Python Database Management Notes for ProgSD Exam

Contents

Chapter 1 DB Management

1.1 Introduction to Relational Databases

- A **database** is an organized collection of structured information stored electronically.
- Example: A database may store details of students and lecturers.
- **Relational databases**:
  - Organize data into **tables** (rows and columns).
  - Efficiently store structured data using relationships between tables.
  - Examples: One-to-many relationships between students and classes.
- **Primary keys**:
  - Uniquely identify each record in a table.
  - Examples: `ID` in the students table or `class` in the class table.

1.2 Important Imports

- To work with relational databases in Python, the following libraries and modules are commonly used:
  - `sqlite3`: Built-in library to interact with SQLite databases.
  - `pymysql`: For connecting to MySQL databases.
  - `psycopg2`: For working with PostgreSQL databases.
  - `sqlalchemy`: High-level SQL toolkit and ORM (Object Relational Mapper).
- Example: Importing `sqlite3` to work with SQLite:

```
import sqlite3 # Built-in library for SQLite databases
```

- Example: Importing `sqlalchemy` for advanced database operations:

```
from sqlalchemy import create_engine, MetaData, Table,
    Column, Integer, String
```

- For external libraries like `pymysql` or `psycopg2`, install them using `pip`:

```
pip install pymysql
pip install psycopg2
```

- Ensure to install any necessary drivers or dependencies based on the database system you are using.

1.3 Data Types in Relational Databases

- Each field in a relational database has a data type, defining the kind of data it can store.
- Commonly used data types:
  - **Integer**:
    - Stores whole numbers.
    - Example: `age INTEGER`

```
CREATE TABLE employees (id INTEGER, age INTEGER);
INSERT INTO employees (id, age) VALUES (1, 25);
```

  - **Real**:
    - Stores decimal values.
    - Example: `salary REAL`

```
CREATE TABLE payments (id INTEGER, salary REAL);
INSERT INTO payments (id, salary) VALUES (1, 2500.50);
```

  - **Text**:
    - Stores string data.
    - Example: `name TEXT`

```
CREATE TABLE students (id INTEGER, name TEXT);
INSERT INTO students (id, name) VALUES (1, 'Alice');
```

  - **Blob**:
    - Stores binary data like images or files.
    - Example: `photo BLOB`

```
CREATE TABLE files (id INTEGER, photo BLOB);
```

  - **Date and Time**:
    - Stores dates, times, or timestamps.
    - Examples: `DATE`, `TIME`, `DATETIME`

```
CREATE TABLE events (id INTEGER, event_date DATE);
INSERT INTO events (id, event_date) VALUES (1, '2024-01-01');
```

  - **Boolean**:
    * Stores `TRUE` or `FALSE`.
    * Example: `is_active BOOLEAN`

```
CREATE TABLE users (id INTEGER, is_active BOOLEAN);
INSERT INTO users (id, is_active) VALUES (1, TRUE);
```

  - **Varchar**:
    * Variable-length string with a max length.
    * Example: `name VARCHAR(50)`

```
CREATE TABLE products (id INTEGER, name VARCHAR(50));
INSERT INTO products (id, name) VALUES (1, 'Laptop');
```

- Additional constraints:
  - `NOT NULL`: Ensures the field cannot be empty.
  - `PRIMARY KEY`: Uniquely identifies a record.
- Choosing the right data type ensures efficient storage, data integrity, and performance.

1.4 SQLite Overview

- **SQLite**: Open-source database system based on SQL.
- Lightweight and self-contained.
- Commonly used for small to medium applications.
- Installation:
  - Download binaries from https://sqlite.org.
  - Use **DB Browser for SQLite** for GUI-based database management.

1.5 SQLite in Python

- Import the `sqlite3` module:

```
import sqlite3
```

- Create and connect to a database:

```
with sqlite3.connect("company.db") as db:
    cursor = db.cursor()
```

- Create a table:

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS students(
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        class TEXT NOT NULL,
        grade INTEGER)
    """)
```

1.6 Querying a Database

- **Select all records**:

```
cursor.execute("SELECT * FROM students")
for row in cursor.fetchall():
    print(row)
```

Output:

```
(1899877D, 'Mary', 'Python', 67)
(2223998M, 'John', 'Maths', 34)
```

- **Select records with conditions**:

```
cursor.execute("SELECT * FROM students WHERE grade > 50")
```

- **Select specific fields**:

```
cursor.execute("SELECT id, name FROM students")
```

1.7 Joining Tables

- Join data from two tables:

```
cursor.execute("""
    SELECT students.name, class.lecturer
    FROM students
    JOIN class ON students.class = class.class
""")
```

- Example: Get student names along with their lecturer's name.

1.8 Insert, Update, and Delete

- **Insert data**:

```
cursor.execute("""
    INSERT INTO students (id, name, class, grade)
    VALUES (2348990M, 'Anne', 'Python', 70)
""")
db.commit()
```

- **Update data**:

```
cursor.execute("""
    UPDATE students
    SET grade = 75
    WHERE name = 'Anne'
""")
db.commit()
```

- **Delete data**:

```
cursor.execute("""
    DELETE FROM students WHERE name = 'Anne'
""")
db.commit()
```

1.9 Practical Example: PhoneBook Database

- Create a PhoneBook database:

```
with sqlite3.connect("PhoneBook.db") as db:
    cursor = db.cursor()

    # Create the Names table
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Names(
            id INTEGER PRIMARY KEY,
            firstname TEXT,
            surname TEXT,
            phonenumber TEXT
        )
    """)

    # Insert data
    cursor.execute("""
        INSERT INTO Names (id, firstname, surname,
            phonenumber)
        VALUES (1, 'Simon', 'Pierre', '0141647 1367')
    """)
    db.commit()

    # Fetch and print records
    cursor.execute("SELECT * FROM Names")
    for row in cursor.fetchall():
        print(row)
```

Output:

```
(1, 'Simon', 'Pierre', '0141647 1367')
```