# 第 5 章

# 办公自动化中的数据存储

在办公自动化中，数据存储是一个关键的环节。数据存储涉及将办公自动化系统所产生的数据进行持久化存储，以便后续的数据分析、报告生成和业务流程支持等操作。

以下是办公自动化中常见的数据存储方式：

（1）数据库：使用数据库来存储和管理办公自动化系统所产生的数据是一种常见的做法。数据库提供了结构化的数据存储和查询能力，可以根据需求选择不同的数据库类型，如关系型数据库（如 MySQL、Oracle）或非关系型数据库（如 MongoDB、Redis）等。通过数据库，可以高效地存储、更新和检索数据。

（2）文件存储：在一些简单的办公自动化系统中，可以将数据以文件的形式进行存储。常见的文件格式包括文本文件（如 CSV、JSON）、Excel 文件和 XML 文件等。文件存储适用于较小规模的数据集，便于数据的导入和导出。

（3）云存储：随着云计算的发展，云存储成为了一种方便、安全且可扩展的数据存储方式。通过将数据存储在云端的存储服务（如 Amazon S3、Google Cloud Storage、Microsoft Azure Blob Storage 等），可以实现数据的远程访问、备份和共享。

（4）数据仓库：数据仓库是一种专门用于存储和管理大规模数据的系统。数据仓库通过将多个数据源的数据进行整合和转换，提供一致性、可靠性和高性能的数据存储和查询。数据仓库通常用于企业级的办公自动化系统，以支持复杂的数据分析和报告需求。

无论选择何种数据存储方式，都需要考虑数据的安全性、可靠性、可扩展性和性能等因素。此外，还需要合理设计数据模型和数据结构，以便满足办公自动化系统的数据存储和操作需求。

本章重点介绍 MySQL 和 JSON 存储数据。

## 5.1 使用 MySQL 数据库

MySQL 是一个开源的关系型数据库管理系统，被广泛用于金融数据分析和应用程序开发，本节我们介绍如何使用 MySQL 数据库。

### 5.1.1 MySQL 数据库管理系统

MySQL 是流行的开放源的数据库管理系统，是 Oracle 旗下的数据库产品。目前 Oracle 提供了多个 MySQL 版本，其中 MySQL Community Edition（社区版）是免费的，该版本比较适合中小企业数据库，本书也对这个版本进行介绍。

社区版安装文件下载如图 5-1 所示，可以选择不同的平台版本，MySQL 可在 Windows、Linux 和 UNIX 等操作系统上安装和运行，读者根据自己情况选择不同平台安装文件下载。

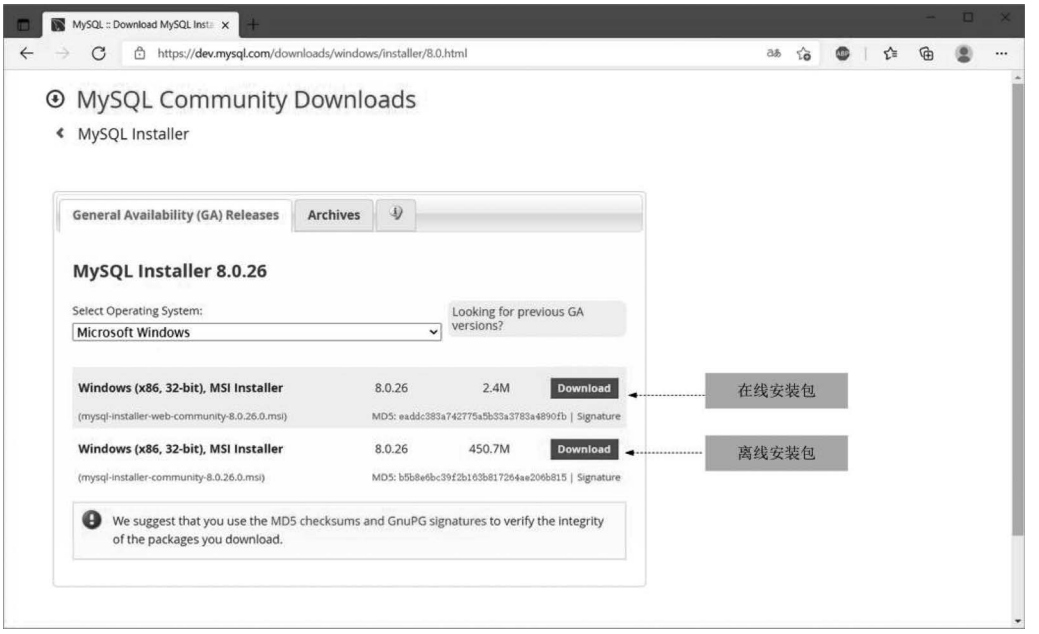



图 5-1 详细下载页面

### 5.1.2 安装 MySQL8 数据库

笔者计算机的操作系统是 Windows10 64，笔者下载的离线安装包，文件是 mysql-installer-community-8.0.28.0.msi，双击该文件就可以安装了。

MySQL8 数据库安装过程如下：

#### 1. 选择安装类型

安装过程第一个步骤是选择安装类型，对话框如图 5-2 所示，此对话框可以让开发人员选择安装类型，如果是为了学习而使用的数据库，则推荐选中 Server only，即只安装 MySQL 服务器，不安装其他的组件。

在图 5-2 所示的对话框中，单击 Next 按钮进入如图 5-3 所示对话框。

然后单击 Execute 按钮，开始执行安装。

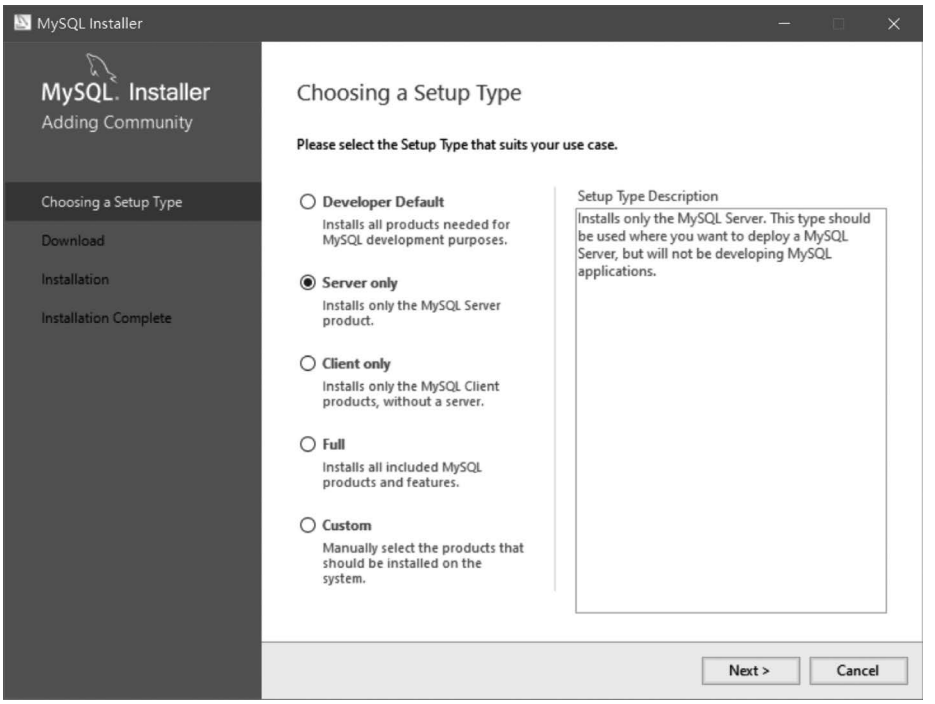



图 5-2　安装类型

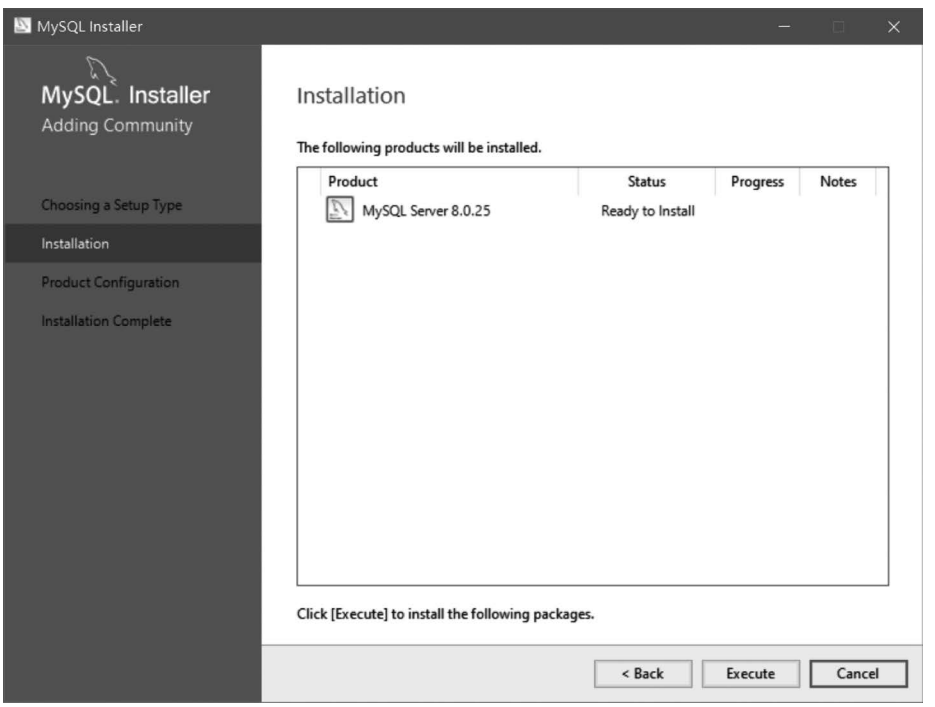



图 5-3　安装对话框

### 2. 配置安装

安装完成后，还需要进行必要的配置过程，其中有两个重要步骤：

（1）配置网络通信端口，如图 5-4 所示，默认通信端口是 3306，如果没有端口冲突，建议不用修改。

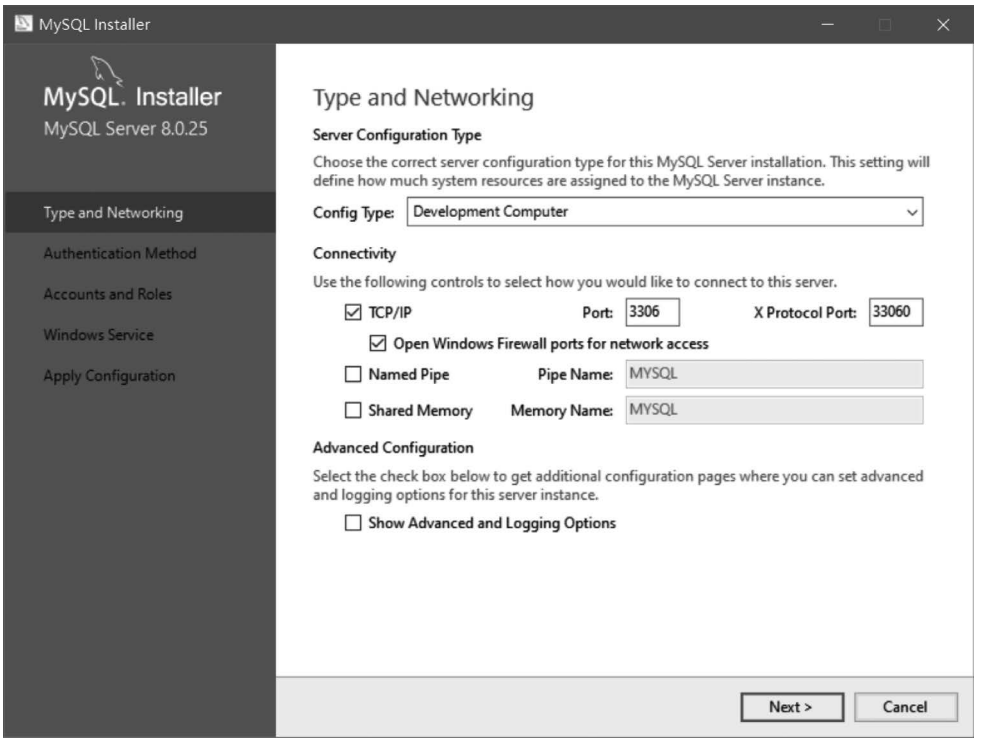



图 5-4　网络配置对话框

（2）配置密码，如图 5-5 所示，配置过程可以为 root 用户设置密码，也可以添加其他普通用户。

### 3. 配置 Path 环境变量

为了使用方便，笔者推荐把 MySQL 安装路径添加到 Path 环境变量中，如图 5-6 所示，打开 Windows 环境变量设置对话框。

双击 Path 环境变量，弹出编辑环境变量对话框，如图 5-7 所示，在此对话框中添加 MySQL 安装路径。

## 5.1.3　客户端登录服务器

如果 MySQL 服务器安装好了，就可以使用了。使用 MySQL 服务器第一步是通过客户端登录服务器。登录服务器可以使用命令提示符窗口（macOS 和 Linux 中终端窗口）或 GUI（图形用户界面）工具登录 MySQL 数据库，笔者推荐使用命令提示符窗口登录，下面介绍命令提示符窗口登录过程。

