Development Seminar

Working with Data

Admin

- News
- Discussions?
- References for these slides
  - https://medium.com/analytics-vidhya/programming-with-databases-in-python-using-sqlite-4cecbef51ab9
- Let's talk about the soft skills podcast
  - If this is Mon/Tues
-

Disclaimer

- As I said on day 1.
  - We need to expose all students to some basic data handling in a required class.
  - In this slideset I'm going to try to summarize comp405 in a week or less.
    - This is not a substitute for a real database class!!!!!
    - That said, if you have questions ask them.
  - Part of what we try to do in this class is to give you a taste of something that every graduate should have
  - Other nearby ABET accredited CS program's CS load compared to ours.

Databases

- Historically Databases come in lots of difference types
  - Hierarchical databases
  - Network databases
  - Relational databases
  - Object-oriented databases
  - Graph databases
  - ER model databases
  - Document databases

Databases

- Realistically, today most people care about 2 of those
  - Relational Databases (SQL)
  - Document Databases (NoSQL)
  - Note on Pronouncing these:
    - Lots of disucussion:
      - https://www.khanacademy.org/computing/computer-programming/sql/sql-basics/v/s-q-l-or-sequel
      - https://database.guide/is-it-pronounced-s-q-l-or-sequel/
    - sqlServer vs MySQL
    - In general I find Americans more likely to use the old IBM “Sequel” pronunciation and the rest of the world to use S-Q-L
    - I learned SQL from a recent immigrant so I often switch back and forth

Relational Databases

- Relational Databases
  - Developed in the early 1970s
  - By 1980’s were all but the only type used
  - (document databases making a big inroad in this dominance this decade)
  -

Relational Database fundamentals

- Database is made up of Tables
  - Tables consist of rows and columns such that:
    - There is no significance to the order of the columns or rows.
    - Each row contains one and only one value for each column.
    - Each value for a given column has the same type.
      - Caveat: blob field equivalent of void* in most DBMS
    - Each table in the database should hold information about one specific thing, such as employees, products, or customers.
    - Each row should hold a unique value
      - At least one column value (or combo) for each row should be unique
  -

emp_ID	emp_first_name	emp_last_name	emp_phone
10057	Barbara	Ku	1096
10693	Jessica	Anne	7821

RDMS fundamentals II

- Every table has a ‘primary key’
  - The value in every row (column or columns) that allows the row to be uniquely identified
  - Eg:

CUSTOMERS TABLE

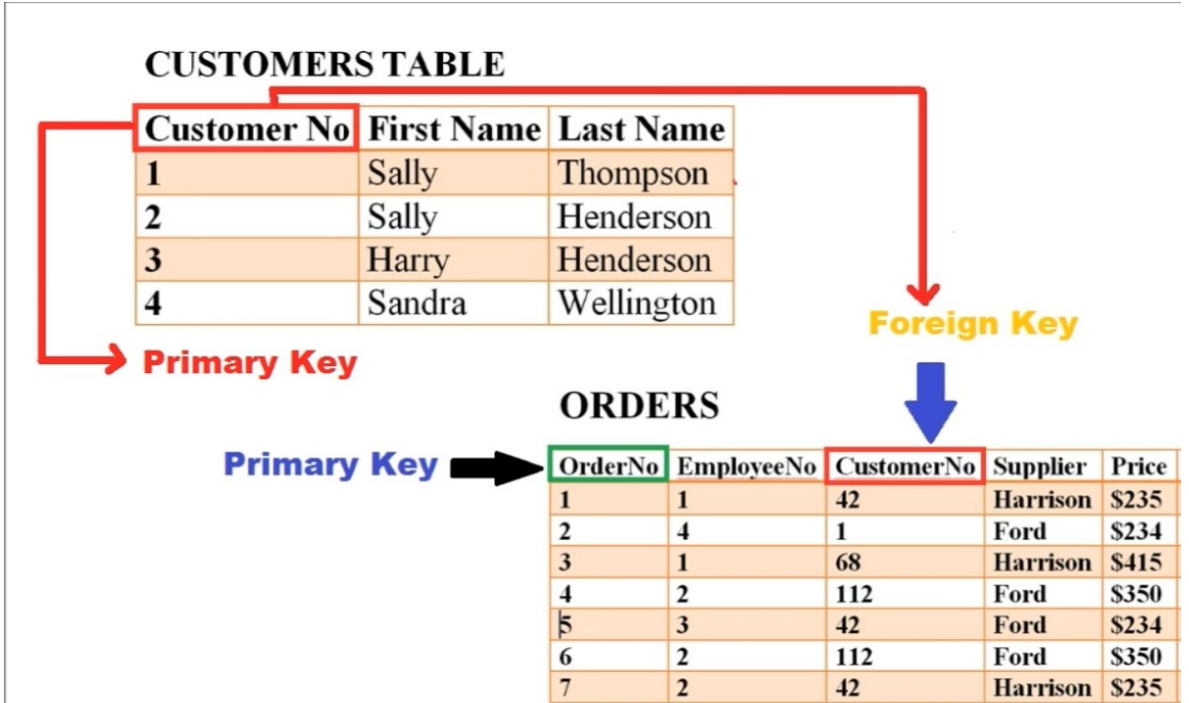
Customer No	First Name	Last Name
1	Sally	Thompson
2	Sally	Henderson
3	Harry	Henderson
4	Sandra	Wellington

Primary Key

- Usually numeric,
  - But doesn’t have to be

RDMS III:

- Foreign Keys
  - Every table can have one or more foreign keys
  - Column in table B has the value of the primary key from Table A
    - Links the records/rows in tables A and B



Tables, Rows and Columns oh my!

