LECTURE 12 SQLITE3

SQLite – The Lightweight and Quick Response Database!

- Fast, compact, and stores data in an easy to share file format
- Used inside countless mobile phones, computers, and applications
- Used by Facebook, Google, Dropbox and other ….

Adapted from Karen Works's COP4521 material, which in turn was adapted from:
https://www.analyticsvidhya.com/blog/2020/06/sql-for-beginners-analysts-sqlite-database-python/

More on SQLite

- Relational database management system based on SQL
- Designed for embedded devices that require fast and reliable data.
- Serverless
- Lightweight
- Requires zero-configuration
- It reads and writes directly to a disk file that can be easily copied
- Platform-independent
- Stores data in variable-length records which requires less memory and makes it run faster
- Is designed for improved performance, reduced cost, and optimized for concurrency.

Connecting to a SQLite db

- create a connection using the **connect()** method that returns a **Connection** object.
  - It accepts a path to the existing database.
  - If no database exists, it will create a new database on the given path (the directory needs to exist).
- 2) generate a **Cursor** object using the **cursor()** method which allows you to execute queries against the database

```
import sqlite3
# create new database
conn = sqlite3.connect('./sql_db/Demo_table.db')

# create Cursor to execute queries
cur = conn.cursor()

print('Database created.')
```

Commit and Close

1) Commit saves the operations that we performed on the database using the commit() method. If we don't commit our queries, then any changes we made to the database will not be saved automatically
2) Close the connection to the database to prevent the SQLite database from getting locked. When an SQLite database is locked, it will not be accessible by other users and will give an error

```
# save changes
conn.commit()
print('Changes saved.')

# close database connection
conn.close()
print('Connection closed.')
```

Running SQL Commands on SQLite

- SQL commands can be executed using the **execute()** method of the **Cursor** object.
  - SQLite3 SQL commands may be slightly different from the standard SQL.
- You just need to write your query inside quotes and you may choose to include a ';' which is a requirement in some databases but not in SQLite.

- *SQL keywords are case-insensitive so you can write the commands in UPPERCASE IF YOU WANT!*

Create Table

- **CREATE TABLE** command
  - *The **Primary key** is an attribute or set of attributes that can determine individual records in a table.*
  - *Defining an attribute **Not Null** will make sure there is a value given to the attribute (otherwise it will give an error).*

```
# create_customer_table.py
import sqlite3
conn = sqlite3.connect('./sql_db/Demo_table.db')
cur = conn.cursor()
# create table in database
cur.execute('''CREATE TABLE CUSTOMER(
User_ID INTEGER PRIMARY KEY NOT NULL,
Product_ID INTEGER NOT NULL,
Name TEXT NOT NULL,
Gender TEXT NOT NULL,
AGE INTEGER NOT NULL,
CITY TEXT);
''')
# commit and save changes to database
conn.commit()
```

Create product table

```
#create_product_table.py
import sqlite3
conn = sqlite3.connect('./sql_db/Demo_table.db')
cur = conn.cursor()
# create table in database
cur.execute('''CREATE TABLE Product(
Product_ID INTEGER PRIMARY KEY NOT NULL,
ProductName TEXT NOT NULL,
COST REAL NOT NULL,
DESCRIPTION TEXT);
''')
# commit and save changes to database
conn.commit()
```

Droping a table

- **DROP TABLE <tablename> command**
  - *drop an entire table from the database and not just its records*
  - *be extra careful before you run this command because all the records in the table along with the table structure will be lost after this!*

```
conn = sqlite3.connect('./sql_db/Demo_table.db')
cur = conn.cursor()
# drop table from database
cur.execute('"Drop table Customer"')
conn.commit()
```

Insert values into a table

- **INSERT INTO** SQL command to add values to the table.