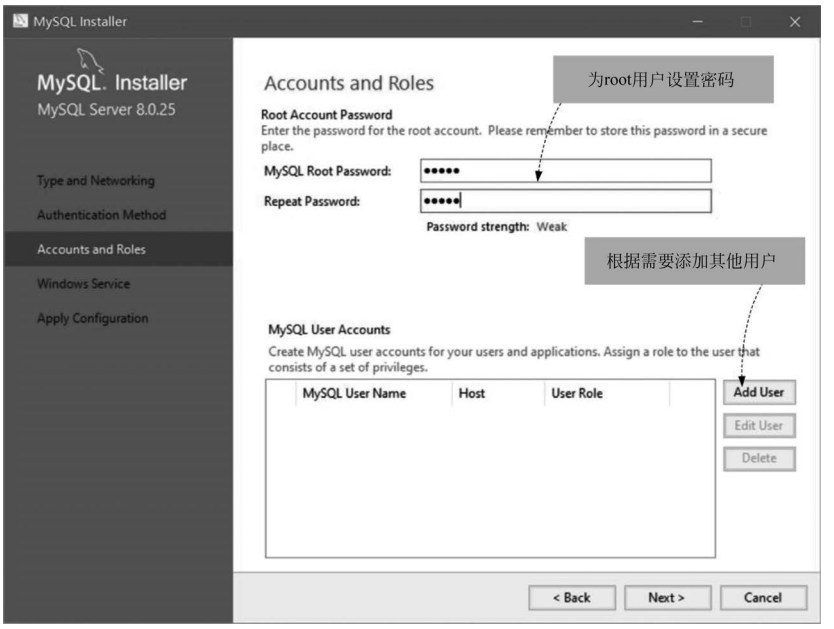



图 5-5　设置用户密码

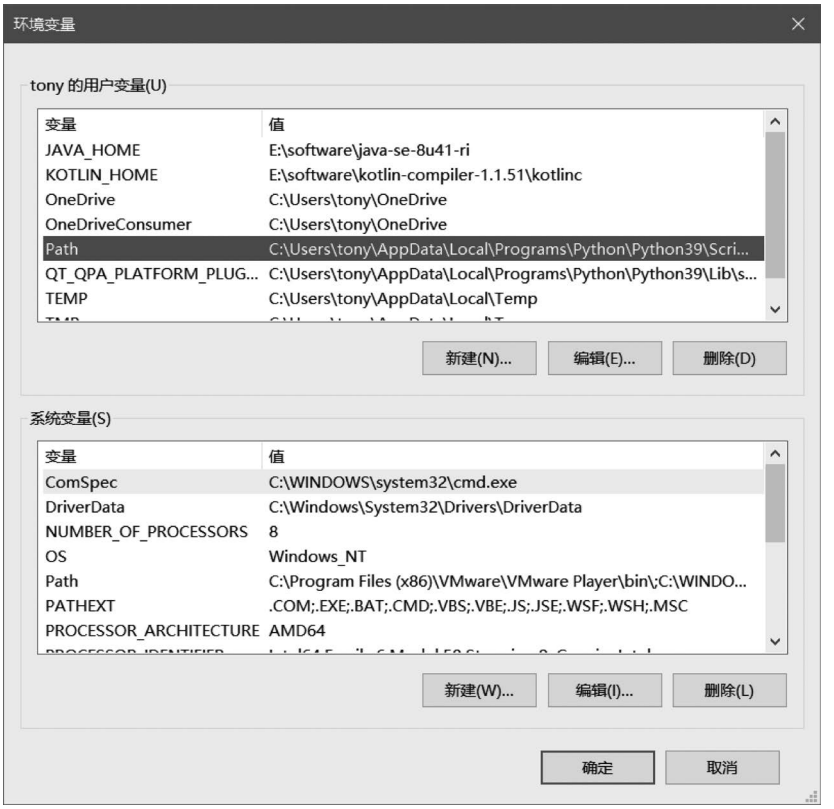



图 5-6　Path 环境变量

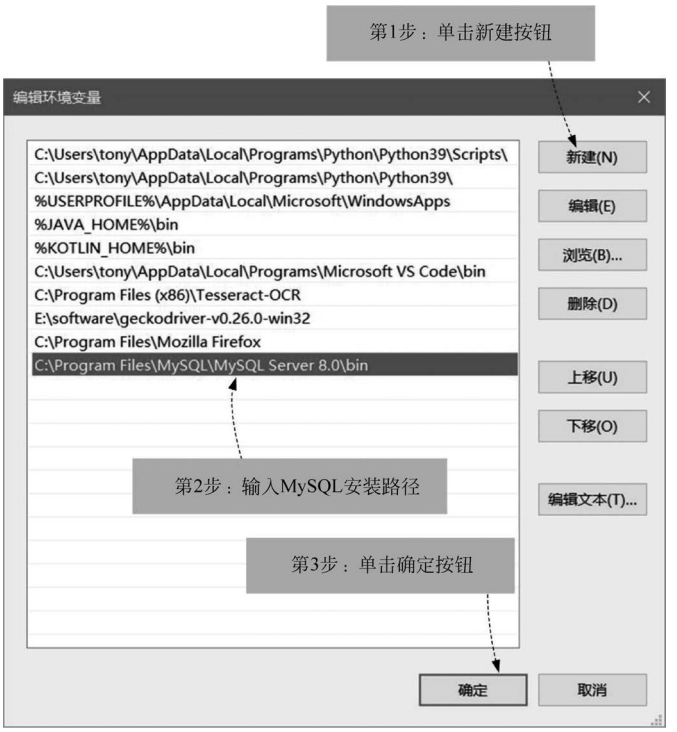



图 5-7　Path 环境变量对话框

使用命令提示符窗口登录服务器完整的指令如下：

```
mysql -h 主机 IP 地址(主机名) -u 用户 -p
```

其中-h、-u、-p 是参数，说明：

（1）-h：是要登录的服务器主机名或 IP 地址，可以是远程的一个服务器主机。注意-h 后面可以没有空格。如果是本机登录可以省略。

（2）-u：是登录服务器的用户，这个用户一定是数据库中存在的，并且具有登录服务器的权限。注意-u 后面可以没有空格。

（3）-p：是用户对应的密码，可以直接-p 后面输入密码，可以在敲回车键后再输入密码。

图 5-8 所示的是 mysql 指令登录本机服务器。

### 5.1.4　图形界面客户端工具

很多人并不习惯使用命令提示符客户端工具来管理和使用 MySQL 数据库，为此可以使用图形界面的客户端工具，这些图形界面工具有很多，考虑到免费且跨平台，笔者推荐使用 MySQL Workbench，它是 MySQL 官方提供的免费、功能较全的图形界面管理工具。

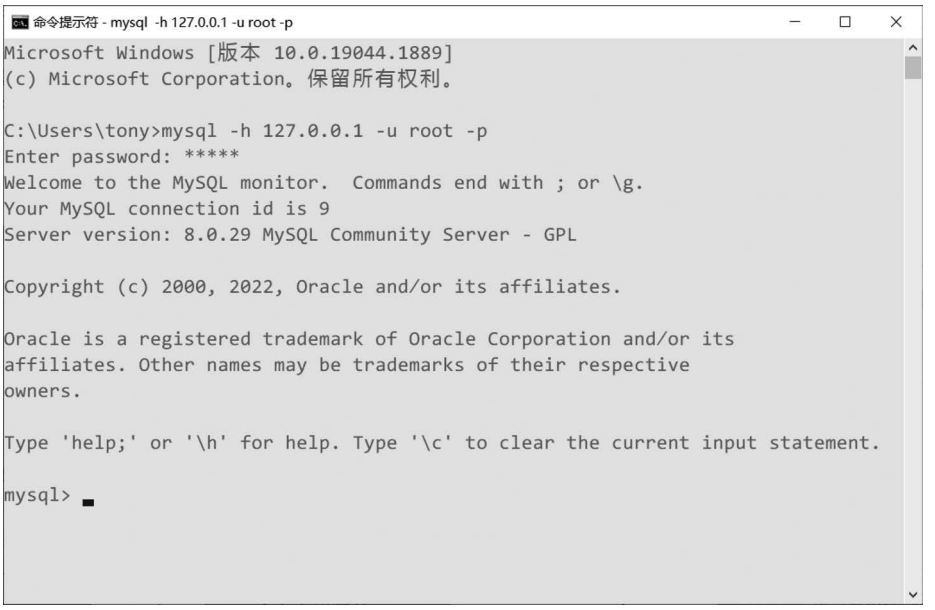



图 5-8　客户端登录服务器

### 1. 安装 MySQL Workbench

在安装 MySQL 过程中，选择 MySQL Workbench 组件，就可以安装和下载 MySQL Workbench。使用 6.1.2 节的 MySQL 社区版安装文件，双击安装文件，启动如图 5-9 所示的 MySQL 安装器。

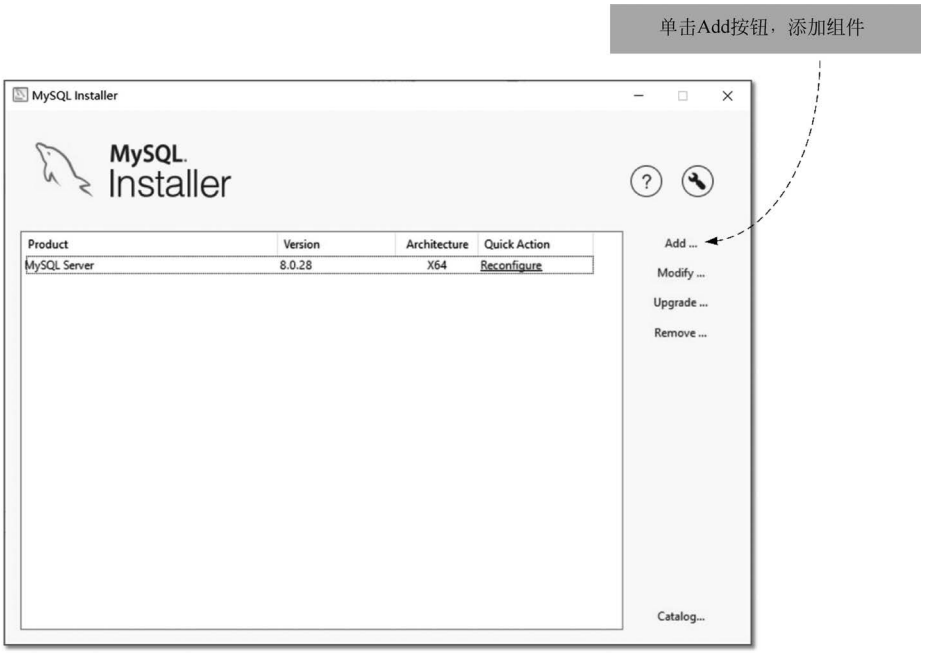



图 5-9　启动 MySQL 安装器

单击 Add 按钮添加组件，进入如图 5-10 所示的安装界面，在此选择要安装的 MySQL Workbench 组件，然后单击 ➡ 按钮将 MySQL Workbench 组件添加到右侧列表准备安装，如图 5-11 所示。

