

```
select * from companies
```

然后单击“运行”按钮，应该可以看到我们刚才录入的 4 行信息，如图 3-30 所示。

我们已经通过 Data Studio 书写并运行了 SQL 语句，创建了表格，输入了数据，并实现了简单查询。下面使用类似的方法来演练 2.4 节中学习的各类 SQL 语句。

Data Studio 的功能很强大，除运行 SQL 指令外，它还允许我们以图形可视化的方式创建表、修改表、录入和编辑数据。不过这些不是本书的教学重点，就留给读者慢慢探索和尝试吧。



	id	name	address	email	phone
1	1	飞创公司	麓山南路332号	xxx@feichaung.com	1887766332
2	2	美名公司	建设路32号	xxx@meiming.com	1387766332
3	3	雷神公司	光明路33号	xxx@leishen.com	1987766332
4	4	天风公司	矿工路132号	xxx@tianfeng.com	1317766332

图 3-30　进行 SQL 查询的结果

3.4　基于 Python 的关系数据库的连接和查询

openGauss 数据库脱胎自开源关系数据库 PostgreSQL，在其基础上做了深度的改进和优化。因此，openGauss 数据库可以借鉴和使用开源 PostgreSQL 数据库的成熟生态环境，包括针对各种编程语言的驱动接口。

在笔者写作的 2022 年，openGauss 官方支持 C/C++、Java 和 Python 这 3 种语言的编程接口。但官方支持的接口和驱动软件都是针对 Linux 的，而我们的大部分读者可能都在使用 Windows 操作系统。因此，我们可以借助开源数据库 PostgreSQL 针对 Python 的编程接口 psycopg2，它发展得更早，而且提供了针对 Windows 的版本。

虽然 psycopg2 是针对开源数据库 PostgreSQL 的，但由于 openGauss 与 PostgreSQL 共享了很多底层的技术，因此 openGauss 的常规操作都可以基于 psycopg2 来调用实现。下面介绍 psycopg2 的安装和使用方法。

3.4.1　psycopg2 的安装


视频讲解

psycopg2 的官方网址详见前言二维码。在笔者写作的 2022 年，共有两个主要版本可以使用，分别是 psycopg2 和 psycopg3。这两个版本是同时被官方支持的，基于兼容性和稳定性的考虑，我们使用更加成熟的 psycopg2 来连接操作 openGauss 数据库。

psycopg2 的安装方法很简单，用 Python 语言的官方安装工具 pip 即可。

首先，打开一个命令行窗口。如果使用的是 Windows 操作系统，建议以管理员身份来打开命令行窗口（见图 3-31），因为安装过程可能会往系统磁盘写入文件，如果没有管理员权限，可能会遇到错误。

在打开的命令行窗口中，输入如下的 pip 指令：

```
pip install psycopg2 - binary
```

在笔者写作的时候，psycopg2 的新版本是 2.9。psycopg2 成功安装后，应该可以看到类似 Successfully installed psycopg2-binary-2.9.5 的字样。

