SQLite3 and Python Primer

CMPSC 305 – Database Systems

ALLEGHENY COLLEGE

Big Data

- Upwards of 2.7 Zetabytes of data exist in the digital universe
- YouTube users upload 48 hours of new video every minute
- Increase in unstructured data: text, photos, etc. https://www.waterfordtechnologies.com/big-data-interesting-facts/

Multiples of bytes V•T•E

Decimal			Binary			
Value	Metric		Value	IEC		JEDEC
1000	kB	kilobyte	1024	KiB	kibibyte	KB kilobyte
1000^2	MB	megabyte	1024^2	MiB	mebibyte	MB megabyte
1000^3	GB	gigabyte	1024^3	GiB	gibibyte	GB gigabyte
1000^4	TB	terabyte	1024^4	TiB	tebibyte	–
1000^5	PB	petabyte	1024^5	PiB	pebibyte	–
1000^6	EB	exabyte	1024^6	EiB	exbibyte	–
1000^7	ZB	**zettabyte**	1024^7	ZiB	zebibyte	–
1000^8	YB	yottabyte	1024^8	YiB	yobibyte	–

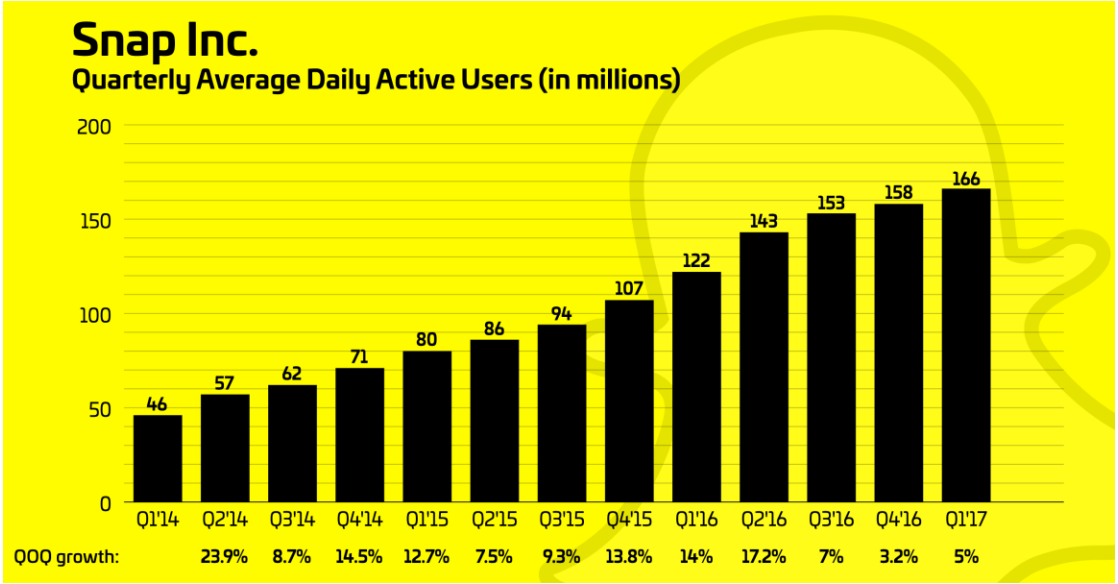
Orders of magnitude of data

Facebook's Daily Data Use

Facebook processes:

- 2.5 billion pieces of content
- Upwards of 500 terabytes of data each day from status and location details
- Processing in 2.7 billion Like actions
- 300 million photos per day
- Scans roughly 105 terabytes of data each half hour
- 100 petabytes of data are stored in a single Hadoop disk cluster (a distributed system for data management)

Current Estimates for Users Online



- Facebook: 2.7 Billion Active users
- Amazon: 112 Million (US users)
- SnapChat: 238 million daily active users worldwide
- Google: 4.39 Billion internet users (worldwide)
- Instagram: 1 Billion monthly active users, 500 Million each day.

Lots of names, photos, passwords and posts to record!

How are we to manage all this data?

