**Module 16. Database Programming**

**Learning Objectives**

* Understand the basics of database management systems (DBMS)
* Demonstrate proficiency in using Python libraries such as sqlite3 for interacting with databases.
* Learn how to establish connections to databases, create tables, and perform CRUD (Create, Read, Update, Delete) operations using Python.
* Gain familiarity with various types of databases, including relational databases

# **Database Management Systems (DBMS)**

Utilizing traditional file storage imposes inherent limitations. While it proves adequate for applications handling minimal data volumes, it becomes impractical for those requiring substantial storage capacity. As data scales up, even simple operations turn laborious and less efficient, highlighting the inadequacy of this approach.

A Database Management System (DBMS) is specialized software crafted to handle extensive datasets efficiently and systematically. Within this system, data is housed and managed. Instead of directly manipulating data, applications developed in languages like Python interact with the DBMS. The DBMS executes commands received from the application and subsequently transmits the outcomes back to it.

A diagram of data processing

Description automatically generated

# **Structured Query Language (SQL)**

SQL, which stands for Structured Query Language, serves as a standardized language primarily employed for interacting with Database Management Systems (DBMS). Unlike general-purpose programming languages, SQL is specifically tailored for managing databases. It comprises various keywords utilized in constructing statements. These SQL statements, conveyed as strings through API method calls from the application to the DBMS, function as directives for the DBMS to execute operations on its stored data.

# **SQLite**

SQLite is a C library that offers a compact, disk-based database solution, eliminating the need for a separate server process. It facilitates database access through a nonstandard variant of the SQL query language. Numerous applications leverage SQLite for internal data storage. Additionally, developers can utilize SQLite to prototype applications before transitioning the code to more expansive databases like PostgreSQL or Oracle.

You can import the sqlite3 module in Python using the following statement:

import sqlite3

# **Database (Table, Row and Column)**

A Database Management System (DBMS) is responsible for storing data in a structured manner within a database. A database consists of one or more tables, each of which holds a collection of related data. These tables organize data into rows and columns. Each row represents a complete set of information about a single item, while columns contain individual pieces of data pertaining to the items in the table.

A screen shot of a computer screen

Description automatically generated

In SQLite, every data value has a specific storage category:

* NULL: Represents missing or unknown information
* INTEGER: Stores whole numbers, using a variable amount of space (0 to 8 bytes) based on the number's size
* REAL: Stores floating-point numbers (numbers with decimals) in a standardized 8-byte format
* TEXT: Holds text strings, encoded using the database's character set (typically UTF-8)
* BLOB (Binary Large Object): Stores raw binary data exactly as it was provided, without any interpretation

1. Primary Keys

A primary key is a column or set of columns that uniquely identifies each row in a table. The primary key constraint ensures that the values in this column (or columns) are unique and not null. It's used to uniquely identify each record in the table, making it easy to retrieve, update, or delete specific rows. For example, the MGA ID column serves as the primary key because each student has a unique ID number, allowing you to identify and work with individual student records efficiently.

1. Identity Column

An identity column, also known as an auto-increment column, is a special type of column in a relational database table where unique integer values are automatically generated by the database management system (DBMS). These values are usually incremented by a set amount (e.g., by 1) each time a new row is inserted into the table.

Identity columns are commonly used to create primary keys when the table doesn't have a suitable candidate column to serve as a primary key. In such cases, the identity column can be designated as the primary key, ensuring each row in the table has a unique identifier.

This feature is particularly useful when dealing with tables that lack a natural key (a column or combination of columns that uniquely identifies each row), or when simplicity and consistency in key generation are desired.

1. Null Values

In relational databases, a null value represents a missing or unknown value in a column. While it's acceptable for some columns to contain null values, primary key columns cannot have null values. This is because the primary key uniquely identifies each row in the table, and having a null value would mean that there could be multiple rows with the same undefined identifier, violating the uniqueness constraint of the primary key.

# **Connect to Database**

Using an SQLite database follows a standard procedure as below:

* Establish a connection with the database.
* Retrieve a cursor for the database.
* Execute operations on the database.
* Apply changes to the database.
* Terminate the connection to the database.