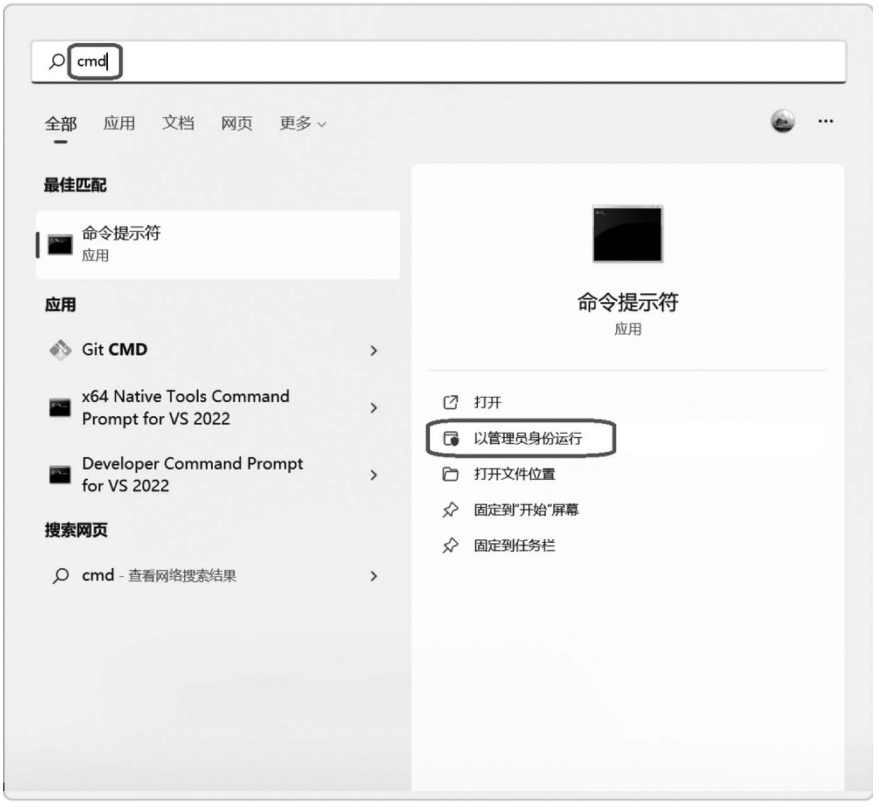


图 3-31　在 Windows 平台上启动命令行

现在，我们就完成了对 psycopg2 的安装。下面讲解其基本的使用方法。

3.4.2　简单的数据库查询

用任何编程语言操作数据库，一般都有连接数据库、发送 SQL 指令、获得指令的运行结果、显示结果、断开连接这几个关键步骤。psycopg2 也不例外。让我们看一段简单的示例代码：

```
#导入 psycopg2 库
import psycopg2
#连接数据库
conn = psycopg2.connect(database = "postgres", user = "stud", password = "Study@2023", host =
"127.0.0.1", port = "15432")
#通过 cursor 对象向数据库发送 SQL 指令
cur = conn.cursor()
cur.execute("SELECT id from companies")
#获得 SQL 指令的运行结果
rows = cur.fetchall()
#对多行返回数据进行显示
for row in rows:
    print(row)
#断开数据库连接
cur.close()
conn.close()
```

上面的代码实现了对我们在 3.3 节中创建的数据表的简单查询，通过以“#”开头的注释

信息，应该不难理解每条语句的作用。下面来一一解析。

第一步，通过 import 指令导入 psycopg2 库，只有先导入库，我们才能使用这个库中的相关功能。

第二步，创建一个关键的 psycopg2 对象，即 connect 类型的对象。这个对象被存储在 conn 这个变量中，它将负责连接数据库。我们在创建这个对象时，通过参数传入了数据库名、用户名、登录密码、主机 IP 和通信端口信息。请回忆一下，我们在 3.3.2 节使用 Data Studio 连接 openGauss 数据库时是不是在窗口中填写了这些信息？所以，无论是通过图形化界面还是通过编程的方式，其本质是相同的，都是跟数据库系统“对话”。

第三步，通过 connect 对象创建第二个关键的 psycopg2 对象，即 cursor 类型的对象。这个对象被存储在 cur 这个变量中，它将负责向数据库发送 SQL 指令并获得运行结果。创建这个对象时，不需要提供什么额外的信息，所以其创建函数带一个空括号，没有任何参数。

第四步，通过 cursor 类型对象的 execute 函数向数据库发送 SQL 指令。这条指令是一个简单的查询指令，目标很明确，就是获得表中 id 列的所有行。这条 SQL 指令是用 Python 的字符串来表示的，在 psycopg2 中，所有的 SQL 指令都将使用字符串来表达，但复杂一些的 SQL 指令需要考虑输入参数，以及 Python 和 SQL 数值类型转换的问题，我们会在后面的示例中进一步说明。

第五步，通过 cursor 类型对象的 fetch 函数获得 SQL 指令的运行结果，返回的结果是 Python 的可迭代容器类型，所以可以通过循环语句来遍历其中的每一个数据，并且进行显示。

第六步，分别调用 cursor 类型和 connect 类型对象的 close 函数，实现对数据库连接的关闭。注意，要先调用 cursor 的 close 函数，再调用 connect 的 close 函数。因为 cursor 是由 connect 对象创建的，所以它是依附于后者的。在关闭数据库连接时，我们要先释放子对象，再释放其母对象。如果不做这一步，Python 程序在关闭时，数据库大概率也会自动关闭这个打开的数据连接，但这并不保险，可能会造成计算机内存等资源的浪费，也会有一些安全隐患，所以建议读者在完成对数据库的操作后，就关闭数据库连接。

这段代码的运行结果如下：

```
(1,)
(2,)
(3,)
(4,)
```

这 4 行输出正是我们在 3.3.3 节中向 companies 表中添加的 4 行记录对应的 id 列的值。说明查询结果是正确的。

3.4.3　对数据库进行修改

前面已经学习了如何进行简单查询，与查询不同的是，对数据库的修改需要在发送 SQL 指令后额外添加一条语句，即

```
conn.commit()
```

也就是说，要在发送 SQL 指令后，通过调用 connect 对象的 commit 函数告诉数据库，让 SQL 语句所做的数据库修改生效，如果不调用这个函数，则对数据库的修改可能没有生效。对于数据库而言，数据是最关键的，不经过用户的反复确认，最好不要随便对数据库进行修改。这样通过对 commit 函数的调用，就是对数据库修改操作的再次确认。

下面我们尝试在 companies 数据库中创建一个新表：

```
#假设已经按照前一个例子创建了 conn 和 cur 对象
#构造一个多行的 SQL 语句,用来创建一个新的表
sql_str = """
CREATE TABLE Customers(
  customer_id SERIAL PRIMARY KEY,
  name VARCHAR(20) NOT NULL,
  phone_no VARCHAR(20)
);
"""
#向数据库发送这条 SQL 指令
cur.execute(sql_str)
#通知数据库,将 SQL 语句所做的修改生效
conn.commit()
#请仿照前一个例子关闭数据库连接
```

