5 数据库编程

数据库编程是指使用编程语言与数据库进行交互，实现数据的存储、查询、更新和删除等操作。在Python中，可以通过各种数据库管理系统（如SQLite、MySQL、PostgreSQL等）的API来操作数据库，实现数据的持久化存储和管理。

5.1 数据库基础与SQL

数据库是组织、存储和管理数据的系统，它允许用户和应用程序以各种方式存取数据。数据库管理系统（DBMS）是用于创建和管理数据库的软件。SQL（结构化查询语言）是一种用于与数据库交互、执行各种数据操作如查询、更新、插入和删除的标准语言。

数据库基础概念

- **数据库（Database）**：一个按照数据结构来组织、存储和管理数据的仓库。
- **数据表（Table）**：数据库中的基本存储单位，以行和列的形式组织数据。
- **字段（Field）**：表中的一列，代表数据的一个属性。
- **记录（Record）**：表中的一行，是一组相关的数据项的集合。
- **主键（Primary Key）**：唯一标识表中每条记录的字段或字段组合。
- **外键（Foreign Key）**：表中的一个字段，它是另一个表的主键，用于关联两个表。

SQL基础

SQL分为几个部分：数据定义语言（DDL）、数据操纵语言（DML）、数据控制语言（DCL）和事务控制语言（TCL）。

- **数据定义语言（DDL）**：用于定义和修改数据库结构的语句，包括`CREATE`、`ALTER`、`DROP`等。
- **数据操纵语言（DML）**：用于对数据库中的数据进行增加、删除、修改和查询的语句，包括`SELECT`、`INSERT`、`UPDATE`、`DELETE`等。
- **数据控制语言（DCL）**：用于定义数据库的安全性、完整性、恢复和并发控制的语句，包括`GRANT`、`REVOKE`等。
- **事务控制语言（TCL）**：用于管理事务的语句，包括`COMMIT`、`ROLLBACK`等。

使用SQL进行基本操作

- **创建表**：

```
CREATE TABLE students (
    id INT PRIMARY KEY,
    name TEXT,
    age INT,
    grade INT
);
```

- **插入数据**：

```
INSERT INTO students (id, name, age, grade) VALUES (1, 'Alice', 20, 3);
```

- **查询数据**：

```
SELECT * FROM students WHERE age > 18;
```

- **更新数据**：

```
UPDATE students SET grade = 4 WHERE id = 1;
```

- **删除数据**：

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

了解这些基础知识后，你可以开始使用Python操作数据库，进行更复杂的数据操作和管理。

在线练习SQL语法的网站

- **SQLZoo** (https://sqlzoo.net/): SQLZoo提供了多种难度的SQL练习题，从基础到高级，涵盖了多种SQL命令和概念，是学习和练习SQL的好地方。
- **LeetCode** (https://leetcode.com/problemset/database/): LeetCode的数据库部分提供了多种难度的数据库相关编程题，适合练习和提高SQL查询和数据结构设计能力。
- **W3Schools SQL Tutorial** (https://www.w3schools.com/sql/): W3Schools提供了一个简单易懂的SQL教程，包括在线练习，可以即时看到SQL语句的执行结果。

通过这些资源，您可以进一步加深对数据库基础和SQL语法的理解，并通过实践提高您的数据库操作技能。

5.2 SQLite数据库操作

SQLite是一个开源的、零配置、自给自足的、在本地运行的SQL数据库引擎。它是世界上最广泛部署的数据库引擎，用于各种大小和类型的应用程序，包括桌面、移动、嵌入式和服务器端应用程序。SQLite的主要特点包括：

- **自给自足**：SQLite不需要一个单独的服务器进程或系统来运行，数据库引擎是直接集成到应用程序中的。
- **无需配置**：不需要安装或管理配置。一个简单的SQLite数据库是一个单一的跨平台的文件。
- **小巧轻便**：SQLite核心库的大小非常小，适合所有类型的设备，从小型嵌入式设备到大型服务器。
- **事务性**：支持完整的ACID事务，确保所有操作都是安全的，即使在系统崩溃或电源故障的情况下。
- **多语言支持**：SQLite提供了多种编程语言的API，包括但不限于C/C++、Python、PHP、Java等。
- **灵活**：支持大多数SQL标准特性，同时也提供了一些独特的功能。

SQLite适用于需要轻量级数据库解决方案的场景，特别是在资源受限的环境中，如移动设备、嵌入式设备和客户端应用程序。由于其简单和易于使用的特性，它也常被用于原型开发和小型应用程序。

Python内置了对SQLite数据库的支持，可以直接使用标准库`sqlite3`来操作SQLite数据库，无需单独安装SQLite。`sqlite3`模块提供了一个与SQLite数据库交互的接口，并且是Python标准库的一部分，因此在安装Python时就已经包含了`sqlite3`模块，可以直接使用。