INSERT INTO table_name (column1, column2, column3, …)

VALUES (value1, value2, value3, …);

adding values for all the columns in the table:

INSERT INTO table_name

VALUES (value1, value2, value3, …);

Insert data into the tables

```
# init_customer_table.py
import sqlite3
conn = sqlite3.connect('./sql_db/Demo_table.db')
cur = conn.cursor()

# execute one command
cur.execute('''Insert Into Customer ('User_ID','Product_ID','Name','Gender','AGE','CITY')
Values (1000, 3, 'Princess Diana', 'Female', 28, 'Amazons');''')
#execute multiple commands
cur.executescript('''Insert Into CUSTOMER Values
            (1001, 2, 'Clark Kent', 'Male', 36, 'Metropolis');

            Insert Into CUSTOMER Values
            (1003, 4, 'Bruce Wayne', 'Male', 39, 'Gotham City');
            ''')
# commit and save changes to database
conn.commit()
```

Insert data into the tables

```
# init_product_table.py

import sqlite3
conn = sqlite3.connect('./sql_db/Demo_table.db')
cur = conn.cursor()

products = [(2, 'Robot', 100.00, 'Robot Toy'),
            (3, 'Crown', 1000.00, '???'),
            (4, 'Movie', 10.00, '')]



cur.executemany('Insert Into product Values (?,?,?,?)', products)

# commit and save changes to database
conn.commit()
```

Insert data into the tables

```
# Insert multiple values into table at once

cur.execute('''Insert Into Product ('Product_ID','ProductName',’Cost’)
Values (1, ‘Teddy Bear', 28);'‘’)

cur.execute('''Insert Into Product ('Product_ID','ProductName',’Cost’)
Values (2, ‘matchbox car', 8);'‘’)

cur.execute('''Insert Into Product ('Product_ID','ProductName',’Cost’)
Values (3, ‘hat’, 12);'‘’)

cur.execute('''Insert Into Product ('Product_ID','ProductName',’Cost’)
Values (4, ‘sun glasses’, 18);'‘’)
```

Display the table

- **Run query:** `SELECT * from table_name`

```
# display_table.py
import sys
import sqlite3

table = 'customer'
if (len(sys.argv) == 2) :
    table = sys.argv[1]

conn = sqlite3.connect('./sql_db/Demo_table.db')
cur = conn.cursor()

t = cur.execute('SELECT * from ' + table)

for row in t:
    print(row)
```

Delete ROWS FROM a SQL table

- the DELETE statement
- Make sure to use the WHERE clause otherwise all the records will be deleted from the table!
- DELETE FROM table_name
- WHERE condition;

Delete ROWS FROM a SQL table

```
# delete table
conn.execute('''Delete from customer
                Where User_ID = 1006; '‘’)

cur.execute('''Select * from Customer;''')
```

Update Column Values

- **UPDATE** SQL command.
- modify existing records in a table
- Always make you sure you provide which records need to be updated in the WHERE clause otherwise all the records will be updated!

```
UPDATE table_name
SET column1 = value1, column2 = value2, …
WHERE condition;
```

Update record example

```
cur.execute('''Insert Into Customer ('User_ID','Product_ID','Name','Gender','AGE','CITY’)
Values (1006, 3, 'Princess Diana', 'Female', 28, 'Amazons');''')

# UPDATE statement
conn.execute('''Update Customer
                Set AGE = 48
                Where User_ID = 1006; '“’)
```

Qeury

- Execute a SELECT-FROM-WHERE statement

- ***SELECT column1, column2, … FROM table_name;***

- ***If you instead wanted to fetch values for all the attributes in the table, use the * character instead of the column names:***

- ***SELECT * FROM table_name;***

Fetching Records from a SQL table

- To fetch multiple rows, you can execute a SELECT statement and iterate over it directly using only a single call on the Cursor object:

```
# iterate over the rows
for row in cur.execute('SELECT Name FROM CUSTOMER;'):
    print(row)
```

Fetching Records from a SQL table

- fetchall() method which returns all the records in a list format:

```
# Fetch all rows of query result which returns a list
cur.execute('SELECT * FROM CUSTOMER;').fetchall()
```

More complex queries

```
# Fetch all rows of query result which returns a list
cur.execute('SELECT * FROM CUSTOMER where age=48;').fetchall()
```

Order by clause in SQL

- ORDER BY clause is used to sort the result into ascending or descending order using the keywords ASC or DESC respectively.
- By default, it sorts the records in ascending order:

```
SELECT column1, column2, … FROM table_name
ORDER BY column1, column2, … ASC|DESC;
```

```
# Fetch all rows of query result which returns a list
cur.execute('SELECT * FROM CUSTOMER ORDER BY age DESC;').fetchall()
```