在这个例子中，我们构造了一个多行的 SQL 语句，并把它存放在一个名为 sql_str 的 Python 变量中。这个变量是字符串类型的。需要特别注意的是，我们通过 3 个连续的英文双引号来开启一个多行的字符串，并且在字符串输入后，再用 3 个连续的英文双引号来告诉 Python 这个多行字符串已经录入完毕了。这个技巧还是很常用的，因为很多数据库操作都需要用比较复杂的 SQL 指令来描述。

在调用 cursor 类型对象的 execute 函数执行这条 SQL 指令后，我们特别调用了 connect 类型对象的 commit 函数让所做的数据库修改生效。如果没有对 commit 函数进行调用，这个表是不会被真正创建的。可以用 Data Studio 连接数据库，来观测 commit 函数调用前后数据库的变化。可以发现，Customers 这个表是在调用 commit 函数之后才出现的。

3.4.4 使用参数构造 SQL 语句

什么叫用参数来构造 SQL 语句？为什么用参数来构造 SQL 语句？为了理解这两个问题，我们需要先看一个简单的例子。在这个例子中，以 3.4.3 节的方法向 Customers 表中新增一条记录。请观察这段代码：

```
#假设已经按照 3.4.2 节中的例子创建了 conn 和 cur 对象
#构造一个多行的 SQL 语句,用来创建一个新表
sql_str = """
INSERT INTO Customers(name, phone_no)
VALUES ('lei cai', '1337667777');
"""
#向数据库发送这条 SQL 指令
cur.execute(sql_str)
#通知数据库,将 SQL 语句所做的修改生效
conn.commit()
#请仿照 3.4.2 节中的例子关闭数据库连接
```

这条语句运行完毕后，打开 Data Studio 应该可以看到这条新记录已经被加入 Customers 表中了。

这个例子有什么问题吗？仔细观察，你会发现这条新记录各个字段的值是写死在代码中的。也就是说，在写 Python 代码的时候，你就得事先知道用户要输入什么数据。这可能吗？一般来说是不可能的。如果在写代码的时候就知道了要输入什么数据，那么我们没有必要编写 Python 程序了，因为它不能帮助我们录入新的数据。

怎么解决这个问题？思路就是：数据在程序运行时才被用户输入，被存放在 Python 的变量中，我们在构造 SQL 指令的时候需要嵌入这些变量。这些变量对于 SQL 语句而言就变成了参数。

在 SQL 语句中，嵌入参数（也就是待定内容）的方法是用“%”标识符。请看下面的例子：

```
#假设已经按照 3.4.2 节中的例子创建了 conn 和 cur 对象
#假设用户在程序运行中输入了两个数据,分别保存在这两个变量中
name = '张三'
phone_no = '1887667777'
#向数据库发送一条 SQL 指令,注意:这条 SQL 指令中包含 2 个参数,其值在运行时会由上面的两个变量值填充
cur.execute("""
INSERT INTO Customers(name, phone_no)
VALUES ( %s, %s);
""", (name, phone_no))
#通知数据库,将 SQL 语句所做的修改生效
conn.commit()
#请仿照 3.4.2 节中的例子关闭数据库连接
```

在这段代码中，我们模仿真实场景，假设用户在程序运行时输入了两个数据，分别保存在 name 和 phone_no 这两个变量中，然后通过 cursor 对象的 execute 函数构造并发送一条带参数的 SQL 指令。在这个指令中出现了两个“%s”，说明这里存在字符串型（s 是字符串 string 的意思）的待定参数。execute 函数的第一个参数就是这条 SQL 指令，第二个参数是一个 Python 的元组类型，即用括号括起来的多个值。这里，元组中存放了两个变量，分别是前面提到的 name 和 phone_no，里面保存了用户在运行时输入的数据。

通过这种方法，psycopg2 可以在运行时根据用户输入的数据来组装 SQL 指令，实现自定义的数据输入。

注意：

- 在%s 的两侧不要再加引号，否则会报错。
- 无论参数中保存什么类型的数据，一律用%s 来代表，因为 SQL 语句整体就是字符串。
- 如果 SQL 语句中有表示求余运算的%符号，为了避免与%s 混淆，把求余运算的%改写为%%。psycopg2 在与数据库通信时会将%%正确转换为表示求余计算的%。
- 只能用%s 参数来代表 SQL 语句中的字段值，不能用%s 来代表表名或字段名。若有这样的需求，则参考 psycopg2 帮助中的高阶用法。