- Designing a database is a science and art of its own
  - What should be rows, what should be columns
  - Which things should be tables
    - One-one, one-many, many-many relationships among data in tables
    - Database managers have to make these decisions based on technological and business concerns
      - And these can change over time like programming
      - But harder to refactor the database.
  - Making these decisions are beyond the scope of this course
  - I'll assume any significant database is built for you.

SQL

- When we interact with a Relational Databas SQL is the only game in town.
  - The full official standard is currently ISO 9075
  - 14 documents from the ISO each 80+ pages and several cost a fair bit of money
    - Giant, in places somewhat contradictory
    - No one implements the whole thing
    - With stored procedures is Turing complete.
      - Turing complete?

Sqlite

- There are many sql/RDMS databases out there
  - For my examples we'll use sqlite
    - Very small lightweight RDMS database
  - Comes pre setup with python
    - Though I'll encourage you to install the graphical editor.
  -

Ubuntu 24.04 and Friends

- To install all the tools you'll need on ubuntu 24.04 (most recent LTS – also same for 24.10 etc)
  - And linux mint and kubuntu based on 24.04
  - sudo apt install sqlite3
  - sudo apt install sqlitebrowser
- And you are Done for python

Mac and Windows

- There are installers that you can download from the official site for the sqlite browser
- https://sqlitebrowser.org/
- At this point the python people should be all set.

Java with Sqlite

- Grab the sqlite jar file for jdbc from its repository
  - https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc
  - As of Jan 2025 the latest version was
    - sqlite-jdbc-3.48.jar
  - Then to work with sqlite from intellij Idea (the jetbrains java IDE)
    - https://www.jetbrains.com/help/idea/connecting-to-a-database.html
    - There are directions for more than a dozen databases there. But consolidated into a page of links, so click for the directions for sqlite.

Other options

- If you want to do this in golang
  - https://github.com/mattn/go-sqlite3 is my preferred driver
  - Tutorial: https://www.codeproject.com/Articles/5261771/Golang-SQLite-Simple-Example
  - If you want to do this with rust
  - Built in support
- If you are working in C/C++ you need to provide *configure* files and *make* files so I can build cross platform but sqlite was built in c/c++
  - https://www.sqlite.org/cintro.html

SQL

- Manageable subset
  - Lets explore a manageable subset of SQL
  - My examples will use python and sqlite.
  - First I’ll use sqlite to create the database

Create the database

- First pass at program:

```
import  sqlite3
from typing import Tuple
def open_db(filename:str)->Tuple[sqlite3.Connection, sqlite3.Cursor]:
    db_connection = sqlite3.connect(filename)#connect to existing DB or create new one
    cursor = db_connection.cursor()#get ready to read/write data
    return db_connection, cursor

def close_db(connection:sqlite3.Connection):
    connection.commit()#make sure any changes get saved
    connection.close()

def main():
    conn, cursor = open_db("demo_db.sqlite")
    print(type(conn))
    close_db(conn)

if __name__ == '__main__':
    main()
```

Look at that

- If we run this, it creates and empty database.
- In pycharm, the new database appears in the project main directory
  - Use the sqlite browser to look
- Empty database
- Note! For java and some other non-python database drivers it won't create the database till we make our first table

Doing stuff to the database

- Once you have your database
  - Use the cursor to execute sql statements
  - Eg:
    - cursor.execute("SELECT id, name, marks from SCHOOL"):

SQL: Create Table

- Create table statement used to create a new table in the db
- Official sqlite docs:
  - CREATE TABLE [IF NOT EXISTS] [schema_name].table_name (
  - column_1 data_type PRIMARY KEY,
  - column_2 data_type NOT NULL,
  - column_3 data_type DEFAULT 0,
  - table_constraint
  -);
- Notice that SQL statements end in ‘;’

Create Table example

- Lets create two tables (then look at DB again)

```
def setup_db(cursor:sqlite3.Cursor):
    cursor.execute('''CREATE TABLE IF NOT EXISTS students(
    banner_id INTEGER PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    gpa REAL DEFAULT 0,
    credits INTEGER DEFAULT 0
    );''')
    cursor.execute('''CREATE TABLE IF NOT EXISTS course(
    course_prefix TEXT NOT NULL,
    course_number INTEGER NOT NULL,
    cap INTEGER DEFAULT 20,
    description TEXT,
    PRIMARY KEY(course_prefix, course_number)
    );''')
```