Establish a connection with the database

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    # Insert your database operations here.

    # Example:

    # cur.execute("CREATE TABLE IF NOT EXISTS contacts (id INTEGER PRIMARY

KEY, name TEXT, email TEXT, GPA REAL)")

    conn.commit()

    conn.close()

# Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

    main()

To perform an SQL operation on an SQLite database:

* Formulate a string containing the SQL command
* Invoke the execute method of the Cursor object, passing the SQL string as an argument

For example,

cur.execute("CREATE TABLE IF NOT EXISTS contacts (id INTEGER PRIMARY

KEY, name TEXT, email TEXT, GPA REAL)")

# **SQL Operations**

1. Create a table

To create a table in an SQLite database, utilize the SQL command CREATE TABLE.

CREATE TABLE TableName (

ColumnName1 DataType1,

ColumnName2 DataType2,

...

)

CREATE TABLE Students (Name TEXT, Email TEXT, GPA REAL)

To specify a column as the primary key, apply the PRIMARY KEY constraint following the column's data type.

CREATE TABLE Students (ID INTEGER PRIMARY KEY, Name TEXT, Email TEXT, GPA REAL)

The NOT NULL constraint ensures that a column must contain a value and cannot be null.

CREATE TABLE Students (ID INTEGER PRIMARY KEY NOT NULL, Name TEXT, Email TEXT, GPA REAL)

For example,

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    sql\_command = "CREATE TABLE Students (ID INTEGER PRIMARY KEY NOT

NULL, Name TEXT, Email TEXT, GPA REAL)"

    cur.execute(sql\_command)

    conn.commit()

    conn.close()

# Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

    main()

1. Delete a table

DROP TABLE Students

DROP TABLE IF EXISTS Students

1. Add data to a table

To insert data into a table, utilize the INSERT statement.

INSERT INTO TableName (ColumnName1, ColumnName2, ...)

VALUES (Value1, Value2, etc...)

INSERT INTO Students (ID, Name, Email, GPA)

VALUES (1, "Tony Stark", "Tony.Stark@mga.edu", 3.88)

For example,

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('''INSERT INTO Students (Name, Email, GPA)

               VALUES ("Tony Stark", "Tony.Stark@mga.edu", 3.88)''')

    cur.execute('''INSERT INTO Students (Name, Email, GPA)

               VALUES ("Steve Rogers", "Steve.Rogers@mga.edu", 3.95)''')

    cur.execute('''INSERT INTO Students (Name, Email, GPA)

               VALUES ("Black Widow", "Black.Widow@mga.edu", 3.52)''')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

You can incorporate the value of a variable into an SQL statement by employing question mark placeholders.

Name = "Black Widow"

Email = "Black.Widow@mga.edu"

cur.execute('''INSERT INTO Inventory (Name, Email, GPA)

VALUES (?, ?, ?)''', (Name, Email, GPA))

1. Search data with SELECT statement

The SELECT statement is employed to fetch specific rows from a table.

SELECT Columns FROM Table

There are three ways to retrieve data from a table.

* Retrieves the Name column from each row in the Students table.
* Multiple column names are separated by commas. This query retrieves the Name and Email columns from each row in the Students table.
* The asterisk (\*) symbol can be employed to fetch all columns in the table.

After executing SELECT statement, fetch the results from a table using fetchall method or fetchone method.

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('SELECT Name FROM Students')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:30}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

(Output)

Tony Stark

Steve Rogers

Black Widow

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('SELECT Name, Email FROM Students')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:30} {row[1]:30}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

(Output)

Tony Stark Tony.Stark@mga.edu

Steve Rogers Steve.Rogers@mga.edu

Black Widow Black.Widow@mga.edu

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('SELECT \* FROM Students')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:30} {row[1]:30} {row[2]:30} {row[3]:30}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

(Output)

1 Tony Stark Tony.Stark@mga.edu 3.88

2 Steve Rogers Steve.Rogers@mga.edu 3.95

3 Black Widow Black.Widow@mga.edu 3.52

1. SELECT statement with WHERE clause

The WHERE clause, when utilized with the SELECT statement, enables the specification of search criteria. Consequently, only the rows that satisfy the search criteria will be included in the result set.