3.4.5　处理 Python 与 SQL 的数据类型转换

Python 和 SQL 是两种不同的语言，它们内置的数据类型也是不同的。比如我们在 2.4.6 节中提到基于 openGauss 的 SQL 支持整型、浮点型、字符串、文本和日期类型等若干数据类型。这些数值类型的名字与 Python 不完全相同。比如在 SQL 中，我们用 VARCHAR 类型来表示可变长度的字符串，但在 Python 中是没有这个类型的。那么，我们在 Python 程序中向数据库发送 SQL 指令时，如何确保相关参数能转换成 SQL 能理解的数据格式呢？

psycopg2 的参数机制中已经内置了 Python 语言中的类型向 SQL 类型自动转换的功能。为了体验这种功能，我们先通过下面的代码给已经存在的 Customers 表增加几个不同类型的字段，然后写一段代码来实现为这些不同类型的字段添加数据，在这个过程中体验两种不同语言之间的数据类型转换。

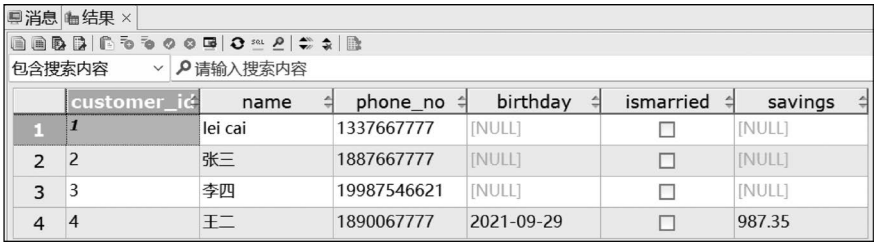
首先，仿照 2.4.6 节中的示例，给现有表添加几个不同类型的字段。

```
#假设已经按照 3.4.2 节中的例子创建了 conn 和 cur 对象
#构造一个多行的 SQL 语句，用来给现存表新增几个不同类型的列
sql_str = """
ALTER TABLE Customers
ADD
(
birthday DATE,
isMarried BOOLEAN,
savings FLOAT
);
"""
#向数据库发送这条 SQL 指令
cur.execute(sql_str)
#通知数据库，将 SQL 语句所做的修改生效
conn.commit()
#请仿照 3.4.2 节中的例子关闭数据库连接
```

现在 Customers 表中就有多种类型的字段了，包括整型的 id 列、字符串型的 name 列、日期型的 birthday 列、布尔型的 isMarried 列和浮点型的 savings 列。下面我们尝试新增一条记录，给这些不同类型的列同时进行赋值，看看能否成功。

```
#假设已经按照 3.4.2 节中的例子创建了 conn 和 cur 对象
#假设用户在程序运行过程中输入了以下不同类型的数据，分别保存在变量中
from datetime import date
name = '王二'
phone_no = '1890067777'          #Python 的字符串型
birthday = date(2021,9,29)       #Python 的日期类型
is_married = False               #Python 的布尔型
savings = 987.35                 #Python 的浮点型
#向数据库发送一条 SQL 指令，注意：这条 SQL 指令中包含 2 个参数，其值在运行时会由上面的两个变量值填充
cur.execute("""
INSERT INTO Customers(name, phone_no, birthday,
                      isMarried, savings)
VALUES ( %s, %s, %s, %s, %s);
""", (name, phone_no, birthday, is_married, savings))
#通知数据库，将 SQL 语句所做的修改生效
conn.commit()
#请仿照 3.4.2 节中的例子关闭数据库连接
```

运行代码，如果此时用 Data Studio 打开数据库，并查看 Customers 表中的数据，应该会看到如图 3-32 所示的结果。


消息 结果 ×

包含搜索内容 请输入搜索内容

	customer_id	name	phone_no	birthday	ismarried	savings
1	1	lei cai	1337667777	[NULL]	☐	[NULL]
2	2	张三	1887667777	[NULL]	☐	[NULL]
3	3	李四	19987546621	[NULL]	☐	[NULL]
4	4	王二	1890067777	2021-09-29	☐	987.35

图 3-32　在 Data Studio 中查看数据插入后的结果

从这个结果可见，psycopg2 已经正确地把 Python 变量中保存的日期型、布尔型、数值型的信息转换成了 SQL 能够理解的形式并进行了保存。如果有更复杂的数据类型要转换，比如二进制型等，可以参考 psycopg2 的官方帮助(网址详见前言二维码)。

3.4.6　简单图形化界面的实现方法


视频讲解

对于一个数据库应用程序而言，图形化界面往往是不可缺少的。比如我们需要提供一个界面，让用户输入数据进行查询，并给出反馈结果。如果一切都用命令行来实现，那么用户操作起来很不直观，也不方便。

用 Python 实现图形界面的方式有很多，大体上可以分为两类：一类是窗口式的应用程序，另一类是基于浏览器的 Web 程序。前者以 PyQt、PySimpleGUI 等为代表，后者以 Streamlit、Gradio 为代表。考虑到基于浏览器的 Web 程序可以被部署在云端，让用户无须安装程序就能访问使用，对用户更加便捷友好，所以本书将以 Gradio 这种新型的界面工具来讲解如何实现图形化的数据库应用。限于篇幅，本书只讲解很基础的 Gradio 使用方法，更详细的帮助信息请参考其官方帮助(网址详见前言二维码)。