下面是一个详细的教学内容，涵盖了使用Python进行SQLite数据库操作的完整过程，包括创建数据库、创建表、插入数据、查询数据、更新数据和删除数据。

步骤1：创建数据库和表

首先，我们需要创建一个SQLite数据库文件，并在其中创建一个表。在SQLite中，数据库是一个文件，如果指定的文件不存在，SQLite会自动创建它。

```
import sqlite3

# 连接到SQLite数据库
# 如果文件不存在，会自动在当前目录创建一个数据库文件
conn = sqlite3.connect('example.db')

# 创建一个Cursor对象
c = conn.cursor()

# 创建表
c.execute('''
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    grade INTEGER
)
''')

# 提交事务
conn.commit()

# 关闭连接
conn.close()
```

步骤2：插入数据

接下来，向`students`表中插入一些数据。

```
conn = sqlite3.connect('example.db')
c = conn.cursor()

# 插入数据
c.execute("INSERT INTO students (name, age, grade) VALUES ('Alice', 20, 3)")
c.execute("INSERT INTO students (name, age, grade) VALUES ('Bob', 21, 4)")
c.execute("INSERT INTO students (name, age, grade) VALUES ('Charlie', 22, 3)")

# 提交事务
conn.commit()

# 关闭连接
conn.close()
```

步骤3：查询数据

现在，我们可以从`students`表中查询数据了。

```
conn = sqlite3.connect('example.db')
c = conn.cursor()

# 查询所有学生
c.execute("SELECT * FROM students")
print("所有学生：")
for row in c.fetchall():
    print(row)

# 查询年龄大于20岁的学生
c.execute("SELECT * FROM students WHERE age > 20")
print("\n年龄大于20岁的学生：")
for row in c.fetchall():
    print(row)

# 关闭连接
conn.close()
```

步骤4：更新和删除数据

最后，我们将演示如何更新和删除表中的数据。

```
conn = sqlite3.connect('example.db')
c = conn.cursor()

# 更新数据
c.execute("UPDATE students SET grade = 5 WHERE name = 'Alice'")
conn.commit()

# 删除数据
c.execute("DELETE FROM students WHERE name = 'Charlie'")
conn.commit()

# 再次查询所有学生
c.execute("SELECT * FROM students")
print("\n更新和删除操作后的所有学生：")
for row in c.fetchall():
    print(row)

# 关闭连接
conn.close()
```

运行结果

由于数据库操作涉及到数据的插入、更新和删除，运行结果会根据您执行这些操作的次数和顺序而有所不同。但基本上，您将看到如下类型的输出：

- 所有学生的列表。
- 年龄大于20岁的学生列表。
- 更新和删除操作后的所有学生列表。

这个示例涵盖了使用Python进行SQLite数据库操作的基本流程，包括创建数据库和表、插入、查询、更新和删除数据。通过这些步骤，您可以开始构建自己的数据驱动的Python应用程序。

5.3 实践指导

编写数据库操作的练习题，进行数据存储与查询

练习题1：创建学生信息表

题目描述：
创建一个名为`students`的表，包含以下字段：`id`（主键，自增），`name`（姓名，文本类型），`age`（年龄，整型），`grade`（年级，整型）。

详细编程步骤：

1. 连接到SQLite数据库（如果数据库文件不存在，则会自动创建）。
2. 使用SQL语句创建`students`表。
3. 提交事务，关闭数据库连接。

完整代码：

```
import sqlite3

# 连接到SQLite数据库
conn = sqlite3.connect('school.db')
c = conn.cursor()

# 创建students表
c.execute('''
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    grade INTEGER
)
''')

# 提交事务
conn.commit()

# 关闭连接
conn.close()

print("创建students表成功。")
```

输入输出结果：

```
创建students表成功。
```

练习题2：向学生信息表中插入数据

题目描述：
向`students`表中插入至少3条学生信息。

详细编程步骤：

1. 连接到SQLite数据库。
2. 使用SQL语句向`students`表中插入数据。
3. 提交事务，关闭数据库连接。

完整代码：

```python
import sqlite3

# 连接到SQLite数据库
conn = sqlite3.connect('school.db')
c = conn.cursor()

# 插入数据
students_data = [
    ('Alice', 20, 3),
    ('Bob', 22, 4),
    ('Charlie', 21, 3)
]
c.executemany("INSERT INTO students (name, age, grade) VALUES (?, ?, ?)", students_data)

# 提交事务
conn.commit()

# 关闭连接
conn.close()

print("成功插入数据。")
```