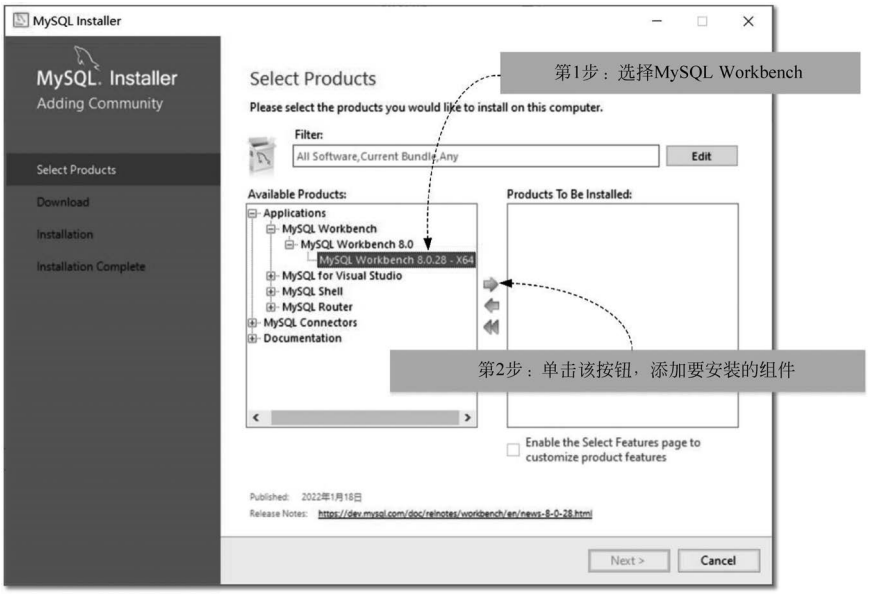



图 5-10　查找 MySQL Workbench 组件

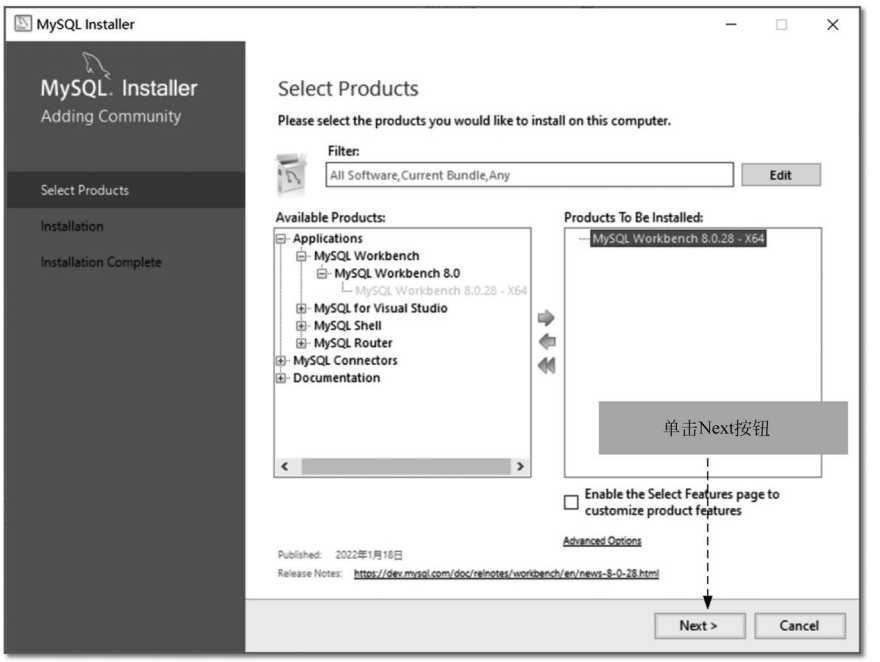



图 5-11　选择 MySQL Workbench 组件

选择好 MySQL Workbench 组件后，单击 Next 按钮，进入如图 5-12 所示的安装界面，单击 Execute 按钮开始安装。

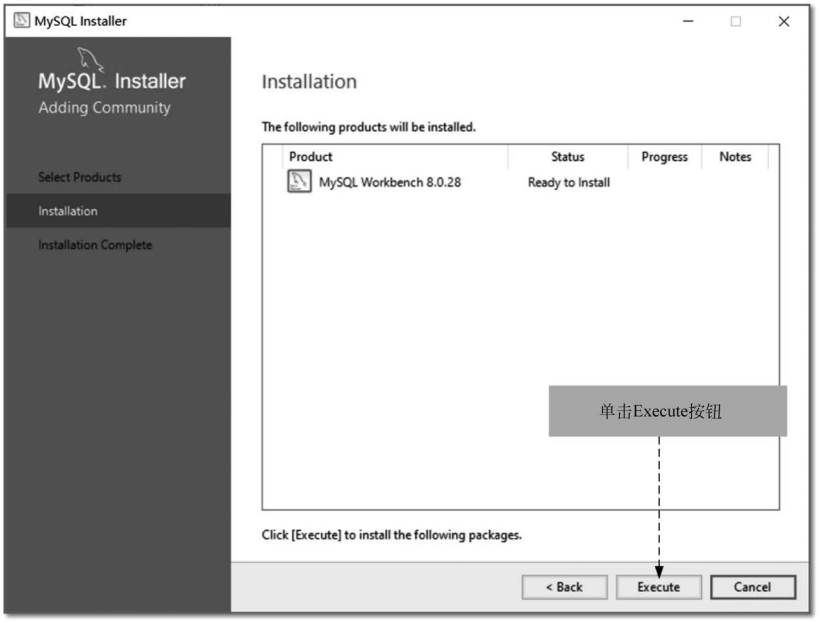



图 5-12　安装执行

在安装前还要下载 MySQL Workbench，如图 5-13 所示，下载完成后单击 Next 按钮开始安装。安装完成后，单击 Finish 按钮，如图 5-14 所示。

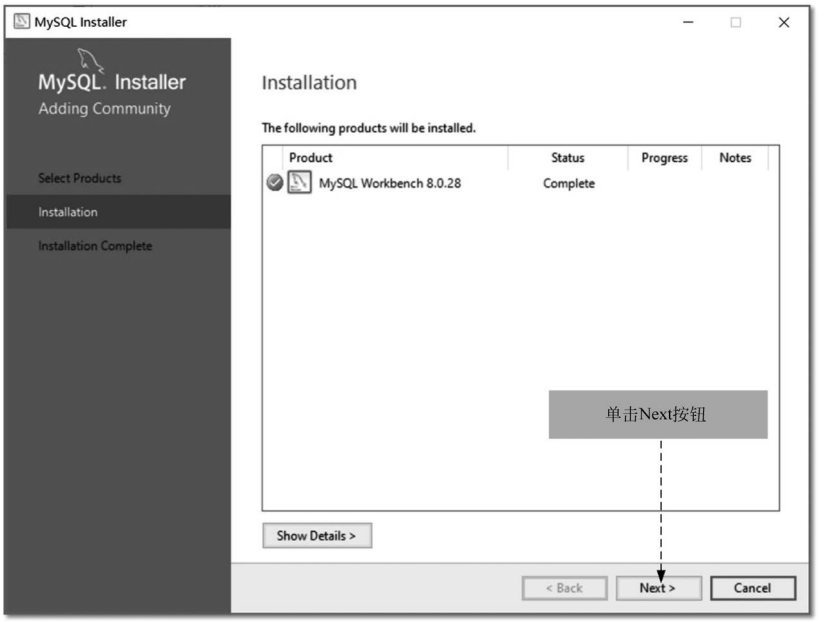



图 5-13　下载 MySQL Workbench

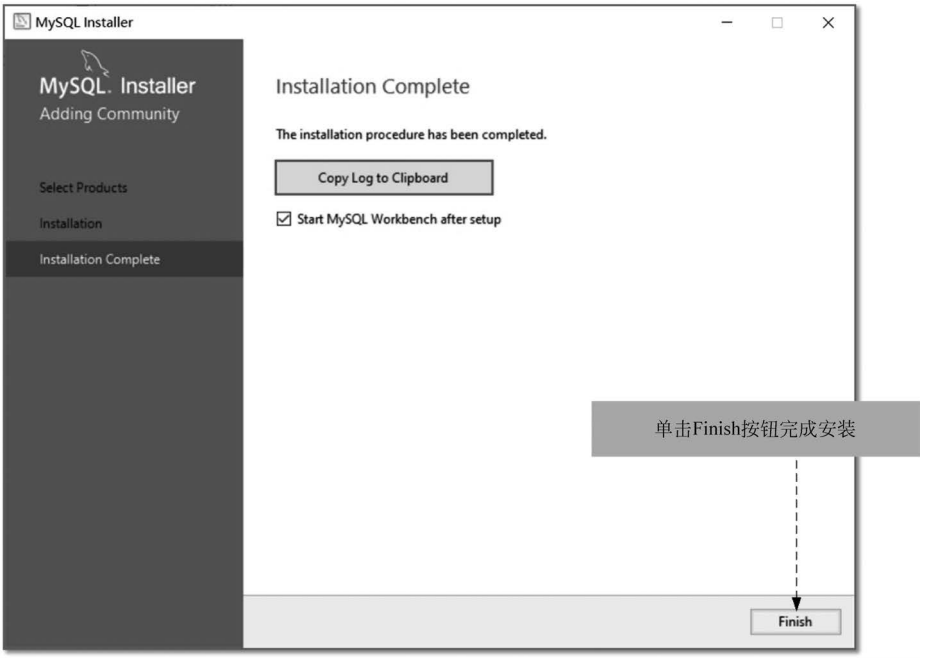



图 5-14　安装完成

### 2. 配置连接数据库

MySQL Workbench 作为 MySQL 数据库客户端管理工具，要想管理数据库，首先需要配置数据库连接。启动 MySQL Workbench，进入如图 5-15 所示的欢迎页面。

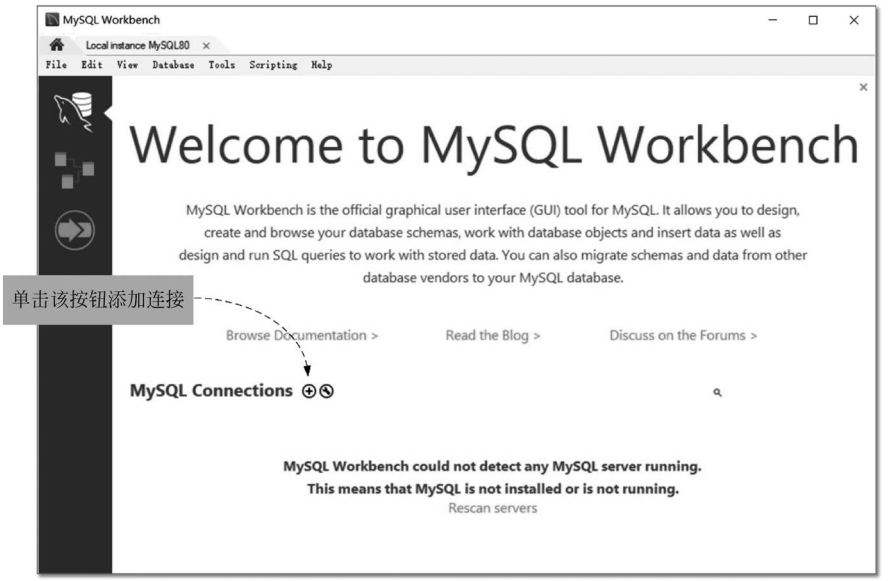



图 5-15　欢迎页面

在 MySQL Workbench 欢迎页面上单击添加按钮 ⊕，进入如图 5-16 所示的 Setup New Connection 对话框，在该对话框中开发人员可以为连接设置一个名字，此外，还需要设置主机名、端口、用户名和密码。设置密码时，需要单击 Store in Vault 按钮，弹出如图 5-17 所示的 Store Password For Connection 对话框。所有项目设置完成后，可以测试一下是否能连接成功，单击 Test Connection 按钮测试连接，如果成功，则弹出如图 5-18 所示的对话框。连接成功，单击 OK 按钮回到欢迎页面，其中 myconnect 是刚刚配置好的连接，如图 5-19 所示。