SELECT Columns FROM Table WHERE Criteria

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('SELECT \* FROM Students WHERE ID <= 2')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:30} {row[1]:30} {row[2]:30} {row[3]:30}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

The first segment of the statement, SELECT \* FROM Students, indicates the retrieval of all columns.

The WHERE clause dictates that only rows where the ID column is less than or equal to 2 should be included.

(Output)

1 Tony Stark Tony.Stark@mga.edu 3.88

2 Steve Rogers Steve.Rogers@mga.edu 3.95

1. SELECT statement with WHERE clause using relational operators and logical operators

Additionally, within the WHERE clause, relational operators such as >, <, >=, <=, ==, =, !=, and <> are employed, alongside logical operators like AND, OR, and NOT.

1. SELECT statement with WHERE clause using LIKE operator

The LIKE operator facilitates searching for substrings. The % symbol functions as a wildcard for multiple characters. The following statement retrieves all rows where the Email column contains the string "Tony" preceded by any characters and followed by any characters.

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('SELECT \* FROM Students WHERE Email LIKE "%Steve%"')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:30} {row[1]:30} {row[2]:30} {row[3]:30}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

(Output)

2 Steve Rogers Steve.Rogers@mga.edu 3.95

1. SELECT statement with ORDER BY clause

This statement will fetch all rows from the Students table, sorted by the Email column. The sorting will be in ascending order based on the Email column.

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('SELECT \* FROM Students ORDER BY Email')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:30} {row[1]:30} {row[2]:30} {row[3]:30}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

(Output)

3 Black Widow Black.Widow@mga.edu 3.52

2 Steve Rogers Steve.Rogers@mga.edu 3.95

1 Tony Stark Tony.Stark@mga.edu 3.88

1. Aggregate Functions

* The AVG function computes the average value within a specified column.
* The SUM function calculates the total sum of values within a column.
* The MIN and MAX functions ascertain the minimum and maximum values present in a column.
* The COUNT function is utilized to determine the total number of rows in a table.

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    ### AVG function

    cur.execute('SELECT AVG(GPA) FROM Students')

    results = cur.fetchall()

    for row in results:

        print(f'Average GPA {row[0]:10}')

    ### SUM function

    cur.execute('SELECT SUM(GPA) FROM Students')

    results = cur.fetchall()

    for row in results:

        print(f'SUM GPA {row[0]:10}')

    ### MIN function

    cur.execute('SELECT MIN(GPA) FROM Students')

    results = cur.fetchall()

    for row in results:

        print(f'MIN GPA {row[0]:10}')

    ### MAX function

    cur.execute('SELECT MAX(GPA) FROM Students')

    results = cur.fetchall()

    for row in results:

        print(f'MAX GPA {row[0]:10}')

    ### COUNT function

    cur.execute('SELECT COUNT(\*) FROM Students')

    results = cur.fetchall()

    for row in results:

        print(f'COUNT {row[0]:10}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

(Output)

Average GPA 3.783333333333333

SUM GPA 11.35

MIN GPA 3.52

MAX GPA 3.95

COUNT 3

1. Update rows

The UPDATE statement modifies the contents of an existing row within a table.

UPDATE Table SET Column = Value WHERE Criteria

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('UPDATE Students SET GPA = 3.75 WHERE Name == "Black Widow"')

    cur.execute('SELECT \* FROM Students')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:30} {row[1]:30} {row[2]:30} {row[3]:30}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

This example updated all rows in the Students table where the Name is "Black Widow". For each of these rows, it set the GPA column from 3.52 to 3.75.

(Output)

1 Tony Stark Tony.Stark@mga.edu 3.88

2 Steve Rogers Steve.Rogers@mga.edu 3.95

3 Black Widow Black.Widow@mga.edu 3.75

**(Be careful!)**

Always include the WHERE clause unless you intend to update every row in the table! For instance, this SQL statement sets the GPA column to 0 for every row in the Students table:

UPDATE Students SET GPA = 0

1. Delete rows

The DELETE statement removes one or more rows from a table.