输入输出结果：

```
成功插入数据。
```

练习题3：查询学生信息

题目描述：
查询`students`表中所有年龄大于20岁的学生信息，并按年级降序排列。

详细编程步骤：

1. 连接到SQLite数据库。
2. 使用SQL语句查询年龄大于20岁的学生信息，并按年级降序排列。

3. 打印查询结果。
4. 关闭数据库连接。

完整代码：

```
import sqlite3

# 连接到SQLite数据库
conn = sqlite3.connect('school.db')
c = conn.cursor()

# 查询年龄大于20岁的学生信息，并按年级降序排列
c.execute("SELECT * FROM students WHERE age > 20 ORDER BY grade DESC")

# 打印查询结果
print("年龄大于20岁的学生信息（按年级降序）：")
for row in c.fetchall():
    print(row)

# 关闭连接
conn.close()
```

输入输出结果：

```
年龄大于20岁的学生信息（按年级降序）：
(2, 'Bob', 22, 4)
(3, 'Charlie', 21, 3)
```

综合题4：图书管理系统

题目描述：
设计一个简单的图书管理系统，使用Python和SQLite。系统需要支持以下功能：

1. 创建一个名为`books`的表，包含以下字段：
    - `id`（主键，自增）
    - `title`（书名，文本类型）
    - `author`（作者，文本类型）
    - `year`（出版年份，整型）
2. 向`books`表中插入至少5本书的信息。
3. 查询所有书籍信息，并按出版年份降序排列。
4. 查询指定作者的所有书籍。
5. 更新一本书的出版年份。
6. 删除一本书。

详细编程步骤：

1. **创建数据库表**：
    - 连接到SQLite数据库。
    - 使用SQL语句创建`books`表。
    - 提交事务，关闭数据库连接。
2. **插入书籍信息**：
    - 连接到SQLite数据库。
    - 使用SQL语句向`books`表中插入数据。
    - 提交事务，关闭数据库连接。
3. **查询所有书籍信息**：
    - 连接到SQLite数据库。
    - 使用SQL语句查询所有书籍信息，并按出版年份降序排列。
    - 打印查询结果。
    - 关闭数据库连接。
4. **查询指定作者的书籍**：
    - 连接到SQLite数据库。
    - 使用SQL语句查询指定作者的所有书籍。
    - 打印查询结果。
    - 关闭数据库连接。
5. **更新书籍信息**：
    - 连接到SQLite数据库。
    - 使用SQL语句更新一本书的出版年份。
    - 提交事务，关闭数据库连接。
6. **删除书籍**：
    - 连接到SQLite数据库。
    - 使用SQL语句删除一本书。
    - 提交事务，关闭数据库连接。

完整代码：

```
import sqlite3

# 连接到SQLite数据库
conn = sqlite3.connect('library.db')
c = conn.cursor()

# 创建books表
c.execute('''
CREATE TABLE IF NOT EXISTS books (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
    title TEXT NOT NULL,
    author TEXT NOT NULL,
    year INTEGER
)
''')

# 插入书籍信息
books_data = [
    ('The Great Gatsby', 'F. Scott Fitzgerald', 1925),
    ('To Kill a Mockingbird', 'Harper Lee', 1960),
    ('1984', 'George Orwell', 1949),
    ('Pride and Prejudice', 'Jane Austen', 1813),
    ('The Catcher in the Rye', 'J.D. Salinger', 1951)
]
c.executemany("INSERT INTO books (title, author, year) VALUES (?, ?, ?)", books_data)

# 查询所有书籍信息
c.execute("SELECT * FROM books ORDER BY year DESC")
print("所有书籍信息（按出版年份降序）：")
for row in c.fetchall():
    print(row)

# 查询指定作者的书籍
author_name = 'George Orwell'
c.execute("SELECT * FROM books WHERE author = ?", (author_name,))
print(f"\n{author_name}的书籍：")
for row in c.fetchall():
    print(row)

# 更新书籍信息
c.execute("UPDATE books SET year = 1950 WHERE title = '1984'")
conn.commit()

# 删除书籍
c.execute("DELETE FROM books WHERE title = 'The Catcher in the Rye'")
conn.commit()

# 关闭连接
conn.close()
```

输入输出结果：

```
所有书籍信息（按出版年份降序）：
(2, 'To Kill a Mockingbird', 'Harper Lee', 1960)
(5, 'The Catcher in the Rye', 'J.D. Salinger', 1951)
(3, '1984', 'George Orwell', 1949)
(1, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925)
(4, 'Pride and Prejudice', 'Jane Austen', 1813)

George Orwell的书籍：
(3, '1984', 'George Orwell', 1949)
```

这个综合题涵盖了Python操作SQLite数据库的基本操作，包括创建表、插入数据、查询数据、更新数据和删除数据，具有一定的实际意义。