1. 安装 Gradio

用 Python 自带的 PIP 安装工具，在命令行中输入如下命令即可安装 Gradio。建议以管理员身份打开命令行窗口(详见 3.4.1 节中的叙述)。

```
pip install gradio
```

2. 创建一个数据库登录界面

首先打开一个能够编辑 Python 程序的编辑器，比如 VS Code、PyCharm 或者一个简单的记事本程序；然后输入下面的代码，并将其保存为.py 文件，比如 gradio_hello.py。

```
import gradio as gr
import psycopg2
#创建一个全局变量,用于保存 psycopg2 的 connect 对象
conn = None
#定义登录函数,成功登录数据库则返回 True,否则返回 False
def login_fn(name, password):
    global conn
    try:
        conn = psycopg2.connect(database = "postgres",
                    user = name, password = password,
                    host = "127.0.0.1", port = "15432")
    except psycopg2.Error as e:
        conn = None
        return False
    else:
        return True
#定义界面
with gr.Blocks() as opengauss_demo:
    gr.Markdown('# 一个简单的数据库 Web 程序')
#运行界面
opengauss_demo.launch(auth = login_fn)
```

可以在 VS Code 中直接单击“运行”按钮 ▷，运行这段代码。也可以在代码所在文件夹，

用命令行执行下面格式的语句：

```
python 文件名.py
```

请把“文件名”替换成你保存文件时起的实际文件名。

运行这段代码后，会看到如图 3-33 所示的命令行提示。

```
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

图 3-33　Gradio 图形界面程序启动后的命令行提示

图 3-33 中显示，这个程序已经在运行了。它是一个 Web 程序，需要打开浏览器，输入图中的 IP 地址来访问。

打开一个浏览器，输入图 3-33 中指定的 IP 地址，然后按回车键，应该就会看到一个登录界面，如图 3-34 所示。在此输入登录 openGauss 数据库的用户名和密码。按照在 3.2.3 节中的设置，正确的用户名是 stud，正确的密码是 Study@2023。

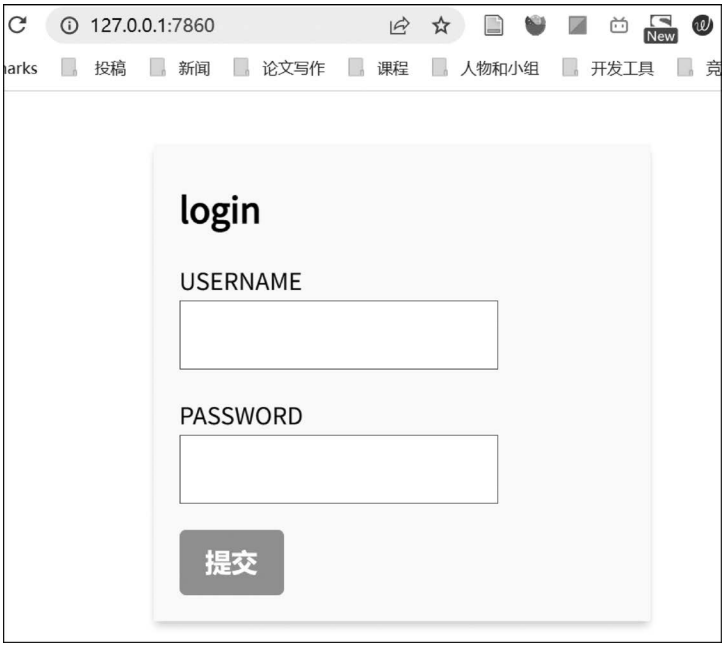


图 3-34　Gradio 登录界面

如果输入的是正确的信息，单击“提交”按钮后会看到如图 3-35 所示的界面，虽然只有一个标题，但说明已经成功登录数据库了。

若登录信息填写错误，则会看到如图 3-36 所示的出错页面。你可以单击浏览器的“返回”按钮 ←（或按键盘上的回退键，即 Backspace 键）回到登录界面，重新填写信息。

这就是第一个图形界面的数据库应用程序，虽然非常简单，但包含 Gradio 的使用要点。让我们逐一分析代码内容。

第一步，为了使用 Gradio 和 psycopg2 库的功能，通过 import 语句导入了这两个库。我们还通过 import 语句的 as 子语句给 Gradio 这个库起了个更短的别名 gr，后面就可以通过这个更短的别名来访问 Gradio 的功能了。

第二步，创建了一个全局变量 conn，用于保存 psycopg2 的 connect 对象。我们把它的值

图 3-35　第一个 Gradio 程序运行成功界面

图 3-36　第一个 Gradio 程序运行出错界面

初始化为 None(就是空的意思),将在后面的代码中调用 psycopg2 的 connect 方法来创建 connect 对象,并赋值给这个全局变量 conn。如果成功,那么在本程序所有后续的代码中都可以通过 conn 来获取对数据库的操作能力。

第三步,定义一个登录函数,它有两个输入参数,分别代表用户名和密码。如果能成功登录数据库,这个函数就返回 True,否则返回 False。在这个函数的内部,通过 psycopg2 的功能来尝试登录,相关语句请参考 3.4.2 节。在这里,首次使用了 Python 的错误处理机制,即 try 语句。把可能会出错的语句放在 try 后面,通过后面的"except psycopg2.Error as e:"能够捕获到 psycopg2 在调用 connect 函数时可能出现的错误,一般就是用户名和密码错误。当我们捕获到错误后,就把全局变量 conn 置为 None,并让登录函数马上返回 False。这里之所以要把 conn 置为 None,是为了将来添加查询等数据操作代码时,可以通过检查 conn 的值是否为 None 来间接判断是否已经正确连接了数据库,只有当确认已经正确连接数据库时才尝试发送 SQL 指令。注意:在这个函数的开头,我们用 global conn 声明在这个函数内部出现的变量 conn 就是全局变量 conn,这个声明是不能少的,否则 Python 会把这个函数中出现的 conn 变量当作是局部变量而非全局变量,这样一来,函数外面的代码就无法再使用这里创建的数据库连接了。如果没有捕获到异常,就让登录函数返回 True,代表登录成功。

第四步,创建一个 Gradio 界面。这里使用了 Python 中的 with 语句,通过调用 Gradio 的 Block 函数创建一个自定义的界面,并将这个界面对象命名为 opengauss_demo。因为 with 语句的使用,只有当 Block 函数运行成功时,才会执行下方的"gr.Markdown('# 一个简单的数据库 Web 程序')"。这条语句会按照 Markdown 标签语言的语法输出格式化文本。这里用了一个 Markdown 语句的标签"#",意思是一级标题格式,因此后面的文本会以比较粗大的字体显示。通过 Markdown 标签语言可以输出各种格式化文本,比如一级标题、二级标题、列表

项、水平横线、数学公式等。关于 Markdown 的语法说明，可以参考各类网络教程（网址详见前言二维码）。

第五步，通过调用 Gradio 界面对象（在第四步中创建了名为 opengauss_demo 的界面对象）的 launch 函数来启动界面。在这个函数中，我们设置了唯一的参数 auth，这个参数的值就是在第三步创建的登录函数。Python 是一种函数式语言，所以函数名可以像数值那样被赋值给一个变量保存，通过该变量也可以实现对函数的调用。通过这个 auth 参数告诉 Gradio 在运行界面时，首先用我们给定的这个函数进行身份验证，当这个函数返回 True 时才让用户看到界面，否则只能看到出错提示。通过这种方式可以实现对系统安全的基本保护。

3. 创建一个数据录入页面

在上一个例子的基础上添加代码，主要是在界面中添加一个标签页，在这个标签页中摆放一些文本框，让用户为数据库中的 Customers 表添加一条记录。关于数据库中的 Customers 表的结构请参考 3.4.3 节。其代码如下：