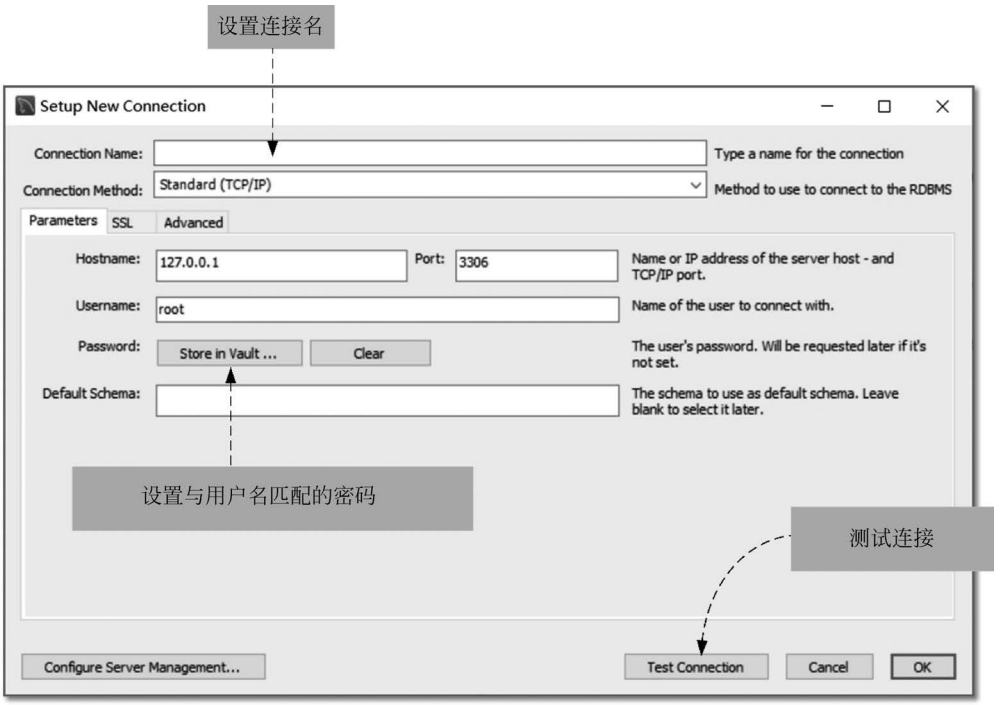



图 5-16　Setup New Connection 对话框


图 5-17　Store Password For Connection 对话框

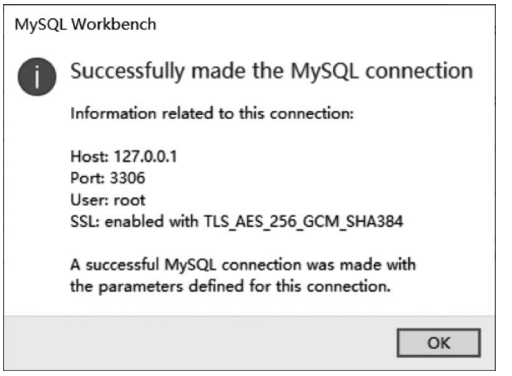



图 5-18　测试连接成功

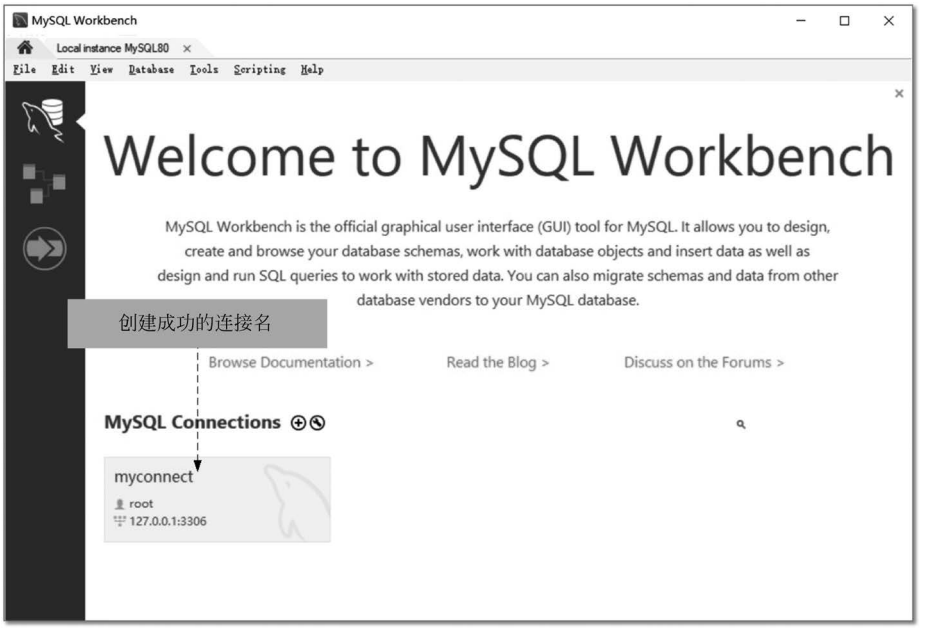



图 5-19　设置完成

### 3. 管理数据库

双击 myconnect 连接就可以登录到 MySQL 工作台，如图 5-20 所示，其中 SCHEMAS 是当前数据库列表，在 MySQL 中 SCHEMAS(模式)就是数据库，其中粗体显示的数据库为当前默认数据库，如果想改变默认数据库，可以右击要设置的数据库，在弹出的快捷菜单选择 Set as Default Schema，就可以设置默认数据库了，如图 5-21 所示。

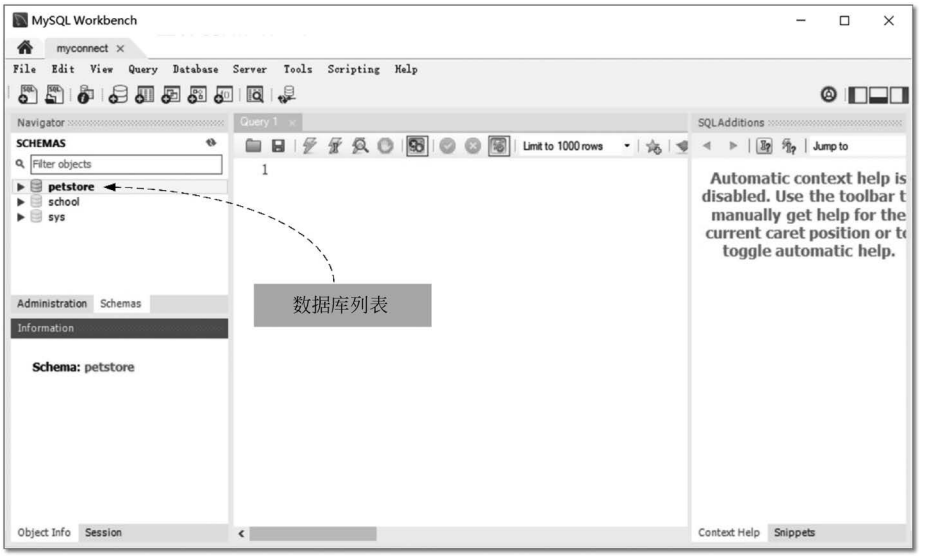



图 5-20　MySQL 工作台


图 5-21　设置默认数据库

注：在图 5-21 所示的快捷菜单中还有 Create Schema 命令，可以创建数据库；Alter Schema 命令可以修改数据库；Drop Schema 命令可以删除数据库。

例如，要创建 school 数据库，则需要选择 Create Schema，弹出如图 5-22 所示的对话框，在 Name 文本框中可以设置数据库名，另外还可以选择数据库的字符集，设置无误后单击 Apply 按钮应用设置。如果取消设置，可以单击 Revert 按钮。

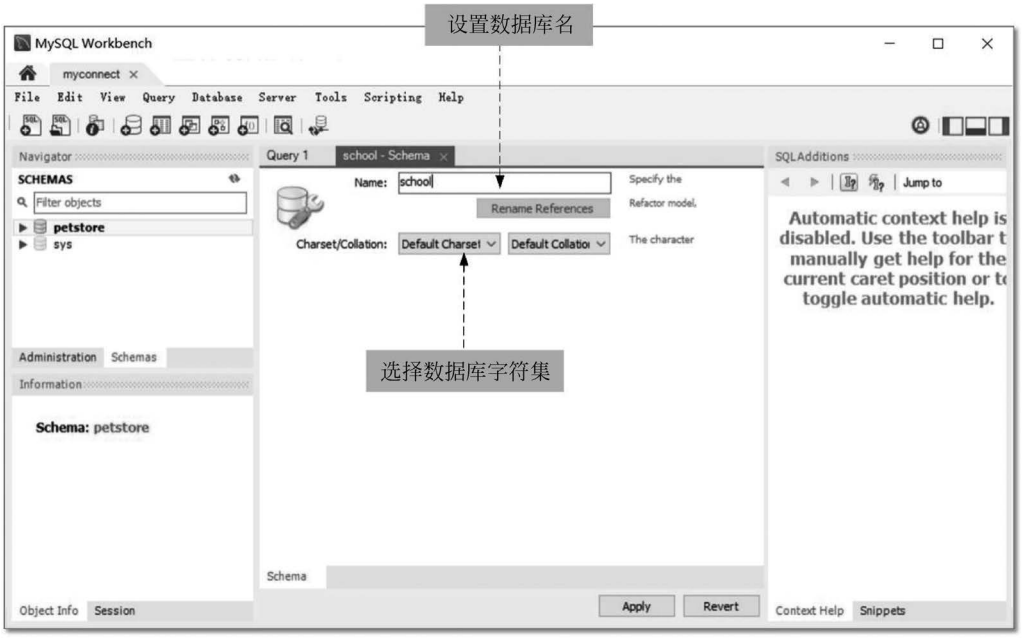



图 5-22　创建数据库

单击 Apply 按钮，弹出如图 5-23 所示的 Apply SQL Script to Database 对话框。确定无误后单击 Apply 按钮创建数据库，然后进入如图 5-24 所示的界面，单击 Finish 按钮，创建完成。