Automate the database management processes using software!!

Standardized Database Access with Python

PEP 0249

- Python Database API Specification v2.0
- https://www.python.org/dev/peps/pep-0249/
- A standard API to encourage similarity between the Python modules used for accessing databases.
- Does not provide a library nor a module, just specifications on how to make them
- Third party modules may adhere to these specifications

Steps to run a command in SQL using Python

Five basic steps to using a database according to the Python Database API Specification v2.0

Building automated framework in Python3

- Step 1: Defining the query
- Step 2: Connecting to the database
- Step 3: Execute the query
- Step 4i, (SELECT): Analyze the result
- Step 4ii, or (UPDATE): Commit the change
- Step 5: Cleaning up; close the database connection

Nice tutorial: http://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html


KEEP
CALM
AND
LET'S
CODE

Setting Up Virtual Environment

- Create a project directory

```
mkdir week08
cd week08
```

- Create virtual environment using Python

```
python3 -m venv myenv
# see the file tree
find . -not -path '*/\.*'
```

- Activate myenv the virtual environment

```
source myenv/bin/activate  # macOS/Linux
myenv\Scripts\activate     # Windows
```

- Deactivate the virtual environment

```
deactivate
```

Making Useful Strings

A concatenated string

Note the 'f' before the quotes to enable formatting

```
myCollege_str = "Allegheny"
mesg_str = f"I go to {myCollege_str }!!"
print(mesg_str)
```

```
myCollege_str = "Allegheny"
myMajor_str = "CompSci"
mesg_str = f"At {myCollege_str}, my major is {myMajor_str}"
print(mesg_str)
```

Adding quotes: note the forward slashes in strings

```
iSay_str = "Cool"
mesg_str = f"They say it is a \"{iSay_str}\" major"
print(mesg_str)
```

Making Useful Strings

A concatenated string

- Queries are strings of code that can be created by Python.
- These queries can be sent to database management software

Making a Query Statement

```
table_str = "Instructor"
a1_str = "deptName"
a2_str = "course"
name_str = "Miller"
```

```
myQuery_str = f"SELECT {a1_str}, {a2_str} FROM {table_str} WHERE name == \"{name_str}\""
```

```
print(myQuery_str)
```

Making Useful Strings

A concatenated string

Making An Insert Statement

```
myTable = "Instructor"
PersonID = "10101"
name_str = "Miller"
student = "S1"
```

```
insert_str = f"INSERT INTO {myTable} VALUES({PersonID}, \"{name_str}\", \"{student}\")"
```

```
print(insert_str)
```

Choose variable names that make sense to your code!

Python: A Database Management System (DMS)

Let's Try It Out!

- Locate the sandbox database builder file sandbox/campusDB build.txt and make your DB.
- Call up your favourite editor and let's begin programming.

Automatic CREATE and INSERT statements using SQLite3

```
# The database managing code begins here. This code creates
# the schema in order to place data into tables.
#
# Usage: python3 simpleInsert1.py
# Outputs text and a table.
# Requirement: None

import sqlite3

dbFilename_str = "myDB_i.sqlite3" #establish the DB file
conn = sqlite3.connect(dbFilename_str) # open connection to the DB

print("\t _____")
print("\n\t Program to demo automatic INSERT statements using SQLite3")
print("\t ----------------------------------------\n")
```

Automatic CREATE statements using SQLite3

```
myTable_str = "StudyMusic" #define the table
attribute1_str = "id INTEGER NOT NULL " # define first attribute statement
attribute2_str = "favSong VARCHAR" # define first attribute statement
attribute3_str = "BandName VARCHAR" # define first attribute statement


# Create the table creation string
myCreation_str = f"CREATE TABLE {myTable_str} ({attribute1_str}, {attribute2_str}, {attribute3_str})"
```

Automatic CREATE statements using SQLite3

```
try:
        print(f"\t my insert: {myCreation_str}")

        # pass the table building string to sqlite3 library
        conn.execute(myCreation_str)

        # Save (i.e., commit) the changes
        conn.commit()

except sqlite3.OperationalError: # does the table already exist? If so, ignore creation statement
        print("\t [-] Note: The table already exists.")
```