```
import gradio as gr
import psycopg2
conn = None
def login_fn(name, password):
    #限于本书篇幅,把这个函数的内容省略,请参考上一个例子中对应的代码
def add_customer(name,phone,birthday,is_married,savings):
    global conn
    if conn is None:
        return '没找到有效的数据库连接,请重新登录'
    else:
        cur = conn.cursor()
        try:
            cur.execute("""
            INSERT INTO customers(name,phone_no,
                                  birthday,ismarried,savings)
            VALUES(
                %s,%s,%s,%s,%s
            );
            """, (name,phone,birthday,is_married,savings))

            conn.commit()
        except psycopg2.Error as e:
            return e
        else:
            return '数据添加成功!'

with gr.Blocks() as opengauss_demo:
    gr.Markdown('# 一个简单的数据库 Web 程序')
    with gr. Column ():
        with gr.Tab('添加客户信息'):
            name = gr.Text(label = '客户名')
            phone = gr.Text(label = '电话号码')
            birthday = gr.Text(label = '生日(年-月-日格式)')
            is_married = gr.Checkbox(label = '婚否')
            savings = gr.Number(label = '余额')
            add_btn = gr.Button("添加")
            msg = gr.Text(label = '结果')
```

```
            add_btn.click(fn = add_customer,
                inputs = [name,phone,birthday,is_married,savings],
                    outputs = msg)