DELETE FROM Table WHERE Criteria

import sqlite3

def main():

    conn = sqlite3.connect('contacts.db')

    cur = conn.cursor()

    cur.execute('DELETE FROM Students WHERE Name == "Steve Rogers"')

    cur.execute('SELECT \* FROM Students')

    results = cur.fetchall()

    for row in results:

         print(f'{row[0]:20} {row[1]:20} {row[2]:20} {row[3]:20}')

    conn.commit()

    conn.close()

    # Execute the main function.

if \_\_name\_\_ == '\_\_main\_\_':

     main()

This statement removed all rows from the Products table where the Name is "Steve Rogers".

(Output)

1 Tony Stark Tony.Stark@mga.edu 3.88

3 Black Widow Black.Widow@mga.edu 3.75

**(Be careful!)**

Always include the WHERE clause unless you intend to delete every row in the table! For instance, this SQL statement deletes every row in the Students table:

DELETE FROM Students

1. Composite Keys

Combining two or more columns to form a composite key is possible. This approach is sometimes necessary to ensure unique values in the primary key.

CREATE TABLE Students (MGA\_ID INTEGER NOT NULL,

Name TEXT NOT NULL,

Email TEXT,

PRIMARY KEY(MGA\_ID, Name))

1. Handing Exceptions

import sqlite3

conn = None

try:

    DatabaseName = 'example.db'

    conn = sqlite3.connect(DatabaseName)

    cur = conn.cursor()

    # Perform database operations here.

except sqlite3.Error as e:

    # Respond to the SQLite database exception.

    print("SQLite Error:", e)

except Exception as e:

    # Respond to the general exception.

    print("Error:", e)

finally:

    if conn is not None:

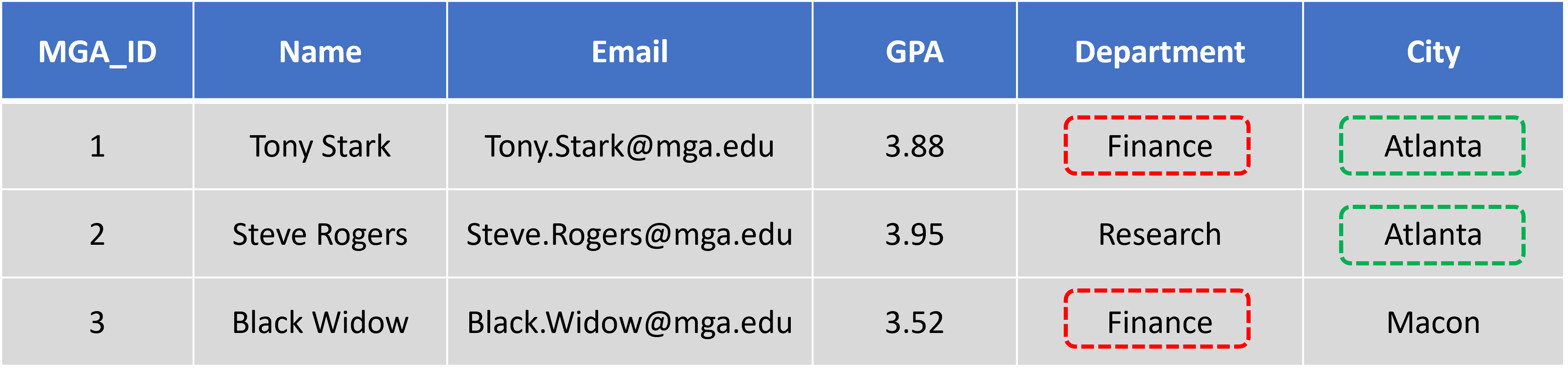
        conn.close()

This code attempts to establish a connection to the SQLite database specified by DatabaseName. Inside the try block, you can perform various database operations. If there's a SQLite-specific error, it's caught in the first except block, and if there's a general error, it's caught in the second except block. Finally, the connection is closed in the finally block to ensure it's properly closed regardless of whether an exception occurred or not.

# **Relational Database**

In relational databases, columns in one table often relate to columns in other tables through associations. Relational database design aims to minimize redundant data, which can cause several issues:

* Unnecessary consumption of storage space
* Inconsistencies and conflicts in stored data due to duplication
* Compromised data integrity



In this table, "Finance" and "Atlanta" are replicated.

