SQLite

SQLite is a self-contained, file-based SQL database.

SQLite is not directly comparable to client/server SQL database engines such as Oracle, PostgreSQL or MySQL, it is intended to provide local data storage for individual applications and devices.

SQLite comes bundled with Python and can be used in any Python applications without having to install any additional software.

In [ ]:
```
import sqlite3
from sqlite3 import Error

print(sqlite3.version)
```

```
2.6.0
```

Creating a connection to a SQLite database

The sqlite3.**connect()** function returns a Connection object that can be used to interact with the SQLite database (that ultimately resides in a file). The database (file) will be created automatically if it does not exists.

In [ ]:
```
db_file= "./journal_articles.db"

connection = None
try:
    connection = sqlite3.connect(db_file)
except Error as e:
    print("Cannot create the database connection.")
    print(e)
```

Creating tables

In order to execute SQL statements like CREATE TABLE, you need to first create a Cursor object, by calling the **cursor()** method of the Connection object, and then pass the SQL statement to the **execute()** method of the Cursor object.

In [ ]:
```
sql_statement = """
CREATE TABLE IF NOT EXISTS journal (
    journal_id integer PRIMARY KEY,
    journal_name text NOT NULL,
    publisher text DEFAULT 'Springer Nature' NOT NULL
    );
"""

if connection is not None:
    try:
        cursor = connection.cursor()
        # Create journal table
        cursor.execute(sql_statement)
```

```
    except Error as e:
        print(e)
else:
    print("A valid connection object is required.")
```

Inserting data

In SQLite, a column with type **INTEGER PRIMARY KEY is an alias** for the **ROWID**, a permanent unique identifier for a row. On an INSERT, if the INTEGER PRIMARY KEY column is not explicitly given a value, then it will be **filled automatically with an unused integer**, usually one more than the largest ROWID currently in use.

To use a variable in a SQL statement, write a question mark (?) **as a placeholder for each argument** in the statement and substitute the actual values by providing them as a Python sequence (e.g. a Python tuple) to the cursor's execute() method.

In [ ]:
```
def insert_new_journal(conn, journal):
    """
    Insert a new journal into the journal table
    Parameters
    ----------
    conn: Connection object
    journal: Values to bind to placeholders in sql statement (a tuple) .
    Returns
    -------
    the row id of the inserted row.
    """
    cursor = conn.cursor()
    # For the question marks style, parameters must be a sequence e.g. a tuple
    cursor.execute("INSERT INTO journal(journal_name, publisher) VALUES(?,?)", j
    conn.commit()
    return cursor.lastrowid
```

In [ ]:
```
if connection is not None:
    #  journal: Values to bind to placeholders in sql statement (a tuple).
    journal = ('Neurocomputing', 'Elsevier');
    journal_id = insert_new_journal(connection, journal)
    print(f"Data successfully inserted, journal_id: {journal_id}")

    journal = ('IEEE Access', 'IEEE');
    journal_id = insert_new_journal(connection, journal)
    print(f"Data successfully inserted, journal_id: {journal_id}")
else:
    print("A valid connection object is required.")
```

```
Data successfully inserted, journal_id: 1
Data successfully inserted, journal_id: 2
```

Querying Data

The **fetchall()** method of the Cursor object fetches all the rows of a query result. It returns all the rows **as a list of Python tuples**. An empty list is returned if there is no record to fetch.

In [ ]:
```
def select_journals_by_publisher(conn, publisher):
```

```
    """
    Query journals by publisher
    Parameters
    ----------
    conn: Connection object
    publisher: Value to bind to placeholder in sql statement.
    """
    cursor = conn.cursor()

    # For the question marks style, parameters must be a sequence e.g. a tuple
    cursor.execute("SELECT * FROM journal WHERE publisher=?", (publisher,))

    rows = cursor.fetchall()

    print("Number of rows returned: ", len(rows))
    for row in rows:
        print(row)
```

Never use string operations to assemble queries, as they are vulnerable to SQL injection attacks

In [ ]:
```
def select_journals_by_publisher_wrong_version(conn, publisher):
    cursor = conn.cursor()
    sql_statement = "SELECT * FROM journal WHERE publisher='%s'" % publisher
    print(sql_statement)
    cursor.execute(sql_statement)

    rows = cursor.fetchall()

    print("Number of rows returned: ", len(rows))
    for row in rows:
        print(row)
```

In [ ]:
```
if connection is not None:
    publisher = "Elsevier"

    print("Query the database for publisher %s:" %publisher)
    select_journals_by_publisher(connection, publisher)

    print("\nQuery the database for publisher %s (wrong version):" %publisher)
    select_journals_by_publisher_wrong_version(connection, publisher)
else:
    print("A valid connection object is required.")
```

```
Query the database for publisher Elsevier:
Number of rows returned:  1
(1, 'Neurocomputing', 'Elsevier')

Query the database for publisher Elsevier (wrong version):
SELECT * FROM journal WHERE publisher='Elsevier'
Number of rows returned:  1
(1, 'Neurocomputing', 'Elsevier')
```

In [ ]:
```
if connection is not None:
    publisher = "' OR TRUE; --"

    print("Query the database for publisher %s:" %publisher)
    select_journals_by_publisher(connection, publisher)
```

```
    print("\nQuery the database for publisher %s (wrong version):" %publisher)
    select_journals_by_publisher_wrong_version(connection, publisher)
else:
    print("A valid connection object is required.")
```

```
Query the database for publisher ' OR TRUE; --:
Number of rows returned:  0

Query the database for publisher ' OR TRUE; -- (wrong version):
SELECT * FROM journal WHERE publisher='' OR TRUE; --'
Number of rows returned:  2
(1, 'Neurocomputing', 'Elsevier')
(2, 'IEEE Access', 'IEEE')
```

Close the database connection

In [ ]:
```
if connection:
    connection.close()
```

In [ ]:
```
%%shell
jupyter nbconvert --to html  /content/SQLite_2025.ipynb
```