opengauss_demo.launch(auth = login_fn)
```

运行这段代码，在成功登录后，会看到如图 3-37 所示的界面。

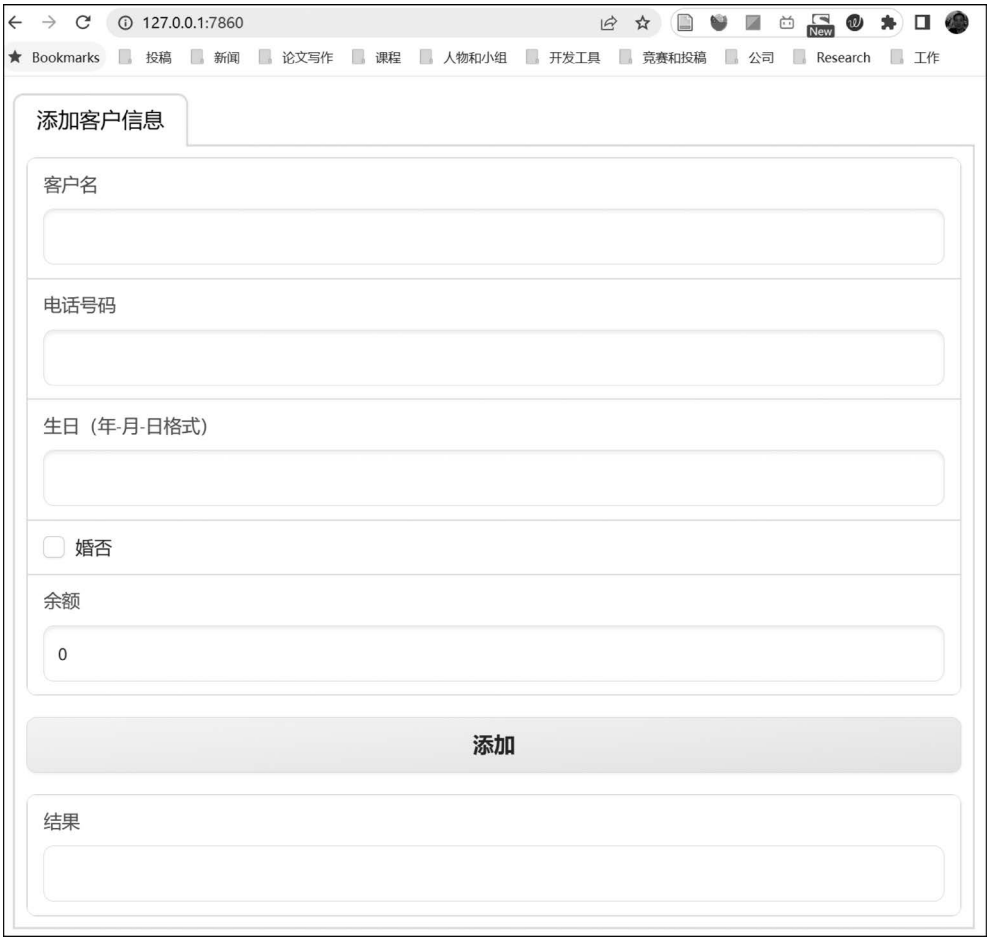


图 3-37　包含简单输入控件的 Gradio 程序

按照图 3-37 中的提示为填写相关信息后，单击“添加”按钮会触发数据库操作，如果一切顺利，能看到在界面下方的“结果”栏中显示“数据添加成功!”字样。如果数据库操作失败，也会在“结果”栏中显示对应的错误提示。

下面来分析这段代码。总的来看，与前一段代码相比，这段代码新增了两部分内容：一是新增了函数 add_customer，二是在 with gr.Blocks()下方添加了很多界面元素。下面分别讲解其内容和原理。

首先这个新增的函数负责进行数据库的操作，具体而言就是把输入参数值（对应 Customers 表的 5 个字段）信息通过 SQL 的 INSERT 指令写入数据表 Customers。其具体步骤如下：

（1）判断是否有可用的数据库连接，如果没有，就返回出错信息。

（2）根据全局变量 conn 获取 psycopg2 的 connect 对象，进而创建 cursor 对象。

（3）通过 cursor 对象向数据库发送基于 INSERT 的 SQL 指令，通过 psycopg2 的参数功能将本函数的 5 个输入参数作为一条记录的 5 个字段值封装到 SQL 指令中（参见 3.4.4 节）。并通过 connect 对象的 commit 操作使得该 SQL 语句生效（参见 3.4.3 节）。这段代码中使用的 try 语句在 Python 语言中是用于处理异常的，具体请参看上一个示例中关于登录代码的相关说明。

（4）在执行 SQL 语句的过程中，如果遇到异常，就返回出错信息；如果没有遇到异常，就返回操作成功信息。

接下来讲解在 with gr.Blocks()下新增的代码。首先，我们连用了两个 with 语句，分别创建了一个 Column 对象和一个 Tab 对象。前者是后者的父对象，后者是前者的子对象，两者形成逻辑上的层次结构。前者代表一个以列方式排列子对象的界面容器，使用它的原因是：它有一个 visible 属性可以控制是否显示这个容器的内容，这个性质将在第 4 章使用。后者代表一个标签页。多个相邻的 Tab 对象会被自动组合在一起，形成可以切换的标签页容器。在这个例子中，只有一个标签页，在第 4 章的示例中将使用多个标签页。

在创建这个 Tab 对象的时候，我们使用了一个字符串参数，这个字符串的内容是“添加客户信息”，这个字符串将显示在标签页的标题，代表这个标签页的主要功能。每个标签页都应该有一个标题以便用户分辨其包含的功能。此后的 7 行分别创建了 7 个界面控件，这 7 个控件都是这个标签页容器的子对象。所谓控件（Control），是指在图形界面中诸如文本框、列表框和按钮这样的界面对象。在这 7 个控件中，前 5 个分别对应 Customers 表中除 customer_id 外的 5 个字段值。因为 customers_id 是 openGauss 的 SERIAL 类型，新增记录时系统会自动为其创建一个自增的整数值。而 Customers 表中的其他 5 个字段是需要用户输入的，因此我们为用户提供了 5 个控件来输入这些信息。第 6 个控件是按钮控件，用户单击后就会触发前面提到的函数 add_customer，以执行新增记录的操作。第 7 个控件是个文本框控件，用于根据函数 add_customer 的返回值来显示操作结果。下面具体来看这 7 个控件的设置细节。

（1）第一个控件是文本框控件，是通过 Gradio 的 Text 函数创建的。这个文本框默认只能输入和显示一行文本。创建文本框时使用的 label 参数代表显示在文本框上方的标签，用来提示该文本框中应该输入的内容。我们把新创建的这个文本框控件赋值给变量 name。通过这个变量，后续的 Gradio 代码将能提取用户在这个文本框中输入的内容。