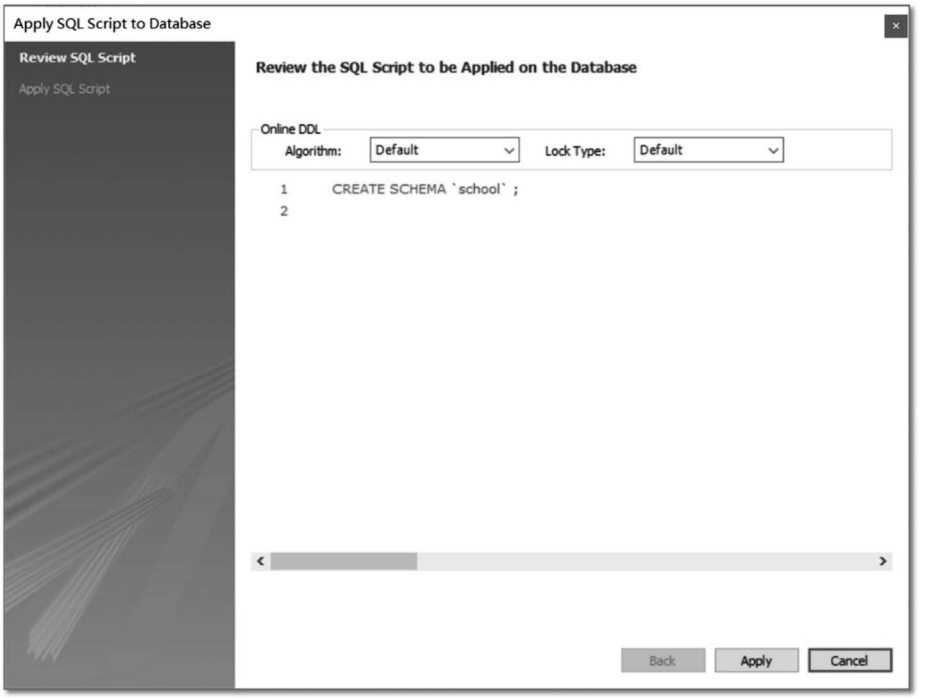



图 5-23　应用脚本对话框

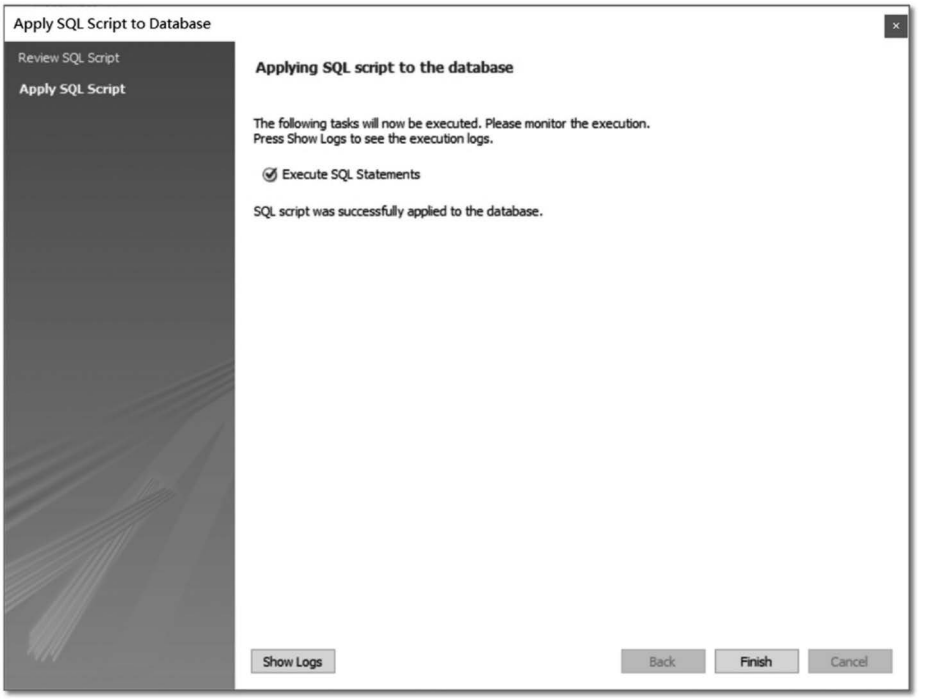



图 5-24　创建完成

有关删除和修改数据库的内容不再赘述。

### 4. 执行 SQL 语句

如果不喜欢使用图形界面向导创建、管理数据库和表，还可以使用 SQL 语句直接操作数据库，要想在 MySQL Workbench 工具中执行 SQL 语句，则需要打开查询窗口。执行菜单命令 File→New Query Tab 或单击快捷按钮可打开查询窗口，如图 5-25 所示。

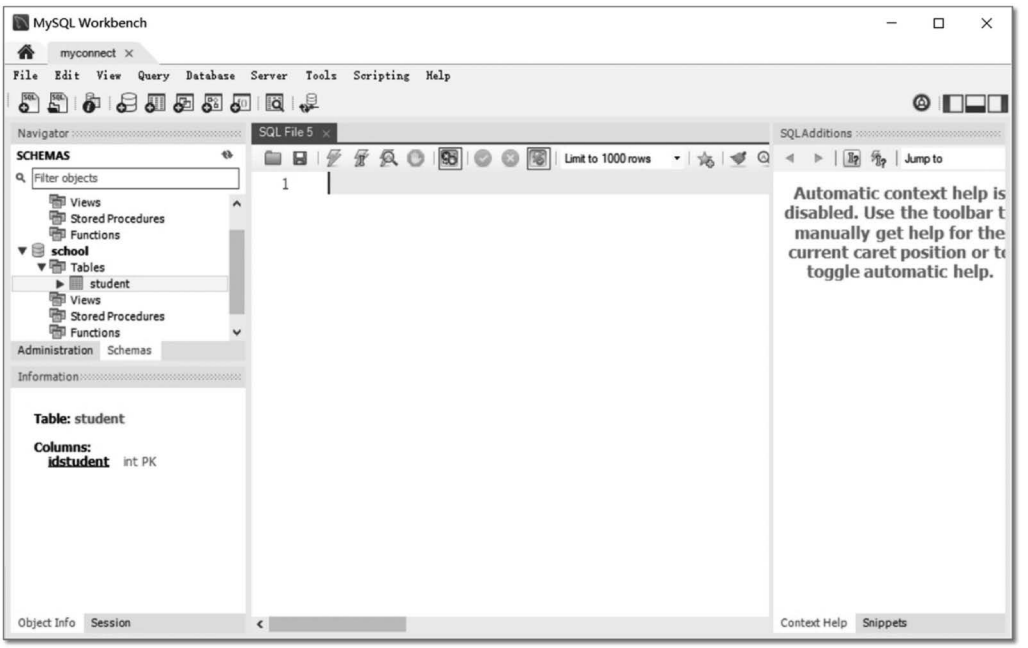



图 5-25　查询窗口

开发人员可以在查询窗口中输入任何 SQL 语句，如图 5-26 所示。可以单击 按钮执行 SQL 语句，注意单击该按钮时，如果有选中的 SQL 语句，则执行选中的 SQL 语句；如果没有选中任何 SQL 语句，则执行当前窗口中全部 SQL 语句。 按钮的功能是执行 SQL 语句到光标所在的位置。

### 5. 执行 SQL 脚本

我们通常会将多条 SQL 语句编写在一个文本文件中，打开 NASDAQ_DB. sql(笔者提供的纳斯达克股票数据)文件如图 5-27 所示。执行该脚本文件的操作如下：

(1) 建创建数据库 nasda；

(2) 在 nasda 建数据库中创建 stocks(股票)表；

(3) 在 stocks 表中插入数据；

(4) 在 nasda 建数据库中创建 historicalquote(股票历史数据)表；

(5) 在 historicalquote 表中插入数据。

在 MySQL Workbench 中可以执行 SQL 脚本文件，首先通过菜单 File→Open SQL Script 打开脚本文件，打开 NASDAQ_DB. sql 文件，如图 5-28 所示。

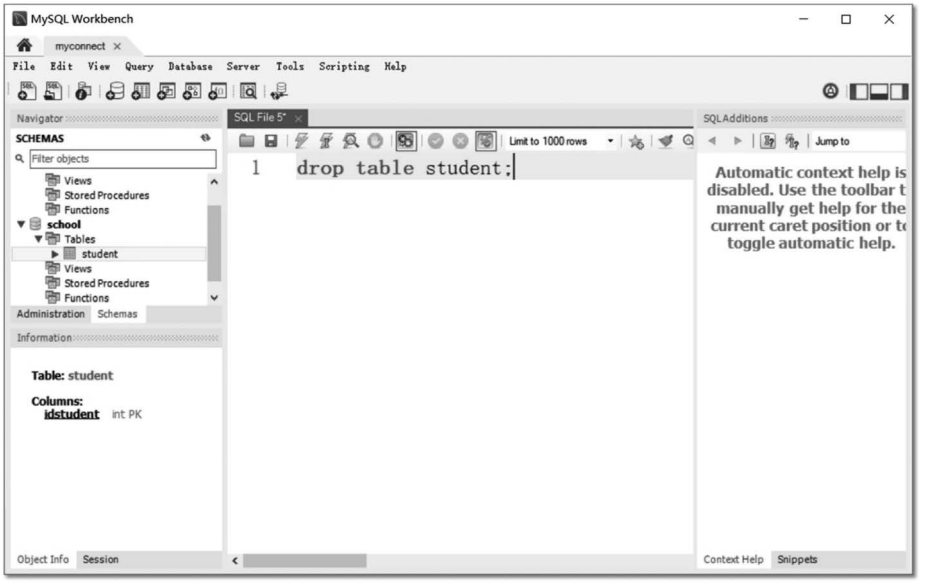



图 5-26　执行 SQL 语句


图 5-27　NASDAQ_DB. sql 脚本文件

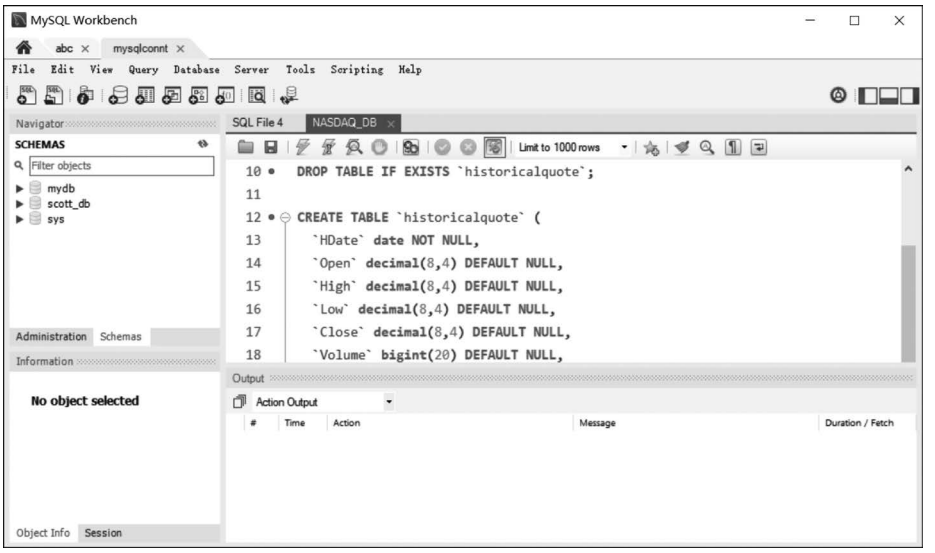



图 5-28　打开 NASDAQ_DB. sql 文件

以单击 按钮执行 SQL 打开脚本文件，具体过程不再赘述。

## 5.1.5　安装 PyMySQL 库

安装 PyMySQL 库可以使用如下 pip 指令：

```
pip install PyMySQL
```

在 Windows 平台命令提示符中安装 PyMySQL 库安装过程如图 5-29 所示。其他平台安装过程也是类似的，这里不再赘述。


图 5-29　安装 PyMySQL 库

另外，由于 MySQL8 采用了更加安全的加密方法，因此还需要安装 cryptography 库。安装 cryptography 库可以使用如下 pip 指令：

```
pip install cryptography
```