Automatic INSERT statements using SQLite3

```
# Insert a row of data

myTable_str = "StudyMusic" # define the table
attrID_str = "10"
attrSONG_str = "Yello Submarine"
attrARTIST_str = "The Beatles"

print(f"\t [+] Simple INSERT into Table {myTable_str}")
```

Automatic INSERT statements using SQLite3

```
# define the insert statement

myInsert_str = f"INSERT INTO {myTable_str} VALUES ({attrID_str}, \"{attrSONG_str}\", \"{attrARTIST_str}\")"

print(f"\t [+] my insert statement {myInsert_str}")


# pass the INSERT string to sqlite3 library
conn.execute(myInsert_str)

# Save (i.e., commit) the changes
conn.commit()
```

Automatic QUERY statements using SQLite3

```
# Do a simple query to check that insert worked
myQuery_str = f"SELECT * FROM {myTable_str}"
result = conn.execute(myQuery_str) # run the query
tables = result.fetchall() # collect query for processing
print("\t "+myQuery_str)
print("\t [+] Results: ")
for i in tables:
        print(f"\t {i}") # show results of query


conn.close() # close the database connection
```

Python: A Database Management System (DMS)

```
import sqlite3

name_str = "hz"

DB_FILE = "myCampusDB.sqlite3"

def connect():
        conn = sqlite3.connect(DB_FILE)
        conn.row_factory = sqlite3.Row
        conn.execute("PRAGMA foreign_keys = ON;")
        return conn
```

Python: A Database Management System (DMS)

```
def list_tables(conn):
        results = conn.execute("SELECT name FROM sqlite_master WHERE
type='table' AND name NOT LIKE 'sqlite_%';")
        rows = results.fetchall()
        if not rows:
                print("(no tables found)")
        else:
                print("Tables:")
                for r in rows:
                        print(" -", r[0])
```

Python: A Database Management System (DMS)

```
def show_schema(conn, table=None):
        results = conn.execute("SELECT name, sql FROM sqlite_master WHERE type='table' AND
name NOT LIKE 'sqlite_%';")
        for row in results.fetchall():
                print(row["sql"])
                print()
```

Python: A Database Management System (DMS)

```
def collect_sql():
        print("Enter a SELECT statement (end with semicolon ';'):")
        buf = [] # creates an empty list
        while True:
                line = input("sql> ")
                buf.append(line)
                if ";" in line:
                        break
        return "\n".join(buf)
```

Python: A Database Management System (DMS)

```
def run_select(conn):
        sql = collect_sql()
        if not sql.strip().lower().startswith("select"):
        print("Only SELECT statements are allowed here.")
                return
        try:
                results = conn.execute(sql) # ! the sqlite3 library sends the SQL text you typed directly to
the SQLite engine.
                rows = results.fetchall()
                if not rows:
                        print("(no rows)")
                else:
                        for row in rows:
                                print(tuple(row)) # simple tuple output
        except sqlite3.Error as e:
                print(f"SQLite error: {e}")
```

Python: A Database Management System (DMS)

```
def main():
        print("\nHi,", name_str, "! Simple DBMS loaded.\n")
        conn = connect()
        while True:
                print("\nMenu:")
                print(" 1) List tables")
                print(" 2) Show schema")
                print(" 3) Run SELECT")
                print(" q) Quit")
                choice = input("Choose (1-3 or q): ").strip().lower()

                if choice == "1":
                        list_tables(conn)
                elif choice == "2":
                        show_schema(conn)
                elif choice == "3":
                        run_select(conn)
                elif choice == "q":
                        print("Goodbye!")
                        break
                else:
                        print("Invalid choice. Try again.")
if __name__ == "__main__":
main()
```

Now Modify Your DMS!

Do Something Different!

- Try adding query code for other tables.
- What attributes can you query?
- Can you write code for a query involving two tables?