（2）第二个和第三个控件也都是文本框控件，其用法与（1）类似，只是分别用于存储用户输入的电话号码和生日信息。

（3）第四个控件是复选框，是通过 Gradio 的 Checkbox 函数创建的。它呈现在界面上的是可以勾选的方框，但内部存储的其实是布尔型的值。我们把这个控件赋值给变量 is_married，通过该变量就可以获取复选框内部保存的布尔型的值。True 代表用户勾选了，False 代表没有勾选。

（4）第五个控件是数字输入框，是通过 Gradio 的 Number 函数创建的。它的外形与文本框相似，只是里面只能填写数字。这个控件用于获取用户输入的余额数值。

（5）第六个控件是按钮，是通过 Gradio 的 Button 函数创建的。这个函数的参数是个字符串，代表将显示在按钮上方的文本。用户单击这个按钮后，我们希望能够触发向数据库中添加记录的功能，即调用前面提到的函数 add_customer。这个按钮对象被存储在变量 add_btn 中。把按钮和函数绑定起来需要额外的代码，请看（7）。

（6）第七个控件还是文本框，但这次是用 Gradio 的 Textbox 函数创建的。与 Text 函数不同，Textbox 函数创建的文本框可以根据内容显示多行文本。当数据库操作出错时，我们用

try…except 捕获的系统错误提示往往是多行的，所以这里用 extbox 函数来创建文本框。这个文本框对象被保存在变量 msg 中，通过这个变量我们将能够获取和设置这个文本框的内容。

最后，通过调用按钮对象（保存在变量 add_btn 中）的 click 函数实现把用户单击按钮操作与函数 add_customer 绑定。click 函数中主要有三个参数：第一个参数名是 fn，用来指示按钮被单击时应该被触发的函数，我们把函数 add_customer 赋值给这个参数；第二个参数名是 inputs，代表要发送给函数 add_customer 的输入数据，这里用一个 5 个元素的列表来赋值给这个参数，对应从 5 个界面控件中获取用户输入的 5 个字段值，这个列表的元素数是与函数 add_customer 的参数数量一致的；第三个参数名是 outputs，代表函数 add_customer 的返回结果应该在哪个控件中显示，我们把保存在 msg 变量中的控件赋值给这个参数，以便实现在最后的文本框中显示数据库操作结果的效果。

用户可以通过这个界面输入多个新记录，并用 Data Studio 查看数据录入后 Customers 表中的内容变化。本小节讲解了基于 Gradio 搭建图形化数据库 Web 应用的基本用法，对于如何显示数据表内容，如何选中某条记录并进行修改等功能，还没有展示。这些功能的实现将在 4.5 节通过一个具体的工程样例来详细讲解。

3.5　本章习题

1.（判断题）openGauss 是中国华为公司自主研发的关系数据库。（　　）

2.（判断题）使用 Docker 平台可以方便开发者进行跨平台应用程序的开发。（　　）

3.（判断题）Docker 平台可以在 Windows 平台上虚拟出一个轻量级的 Linux 容器，供人们在容器中进行 Linux 程序开发。（　　）

4.（单选题）下面哪条 gsql 指令可以实现列出 openGauss 系统中现有的数据库？（　　）

A. gsql -p 5432 -l　　B. gsql -d postgres -p 5432

C. gsql -p 5432　　D. gsql -l 5432

5.（单选题）下面哪个软件是 openGauss 官方开发的数据库访问 GUI 程序？（　　）

A. JDK　　B. Data Studio　　C. DBeaver　　D. Management Studio

6.（简答题）使用 psycopg2 进行基于 Python 的 openGauss 数据库开发的一般步骤是什么？

7.（简答题）在使用 psycopg2 进行基于 Python 的 openGauss 数据库开发时如何向查询语句中注入参数？