在 Windows 平台命令提示符中安装 cryptography 库安装过程如图 5-30 所示。其他平台安装过程也是类似的，这里不再赘述。

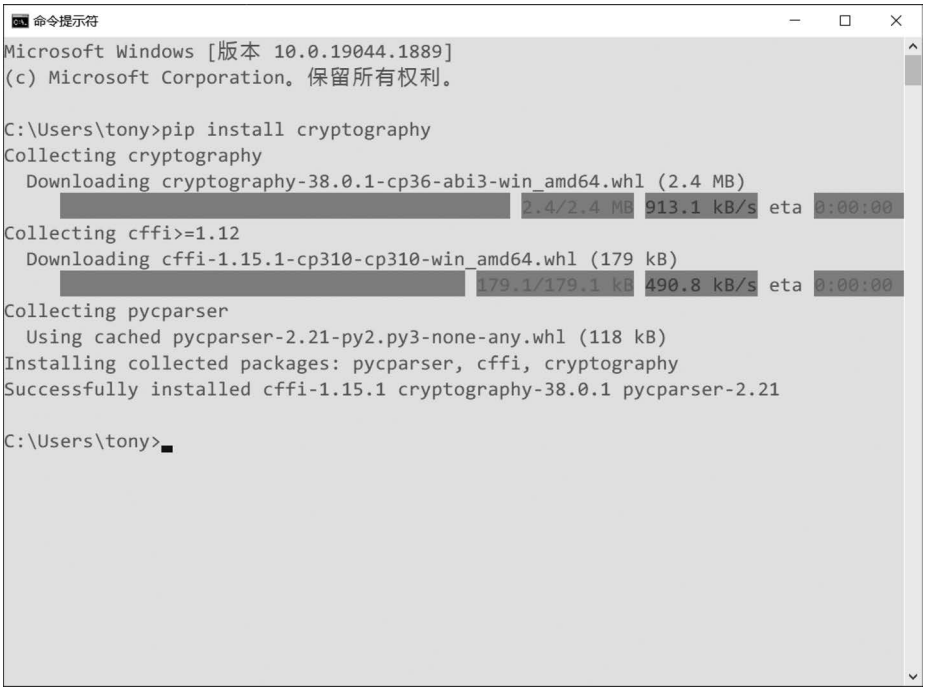



图 5-30　安装 cryptography 库

## 5.1.6　访问数据库一般流程

访问数据库操作分为两大类：查询数据和修改数据。

### 1. 查询数据

查询数据就是通过 Select 语句查询数据库，其流程如图 5-31 所示，该流程有如下 6 个步骤。

（1）**建立数据库连接**。数据库访问的第一步是进行数据库连接。建立数据库连接可以通过 PyMySQL 库提供的 connect(parameters...)方法实现，该方法根据 parameters 参数连接数据库，连接成功返回 Connection(数据库连接)对象。

（2）**创建游标对象**。游标是暂时保存了 SQL 操作所获得的数据，创建游标是通过 Connection 对象的 cursor()方法创建的。

（3）**执行查询操作**。执行 SQL 操作是通过游标对象的 execute(sql)方法实现的，其中

参数 sql 表示要执行 SQL 语句字符串。

（4）**提取结果集**。执行 SQL 操作会返回结果集对象，结果集对象的结构与数据库表类似，由记录和字段构成。提取结果集可以通过游标的 fetchall()或 fetchone()方法实现，fetchall()是提取结果集中的所有记录，fetchone()方法是提取结果集中的一条记录。

（5）**关闭游标**。数据库游标使用完成之后，需要关闭游标，关闭游标可以释放资源。

（6）**关闭数据库连接**。数据库操作完成之后，需要关闭数据库连接，关闭连接也可以释放资源。

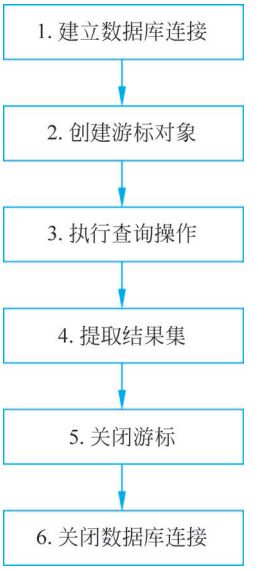



图 5-31　查询数据步骤

### 2. 修改数据

修改数据就是通过 Insert、Update 和 Delete 等语句修改数据，其流程如图 5-32 所示。修改数据与查询数据流程类似，也有 6 个步骤。但是修改数据时，如果执行 SQL 操作成功时需要提交数据库事务，如果失败则需要回滚数据库事务。另外，修改数据时不会返回结果集，也就不能从结果集中提取数据了。

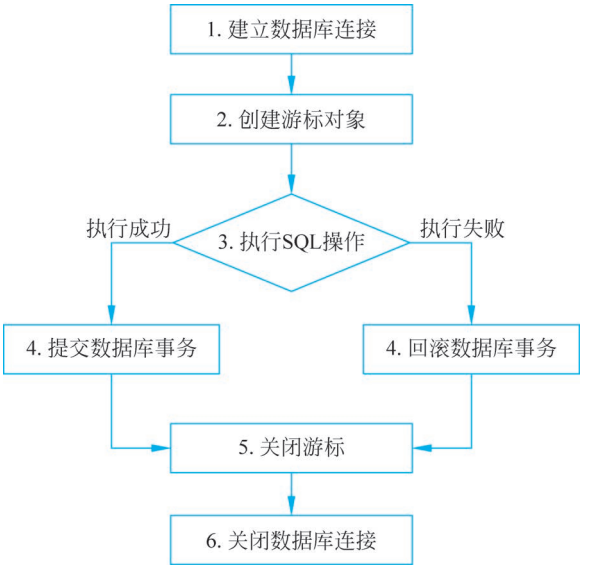
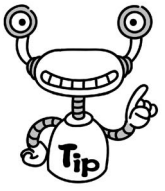



图 5-32　查询修改数据步骤




数据库事务通常包含了多个对数据库的读/写操作，这些操作是有序的。若事务被提交

给了数据库管理系统，则数据库管理系统需要确保该事务中的所有操作都成功完成，结果被永久保存在数据库中。如果事务中有的操作没有成功完成，则事务中的所有操作都需要被回滚，回到事务执行前的状态。

（1）连接数据库代码：

```
import pymysql
# 建立数据库连接
cnx = pymysql.connect(user='username', password='password',
                      host='host_address',
                      database='database_name')
```

在上述代码中，需要将 username、password、host_address 和 database_name 替换为实际的数据库连接信息。

（2）创建游标对象：在建立数据库连接后，需要创建一个游标对象，用于执行 SQL 语句。

```
cursor = cnx.cursor()
```

（3）执行 SQL 查询：使用游标对象执行 SQL 查询语句，获取数据库中的数据。

```
query = "SELECT * FROM table_name"
cursor.execute(query)

# 获取查询结果
result = cursor.fetchall()
```

在上述代码中，table_name 应替换为实际的表名。

（4）执行 SQL 插入/更新：使用游标对象执行 SQL 插入或更新语句，将数据写入 MySQL 数据库。

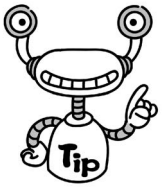
```
insert_query = "INSERT INTO table_name (column1, column2) VALUES (%s, %s)"
data = ('value1', 'value2')
cursor.execute(insert_query, data)
# 提交事务
cnx.commit()
```

在上述代码中，table_name 应替换为实际的表名，column1 和 column2 应替换为实际的列名，value1 和 value2 应替换为要插入的实际值。

（5）关闭游标和数据库连接：在完成所有数据库操作后，需要关闭游标和数据库连接。

```
cursor.close()
cnx.close()
```

这样就完成了对 MySQL 数据库的读写操作。



以上代码示例是基本的 MySQL 数据库读写操作，具体的 SQL 查询和插入/更新语句需要根据实际情况进行调整。此外，还应该考虑异常处理、数据类型转换等方面的处理。

## 5.1.7　案例 1：访问苹果股票数据

下面通过一个案例介绍如何使用 Python 语言访问 MySQL 数据库。

案例背景：

该案例的该项目的数据库设计模型如图 5-33 所示，项目中包含两个数据表：股票信息表(Stocks)和股票历史价格表(HistoricalQuote)。

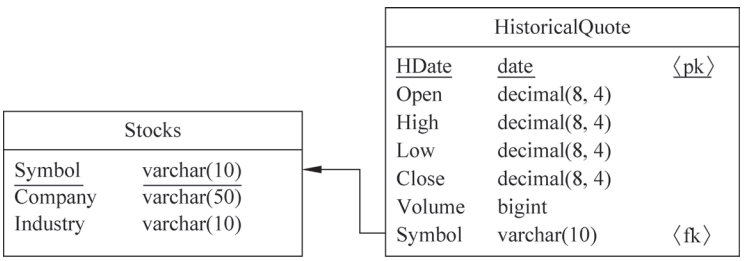



图 5-33　数据库设计模型

数据库设计模型中各个表说明如下：

### 1. 股票信息表

股票信息表(英文名 Stocks)是纳斯达克股票，股票代号(英文名 Symbol)是主键，股票信息表结构如表 5-1 所示。该项目目前的功能不包括维护股票信息表，所需要数据在创建数据表时预先插入。

表 5-1　股票信息表结构

| 字段名 | 数据类型 | 长度 | 精度 | 主键 | 外键 | 备注 |
|---|---|---|---|---|---|---|
| Symbol | varchar(10) | 10 | — | 是 | 否 | 股票代号 |
| Company | varchar(50) | 50 | — | 否 | 否 | 公司 |
| Industry | varchar(10) | 10 | — | 否 | 否 | 所属行业 |

### 2. 股票历史价格表

股票历史价格表(英文名 HistoricalQuote)是某一只股票的历史价格表，交易日期(英文名 HDate)是主键，股票历史价格表结构如表 5-2 所示。

表 5-2　股票历史价格表结构

| 字段名 | 数据类型 | 长度 | 精度 | 主键 | 外键 | 备注 |
|---|---|---|---|---|---|---|
| HDate | date | | — | 是 | 否 | 交易日期 |
| Open | decimal(8,4) | 8 | 4 | 否 | 否 | 开盘价 |
| High | decimal(8,4) | 8 | 4 | 否 | 否 | 最高价 |
| Low | decimal(8,4) | 8 | 4 | 否 | 否 | 最低价 |

续表

| 字段名 | 数据类型 | 长度 | 精度 | 主键 | 外键 | 备注 |
|---|---|---|---|---|---|---|
| Close | decimal(8,4) | 8 | 4 | 否 | 否 | 收盘价 |
| Volume | bigint | | — | 否 | 否 | 成交量 |
| Symbol | varchar(10) | 10 | — | 否 | 是 | 股票代号 |

编写及执行 SQL 脚本这里不再赘述。

那么根据股票代号查询股历史数据的 Python 代码如下：

```
import pymysql

# 建立数据库连接
cnx = pymysql.connect(
    host = '127.0.0.1',
    user = 'root',
    password = '12345',
    database = 'nasdaq'
)

# 创建游标对象
cursor = cnx.cursor()
# 定义查询语句和参数
query = "SELECT * FROM historicalquote WHERE Symbol = %s"          ①
params = ('AAPL',)

# 执行查询语句
cursor.execute(query, params)                                      ②

# 获取查询结果
result = cursor.fetchall()                                         ③

# 输出查询结果
for row in result:                                                 ④
    print(row)

# 关闭游标和数据库连接
cursor.close()
cnx.close()
```

上述代码执行后，输出结果如下。

```
(datetime.date(2018, 2, 1), Decimal('167.1650'), Decimal('168.6200'), Decimal('166.7600'),
Decimal('167.7800'), 44453230, 'AAPL')
...
(datetime.date(2023, 1, 31), Decimal('166.8700'), Decimal('168.4417'), Decimal('166.5000'),
Decimal('167.4300'), 32234520, 'AAPL')
```

代码解释如下：

- 代码第①行在这一行中，定义了查询语句，使用了参数占位符 %s 来表示待传入的

参数。

- 代码第②行 execute()方法用于执行查询语句，并将参数值传递给查询语句中的占位符。在这里，params 变量包含了要绑定的参数值。
- 代码第③行使用 fetchall() 方法获取查询结果集。这个方法会返回一个包含所有查询结果的列表。
- 代码第④行使用 for 循环遍历结果列表，并输出每一行的内容。

