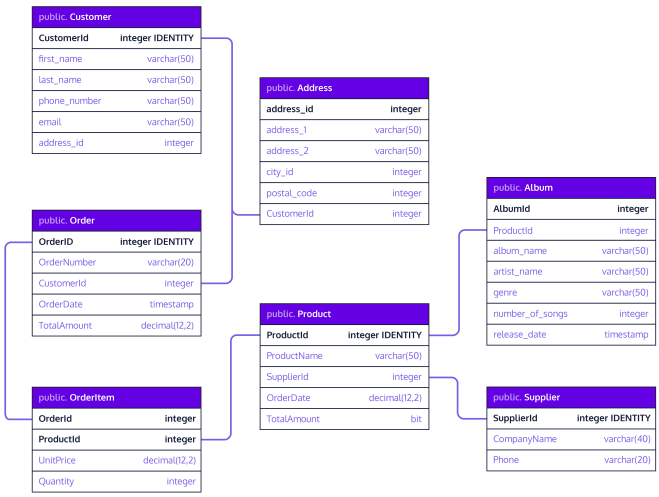
Database Operations

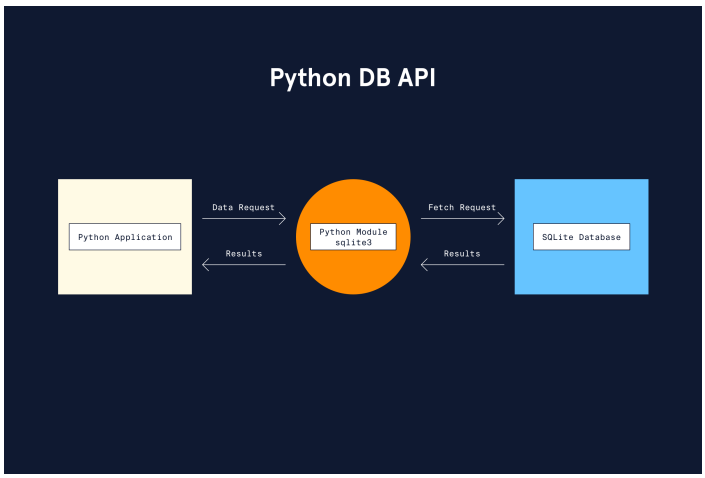
SQL and Relational Databases

Relational databases are the primary means of storage for structured data. They organize data into tables that each contain data related to one another.
We commonly use *SQL*, which stands for Structured Query Language, to query and play around with relational databases. This programming language is designed to manage data stored in relational databases.
We can visualize a relational database in the image of tables below.



Python and SQLite Working Together

Thanks to Python's Database-API (DB-API 2.0), we can connect Python to RDBMS (Relational Database Management Systems) like SQLite. To access a SQLite database, we must import the `sqlite3` into the Python environment.
The following image demonstrates how Python and SQLite function together.



Connecting to the SQLite Database

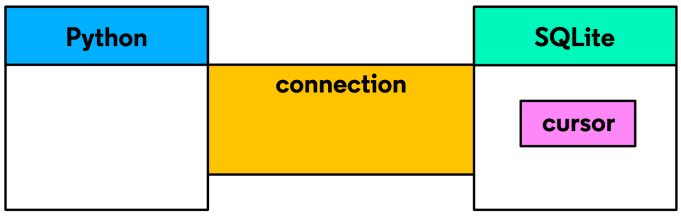
In order to edit a new or pre-existing SQLite database from a Python environment, one must connect with the database using `sqlite3.connect()` .

```
# Create connection to database
connection = sqlite3.connect("exa
```

The connection object is like a cable that connects our python environment to our SQLite database.

Creating a Cursor Object

In a database, a cursor allows us to call statements and return data.
To create a cursor object from a Python environment, one must attach the connection object (in this case `connection`) to the `.cursor()` method.




```
# Create cursor object
cursor = connection.cursor()
```

Executing SQL Statements in Python

Once we have connected to the SQLite database, we can use our cursor object and the `.execute()` method to execute a SQL statement.
Using the `.execute()` method with the `CREATE` clause will create a table within our SQLite database.

```
# Create cursor object
curs = connection.cursor()
# Create table named members
curs.execute('''CREATE TABLE memb
                    id INTEGER,
                    name TEXT)''')
```

We can also use the INSERT clause to insert data into a pre-existing table.

```
# Insert a row of data in the mem
curs.execute('''INSERT INTO membe
```

Fetching SQLite Data in Python

We can pull data from a SQLite data table into our Python environment by using the fetch methods: `.fetchone()`, `.fetchmany()`, and `.fetchall()`.
`.fetchone()` example:

```
# Return first row in students
cursor.execute("SELECT * FROM stu
```

```
# Output
(101, 'Alex', 32, '2022-05-16', '
```

.fetchmany() example:

```
# Return first three rows in stud
cursor.execute("SELECT * FROM stu
# Output
[(101, 'Alex', 32, '2022-05-16',
(102, 'Joe', 32, '2022-05-16', 'P
(103, 'Stacy', 10, '2022-05-16',
```

.fetchall() example:

```
# Return all rows in students
cursor.execute("SELECT * FROM stu
```

For Loop with SQLite Statement

We can use a for loop and a SQL statement to retrieve SQLite data.
The following code will iterate through each row in the students table and print each row where the Grade field is 'Pass' .

```
for row in cursor.execute(```SELE
    print(row)
```

You can also use a for loop to iterate through a table field and calculate a measurement.

```
# save all rows from a field, the
major_codes = cursor.execute("SEL


# Find the average of the tuple l
sum = 0
for num in major_codes:
    for i in num:
        sum = sum + i
```

```
average = sum / len(major_codes)

# Show average
print(average)
```

Committing and Closing SQLite

After making changes to the SQLite database, we must commit the changes using the `.commit()` method. Committing the changes ensures that others can view these changes in the database.

```
# commit changes to database
connection.commit()
```

When we've finished editing the SQLite database and have committed the changes, we may use the `.close()` method to close the database connection.

```
# close connection
connection.close()
```

Inserting many rows with SQLite

To insert multiple rows/records of data into a SQLite database via Python, use the `.executemany()` method.
In the example below, the object `new_students` containing a list of rows is inserted into the already existing `students` data table. Remember, these rows follow the same table schema as the `students` table.

```
# Insert multiple values into tab
new_students = [(102, 'Joe', 32,
           (103, 'Stacy', 10, '20
           (104, 'Angela', 21, '2
           (105, 'Mark', 21, '202
           (106, 'Nathan', 21, '2
           ]
# Insert values into the students
cursor.executemany('''INSERT INTO
```

In the last line of code, there is a list of question marks that act as field placeholders. The five question marks represent each of the five fields in the database we are inserting values into.