In a relational design, data would be divided into multiple tables such as Department, Student, and Location table.

A screen shot of a cellphone

Description automatically generated

A foreign key (FK) is a column within one table that refers to the primary key in another table.

The following SQL statements are used to create the Department and Location tables:

CREATE TABLE Department(Department\_ID INTEGER PRIMARY KEY NOT NULL, Department\_Name TEXT)

CREATE TABLE Location(Location\_ID INTEGER PRIMARY KEY NOT NULL, City TEXT)

When creating the Student table, we utilize the FOREIGN KEY table constraint to designate the foreign keys:

CREATE TABLE Student(MGA\_ID INTEGER PRIMARY KEY NOT NULL,

Name TEXT,

Email TEXT,  
 GPA REAL,

Department\_ID INTEGER,

Location\_ID INTEGER,

FOREIGN KEY(Department\_ID) REFERENCES

Department(Department\_ID),

FOREIGN KEY(Location\_ID) REFERENCES

Location(Location\_ID))

**Summary**

1. A Database Management System (DBMS) is specialized software crafted to handle extensive datasets efficiently and systematically.
2. SQL, which stands for Structured Query Language, serves as a standardized language primarily employed for interacting with Database Management Systems (DBMS).
3. SQLite is a C library that offers a compact, disk-based database solution, eliminating the need for a separate server process.
4. A database consists of one or more tables, each of which holds a collection of related data. These tables organize data into rows and columns.
5. A primary key is a column or set of columns that uniquely identifies each row in a table. The primary key constraint ensures that the values in this column (or columns) are unique and not null.
6. SQLite database follows a standard procedure as follows:
   * Establish a connection with the database.
   * Retrieve a cursor for the database.
   * Execute operations on the database.
   * Apply changes to the database.
   * Terminate the connection to the database.
7. To create a table in an SQLite database, utilize the SQL command CREATE TABLE.
8. To delete a table in an SQLite database, utilize the SQL command DROP TABLE.
9. The SELECT statement is employed to fetch specific rows from a table.
10. The WHERE clause, when utilized with the SELECT statement, enables the specification of search criteria.
11. within the WHERE clause, relational operators such as >, <, >=, <=, ==, =, !=, and <> are employed, alongside logical operators like AND, OR, and NOT.
12. The LIKE operator facilitates searching for substrings. The % symbol functions as a wildcard for multiple characters.
13. This SELECT statement with ORDER BY clause will fetch all rows from the table. The sorting will be in ascending order based on the column.
14. The AVG function computes the average value within a specified column.
15. The SUM function calculates the total sum of values within a column.
16. The MIN and MAX functions ascertain the minimum and maximum values present in a column.
17. The COUNT function is utilized to determine the total number of rows in a table.
18. The UPDATE statement modifies the contents of an existing row within a table.
19. The DELETE statement removes one or more rows from a table.
20. Combining two or more columns to form a composite key is possible.
21. In relational databases, columns in one table often relate to columns in other tables through associations. Relational database design aims to minimize redundant data.
22. A foreign key (FK) is a column within one table that refers to the primary key in another table.

**Programming Exercises**

**Exercise 1: Creating a Database and Table**

Write a Python script to create a SQLite database named "mydatabase.db" and a table named "employees" with columns for "id" (integer), "name" (text), and "salary" (real).

**Exercise 2: Inserting Data**

Write a Python script to insert multiple rows of data into the "employees" table. Prompt the user to input employee details such as name and salary, and insert them into the table.

**Exercise 3: Querying Data**

Write Python functions to perform the following queries on the "employees" table:

* Retrieve all employees' details.
* Retrieve the details of an employee by their ID.
* Retrieve employees with a salary greater than a specified amount.

**Exercise 4: Updating Data**

Write a Python script to update the salary of an employee specified by their ID. Prompt the user to input the employee ID and the new salary, and update the corresponding record in the database.

**Exercise 5: Deleting Data**

Write a Python script to delete an employee record from the "employees" table based on their ID. Prompt the user to input the employee ID to be deleted.