最后，在代码的末尾，通过调用 close()方法关闭游标对象和数据库连接，释放相关资源。

请注意，代码中的连接参数(host、user、password、database)是示例值，需要根据实际情况进行修改。同时，查询的表名、列名以及参数值也需要根据数据库的实际结构进行调整。

这段代码的作用是连接到数据库，执行一条查询语句，将结果打印输出，并关闭数据库连接。在这个示例中，查询的条件是 Symbol = 'AAPL'，即查询 historicalquote 表中 Symbol 列为 'AAPL' 的数据行，AAPL 是苹果股票代号。

## 5.2　使用 Pandas 读写 MySQL 数据库

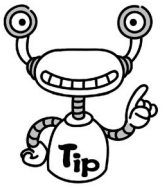
Pandas 提供了用于读取和写入 MySQL 数据库的函数。下面是几个常用的函数：

(1) pd.read_sql(query, con)：从 MySQL 数据库中执行查询语句并返回结果作为 DataFrame。query 是查询语句，con 是与 MySQL 数据库建立的连接对象。

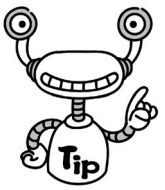
(2) df.to_sql(name, con, if_exists='fail', index=False)：将 DataFrame 中的数据写入 MySQL 数据库中的表。name 是目标表的名称，con 是与 MySQL 数据库建立的连接对象，if_exists 是指定写入操作的行为，默认为'fail'，表示如果目标表已存在，则引发异常。index=False 表示不将 DataFrame 的索引写入数据库。

(3) pd.read_sql_table(table_name, con)：从 MySQL 数据库中读取整个表的数据并返回结果作为 DataFrame。table_name 是表的名称，con 是与 MySQL 数据库建立的连接对象。

这些函数可以方便地与 MySQL 数据库进行数据的读取和写入操作。




建议在使用 Pandas 读取数据时，使用 SQLAlchemy 的 connectable (engine/connection)创建数据库连接对象，而不推荐使用 pymysql.connect()来创建数据库连接对象。这是因为使用 SQLAlchemy 的 connectable 可以避免不兼容性问题。




SQLAlchemy是一个Python SQL工具和对象关系映射(ORM)库，它提供了一种与数据库进行交互的高级抽象和灵活性。它支持多种数据库后端，并提供了统一的API，使得在不同数据库之间切换变得更加容易。

使用SQLAlchemy，你可以执行以下操作：

(1) 创建数据库连接：通过create_engine()函数创建数据库连接对象，指定数据库类型、主机、用户名、密码和数据库名称等参数。

(2) 执行SQL查询：使用连接对象的execute()方法执行SQL查询语句。

(3) 获取查询结果：通过执行查询后返回的结果集对象，使用fetchall()、fetchone()等方法获取查询结果。

(4) 执行事务操作：使用连接对象的begin()、commit()和rollback()等方法执行数据库事务操作。

(5) ORM映射：使用SQLAlchemy的ORM功能，将数据库表映射为Python对象，实现面向对象的数据库操作，简化了数据库的访问和操作。

通过使用SQLAlchemy，我们可以更方便地操作数据库，实现数据读取、写入和查询等功能，并且具有较高的灵活性和可扩展性。它也被广泛应用于许多Python项目和框架中，如Django、Flask等。

示例：使用Pandas从数据库读取股票数据

下面我们是要使用Pandas库的pd.read_sql_table(table_name, con)函数从historicalquote(股票历史数据)表读取数据，具体代码如下：

```
import pandas as pd
import pymysql
from sqlalchemy import create_engine

# 创建数据库连接
engine = create_engine('mysql+pymysql://root:12345@localhost/nasdaq')

# 从MySQL数据库中读取数据
df = pd.read_sql_table('historicalquote', engine)
print(df)
```

示例运行后，输出结果如下：

```
        HDate      Open      High       Low     Close    Volume    Symbol
0  2018-02-01  167.165  168.6200  166.7600  167.78  44453230   AAPL
1  2018-02-02  166.000  166.8000  160.1000  160.50  85957050   AAPL
```

```
2   2018-02-05  159.100  163.8800  156.0000  156.49  72215320   AAPL
3   2018-02-06  154.830  163.7200  154.0000  163.03  68171940   AAPL
4   2018-02-07  163.085  163.4000  159.0685  159.54  51467440   AAPL
...        ...      ...       ...       ...     ...       ...    ...
60 2023-01-29  170.160  170.1600  167.0700  167.96  50565420   AAPL
61 2023-01-30  165.525  167.3700  164.7000  166.97  45635470   AAPL
62 2023-01-31  166.870  168.4417  166.5000  167.43  32234520   AAPL
63 2023-07-01  100.250  105.5000   98.7500  102.80    100000   AAPL
64 2023-07-02  103.000  106.2000  101.5000  105.40    120000   AAPL

[65 rows x 7 columns]
```

上述代码使用了 pd.read_sql_table('historicalquote', engine)函数读取 historicalquote 表查询所有字段，如果只是关系部分字段，这可以使用 pd.read_sql(query, engine)函数。

那么对应代码如下：

```
import pandas as pd
import pymysql
from sqlalchemy import create_engine

# 创建数据库连接
engine = create_engine('mysql+pymysql://root:12345@localhost/nasdaq')

# 从 MySQL 数据库中读取数据
query = "SELECT HDate,Open,High, Low,Close FROM historicalquote WHERE Symbol = 'AAPL'"
df = pd.read_sql(query, engine)
print(df)
```

示例运行后，输出结果如下：

```
         HDate      Open      High       Low      Close
0   2018-02-01  167.165  168.6200  166.7600  167.78
1   2018-02-02  166.000  166.8000  160.1000  160.50
2   2018-02-05  159.100  163.8800  156.0000  156.49
3   2018-02-06  154.830  163.7200  154.0000  163.03
4   2018-02-07  163.085  163.4000  159.0685  159.54
...        ...      ...       ...       ...     ...
60  2023-01-29  170.160  170.1600  167.0700  167.96
61  2023-01-30  165.525  167.3700  164.7000  166.97
62  2023-01-31  166.870  168.4417  166.5000  167.43
63  2023-07-01  100.250  105.5000   98.7500  102.80
64  2023-07-02  103.000  106.2000  101.5000  105.40

rows x 5 columns]
```

## 5.3　JSON 数据交换格式

JSON 是一种轻量级的数据交换格式。所谓轻量级，是与 XML 文档结构相比而言的。描述项目的字符少，所以描述相同数据所需的字符个数要少，那么传输速度就会提高，而流量也会减少。

### 5.3.1 JSON 文档结构

由于 Web 和移动平台开发对流量的要求是尽可能少，对速度的要求是尽可能快，而轻量级的数据交换格式 JSON 就成为理想的数据交换格式。

构成 JSON 文档的两种结构分别为对象(object)和数组(array)。其中，对象是“名称：值”对集合，它类似于 Python 中 Map 类型，而数组是一连串元素的集合。

JSON 对象是一个无序的“名称/值”对集合，一个对象以“{”开始，以“}”结束。每个“名称”后跟一个“:”，“名称：值”对之间使用“,”分隔，“名称”是字符串类型(string)，“值”可以是任何合法的 JSON 类型。JSON 对象的语法表如图 5-34 所示。

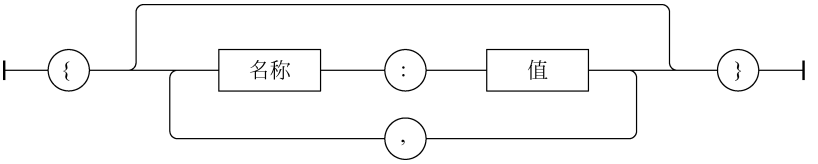



图 5-34 JSON 对象的语法表

下面是一个 JSON 对象的例子：

```
{
    "name":"a.htm",
    "size":345,
    "saved":true
}
```

JSON 数组是值的有序集合，以“[”开始，以“]”结束，值之间使用“,”分隔。JSON 数组的语法表如图 5-35 所示。

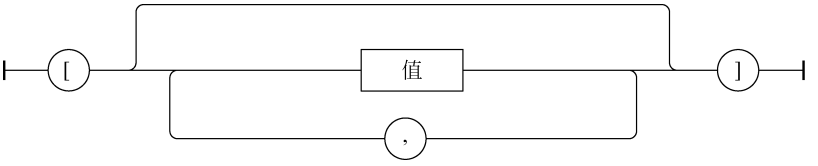



图 5-35 JSON 数组的语法表

下面是一个 JSON 数组的例子：

```
["text","html","css"]
```

JSON 数组中的值可以是双引号括起来的字符串、数字、对象、数组、true、false 或 null，而且这些结构可以嵌套。JSON 值的语法结构图如图 5-36 所示。

### 5.3.2 JSON 数据编码

在 Python 程序中要想将 Python 数据网络传输和存储，可以将 Python 数据转换为 JSON 数据再进行传输和存储，这个过程称为“编码(encode)”。

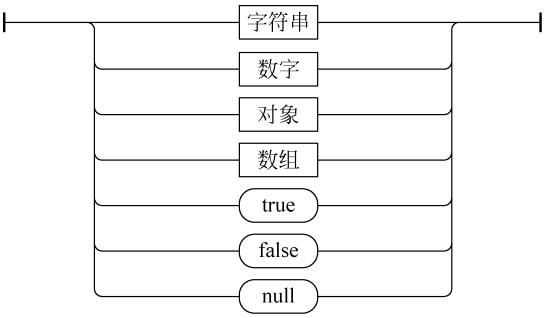
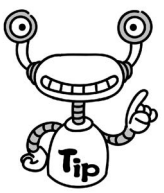



图 5-36　JSON 值的语法结构图

在编码过程中 Python 数据转换为 JSON 数据的映射关系如表 5-3 所示。

表 5-3　Python 数据与 JSON 数据映射关系

| Python | JSON | Python | JSON |
|---|---|---|---|
| 字典 | 对象 | True | true |
| 列表、元组 | 数组 | False | false |
| 字符串 | 字符串 | None | null |
| 整数、浮点等数字类型 | 数字 | | |




JSON 数据在网络传输或保存到磁盘中时，推荐使用 JSON 对象，偶尔也使用 JSON 数组。所以一般情况下只有 Python 的字典、列表和元组才需要编码，Python 字典编码 JSON 对象；Python 列表和元组编码 JSON 数组。

Python 提供的内置模块 json 可以帮助实现 JSON 的编码和解码，JSON 编码使用 dumps()和 dump()函数，dumps()函数将编码的结果以字符串形式返回，dump()函数将编码的结果保存到文件对象(类似文件对象或流)中。

下面具体介绍 JSON 数据编码过程，示例代码如下：

```
import json

# 准备数据
py_dict = {'name': 'tony', 'age': 30, 'sex': True}   # 创建字典对象
py_list = [1, 3]                                     # 创建列表对象
py_tuple = ('A', 'B', 'C')                           # 创建元组对象

py_dict['a'] = py_list                               # 添加列表到字典中
py_dict['b'] = py_tuple                              # 添加元组到字典中
```

```
print(py_dict)
print(type(py_dict))                                    # <class 'dict'>

# 编码过程
json_obj = json.dumps(py_dict)                                          ①
print(json_obj)
print(type(json_obj))                                   # <class 'str'>

# 编码过程
json_obj = json.dumps(py_dict, indent=4)                                ②
# 漂亮的格式化字符串后输出
print(json_obj)

# 写入 JSON 数据到 data1.json 文件
with open('data/data1.json', 'w') as f:
    json.dump(py_dict, f)                                               ③

# 写入 JSON 数据到 data2.json 文件
with open('data/data2.json', 'w') as f:
    json.dump(py_dict, f, indent=4)                                     ④
```

上述代码运行，输出结果如下：

```
{'name': 'tony', 'age': 30, 'sex': True, 'a': [1, 3], 'b': ('A', 'B', 'C')}
<class 'dict'>
{"name": "tony", "age": 30, "sex": true, "a": [1, 3], "b": ["A", "B", "C"]}
<class 'str'>
{
    "name": "tony",
    "age": 30,
    "sex": true,
    "a": [
        1,
        3
    ],
    "b": [
        "A",
        "B",
        "C"
    ]
}
```

解释如下：

- 上述代码第①行是对 Python 字典对象 py_dict 进行编码，编码的结果是返回字符串，这个字符串中没有空格和换行等字符，可见减少字节数适合网络传输和保存。
- 代码第②行也是对 Python 字典对象 py_dict 进行编码，在 dumps()函数中使用了参数 indent。indent 可以格式化字符串，indent＝4 表示缩进 4 个空格，这种漂亮的格式化的字符串，主要用于显示和日志输出，但不适合网络传输和保存。

- 代码第③行和第④行是 dump()函数将编码后的字符串保存到文件中，dump()与 dumps()函数具有类似的参数，这里不再赘述。

### 5.3.3　JSON 数据解码

编码的相反过程是“解码”(decode)，即将 JSON 数据转换为 Python 数据。从网络中接收或从磁盘中读取 JSON 数据时，需要解码为 Python 数据。

在编码过程中，JSON 数据转换为 Python 数据的映射关系如表 5-4 所示。

表 5-4　JSON 数据与 Python 数据映射关系

| JSON | Python | JSON | Python |
|---|---|---|---|
| 对象 | 字典 | 实数数字 | 浮点 |
| 数组 | 列表 | true | True |
| 字符串 | 字符串 | false | False |
| 整数数字 | 整数 | null | None |

json 模块提供的解码函数是 loads()和 load()，loads()函数将 JSON 字符串数据进行解码，返回 Python 数据，load()函数读取文件或流，对其中的 JSON 数据进行解码，返回结果为 Python 数据。

下面具体介绍 JSON 数据解码过程，示例代码如下：

```
import json

# 准备数据
json_obj = r'{"name": "tony", "age": 30, "sex": true, "a": [1, 3], "b": ["A", "B", "C"]}'
                                                                    ①

py_dict = json.loads(json_obj)                                      ②
print(type(py_dict))                              # <class 'dict'>
print(py_dict['name'])
print(py_dict['age'])
print(py_dict['sex'])

py_lista = py_dict['a']                           # 取出列表对象
print(py_lista)
py_listb = py_dict['b']                           # 取出列表对象
print(py_listb)

# 读取 JSON 数据到 data2.json 文件
with open('data/data2.json', 'r') as f:
    data = json.load(f)                                             ③
    print(data)
    print(type(data))                             # <class 'dict'>
```

解释如下：

- 代码第①行是一个表示 JSON 对象的字符串。
- 代码第②行是对 JSON 对象字符串进行解码，返回 Python 字典对象。

- 代码第③行是从 data2.json 文件中读取 JSON 数据解析解码，返回 Python 字典对象。

### 5.3.4 案例 2：解码搜狐证券贵州茅台股票数据

我们在 3.3.3 节介绍过使用 Selenium 从搜狐证券网爬取贵州茅台股票数据，事实上网页中的股票数据，通过如下网址返回：

http://q.stock.sohu.com/hisHq?code=cn_600519&stat=1&order=D&period=d&callback=historySearchHandler&rt=jsonp&0.8115656498417958

直接将网址在浏览器中打开，如图 5-37 所示，浏览器展示了一个字符串。从返回的字符串可见，并不是一个有效的 JSON 数据，而 JSON 数据是放置在 historySearchHandler(…)中的，historySearchHandler 应该是一个 JavaScript 变量或函数，开发人员只需要关心括号中的 JSON 字符串就可以了。

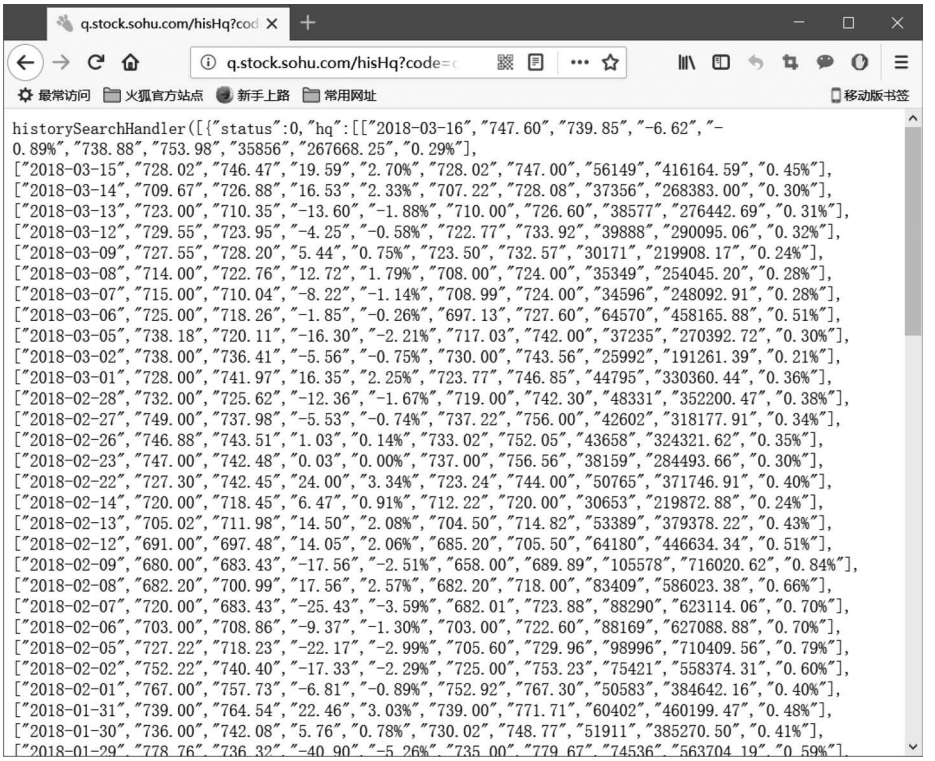



图 5-37 浏览器中展示返回的字符串

笔者将上述返回的 JSON 字符串进行了处理并保存为“贵州茅台股票数据.json”文件，具体处理过程不再赘述，最后获取的文件内容如图 5-38 所示。

解码“贵州茅台股票数据.json”文件具体代码如下：

```
import json
```

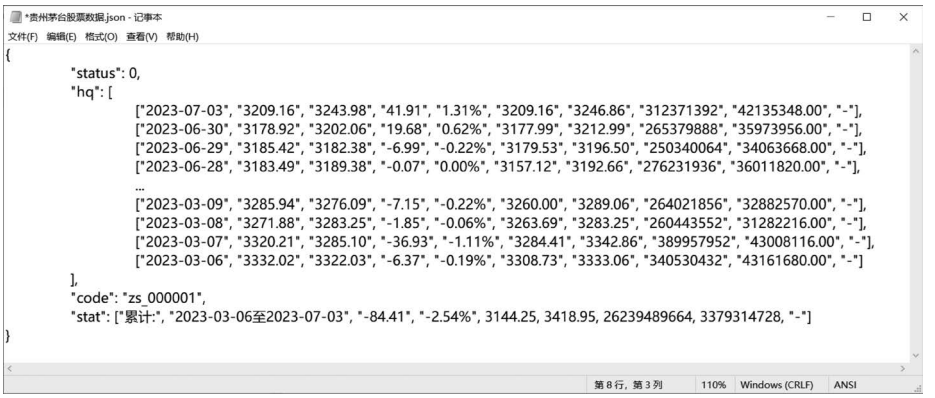



图 5-38　“贵州茅台股票数据.json”文件

```
import pandas as pd

data = []

with open('data/贵州茅台股票数据.json', 'r') as json_file:                    ①
    data_dict = json.load(json_file)                                          ②
    hqlist = data_dict['hq']                                                  ③

    for item in hqlist:
        fields = {}
        fields['Date'] = item[0]                    # 日期                    ④
        fields['Open'] = item[1]                    # 开盘
        fields['Close'] = item[2]                   # 收盘
        fields['Low'] = item[5]                     # 最低
        fields['High'] = item[6]                    # 最高
        fields['Volume'] = item[7]                  # 成交量                  ⑤
        data.append(fields)                                                   ⑥

df = pd.DataFrame(data)                                                       ⑦
print(df)
```

示例运行后，输出结果如下：

```
         Date      Open     Close       Low      High     Volume
0   2023-07-03  3209.16  3243.98  3209.16  3246.86  312371392
1   2023-06-30  3178.92  3202.06  3177.99  3212.99  265379888
2   2023-06-29  3185.42  3182.38  3179.53  3196.50  250340064
3   2023-06-28  3183.49  3189.38  3157.12  3192.66  276231936
4   2023-06-27  3153.31  3189.44  3148.27  3194.41  287604320
..         ...      ...      ...      ...      ...        ...
75  2023-03-10  3255.51  3230.08  3229.50  3262.15  281135424
76  2023-03-09  3285.94  3276.09  3260.00  3289.06  264021856
77  2023-03-08  3271.88  3283.25  3263.69  3283.25  260443552
78  2023-03-07  3320.21  3285.10  3284.41  3342.86  389957952
79  2023-03-06  3332.02  3322.03  3308.73  3333.06  340530432
```

```
[80 rows x 6 columns]
```

解释代码：

- 代码第①行打开名为'贵州茅台股票数据.json'的JSON文件，并将其赋值给json_file变量。使用with语句可以确保在处理完文件后自动关闭文件。
- 代码第②行将JSON文件内容加载为Python字典，并将其赋值给data_dict变量。JSON文件中的数据被转换为Python的字典形式，便于后续的处理。
- 代码第③行从data_dict字典中获取键为'hq'的值，并将其赋值给hqlist变量。假设JSON文件中有一个键为'hq'的列表，该列表包含了一组股票数据。
- 代码第④行for item in hqlist::这是一个for循环，用于遍历hqlist列表中的每个元素，每个元素称为item。
- 代码第⑤行将item列表中的第一个元素(索引为0)赋值给fields字典的键'Date'。假设该元素是日期。
- 代码第⑥行fields['Open'] = item[1]～fields['Volume'] = item[7]：这些行代码将item列表中不同索引位置的元素分别赋值给fields字典的相应键，例如，将第二个元素赋值给'Open'键，第三个元素赋值给'Close'键，以此类推。假设这些元素分别表示开盘价、收盘价、最低价、最高价和成交量。
- 代码第⑦行df = pd.DataFrame(data)：这行代码将data列表转换为Pandas DataFrame对象，并将其赋值给df变量。

## 5.4 本章总结

本章主要介绍了办公自动化中数据存储的两种常用方式——MySQL数据库和JSON格式，以及在Python中对其的操作。

首先，介绍了MySQL数据库管理系统的特点，并学习了如何安装MySQL数据库。我们还学习了如何使用命令行客户端以及图形界面工具管理MySQL数据库。

随后，安装了PyMySQL库，并学习了如何使用Python程序访问MySQL数据库的一般流程。通过一个示例，我们连接MySQL数据库，并读取了苹果股票数据。

接下来，介绍了如何用Pandas从MySQL数据库中读取和写入数据。Pandas提供了便捷的数据库接口，可以大幅简化数据库操作。

此外，还介绍了JSON这种轻量级的数据交换格式。JSON使用JavaScript语法来表示数据对象、数组等。我们学习了JSON的编码与解码，并完成了一个将爬取的贵州茅台股票数据解码为Python字典的示例。

总体而言，本章介绍的MySQL数据库和JSON数据格式在办公自动化中数据存储功能非常关键。熟练掌握它们的操作可以大大提高程序获取和处理数据的效率。