Now the last table

- Now lets ‘hook/glue it all together’ (lots to unpack here- describe)

```
cursor.execute('''CREATE TABLE IF NOT EXISTS class_list(
registration_id INTEGER PRIMARY KEY,
course_prefix TEXT NOT NULL,
course_number INTEGER NOT NULL,
banner_id INTEGER NOT NULL,
registration_date TEXT,
FOREIGN KEY (banner_id) REFERENCES students (banner_id)
ON DELETE CASCADE ON UPDATE NO ACTION,
FOREIGN KEY (course_prefix, course_number) REFERENCES course
(course_prefix, course_number)
ON DELETE CASCADE ON UPDATE NO ACTION
);''')
```

Putting Data in the tables

- Use the insert statement to put data into tables

```
def make_intial_students(cursor:sqlite3.Cursor):
    cursor.execute(f'''INSERT INTO STUDENTS (banner_id, first_name, last_name, gpa, credits)
                   VALUES (1001, "John", "Santore", {random.uniform(0.0,4.0)},
{random.randint(0,120)})''')
    cursor.execute(f'''INSERT INTO STUDENTS(banner_id, first_name, last_name, gpa, credits)
            VALUES(1002, "Enping", "Li", {random.uniform(0.0, 4.0)}, {random.randint(0,
120)})''')
    cursor.execute(f'''INSERT INTO STUDENTS(banner_id, first_name, last_name, gpa, credits)
            VALUES(1003, "Margaret", "Black", {random.uniform(0.0, 4.0)}, {random.randint(0,
120)})''')
    cursor.execute(f'''INSERT INTO STUDENTS(banner_id, first_name, last_name, gpa, credits)
            VALUES(1004, "Seikyung", "Jung", {random.uniform(0.0, 4.0)}, {random.randint(0,
120)})''')
    cursor.execute(f'''INSERT INTO STUDENTS(banner_id, first_name, last_name, gpa, credits)
            VALUES(1005, "Haleh", "Khojasteh", {random.uniform(0.0, 4.0)},
{random.randint(0, 120)})''')
```

And the courses

- For good measure lets do the courses

```
def make_initial_courses(cursor:sqlite3.Cursor):
    cursor.execute(f'''INSERT INTO COURSE (course_prefix, course_number, cap,
description)
        VALUES ('COMP', 151, 24, 'This is the intro course, you will learn to program,
maybe for the first time')''')
    cursor.execute(f'''INSERT INTO COURSE (course_prefix, course_number, cap,
description)
        VALUES ('COMP', 490, 20, 'This is the final course. You will get a chance to
apply much of what you learned throughout the undergrad degree')''')
    cursor.execute(f'''INSERT INTO COURSE (course_prefix, course_number, cap,
description)
        VALUES ('MATH', 130, 20, 'This course is changing to include much more on graph
theory and number bases/systems')''')
```

Now the interesting data

Now We need to put data into the table with the foreign keys (note that I'm not explicitly adding primary keys this time, Also arbitrary column order

```
def make_initial_classLists(cursor:sqlite3.Cursor):
    cursor.execute(f'''INSERT INTO CLASS_LIST (banner_id, course_prefix, course_number,  registration_date)
    VALUES(1001, 'Comp', 490, DATE('now'))
    ''')
    cursor.execute(f'''INSERT INTO CLASS_LIST (banner_id, course_prefix, course_number,  registration_date)
        VALUES(1002, 'Comp', 490, DATE('now'))
        ''')
    cursor.execute(f'''INSERT INTO CLASS_LIST (banner_id, course_prefix, course_number,  registration_date)
            VALUES(1003, 'Comp', 490, DATE('now'))
            ''')
    cursor.execute(f'''INSERT INTO CLASS_LIST (banner_id, course_prefix, course_number,  registration_date)
            VALUES(1004, 'Comp', 490, DATE('now'))
            ''')
    cursor.execute(f'''INSERT INTO CLASS_LIST (banner_id,  course_prefix, course_number, registration_date)
            VALUES(1005, 'Comp', 490, DATE('now'))
            ''')
```

- I've just put the data in the database because it is all constants written by me the programmer
- But when dealing with any data from users **always** use parameterized insert to let library do sanitization of data
  - #records or rows in a list
  - records = [(1, 'Glen', 8)
(2, 'Elliot', 9),
(3, 'Bob', 7)]
  - #insert multiple records in a single query
  - c.executemany('INSERT INTO students VALUES(?,?,?);',records);

Dates

- Some RDMS have a datetime type for fields
- Not sqlite
  - Dates can be stored as Strings
  - Dates can be stored as floats/Reals (use julianday function)
  - Dates can be stored as ints (unix time epoch)
- Two builtin functions you want to think about (for string version)
  - DATE
  - DATETIME
  - Both take params, most common value is ‘now’

More data maintenance

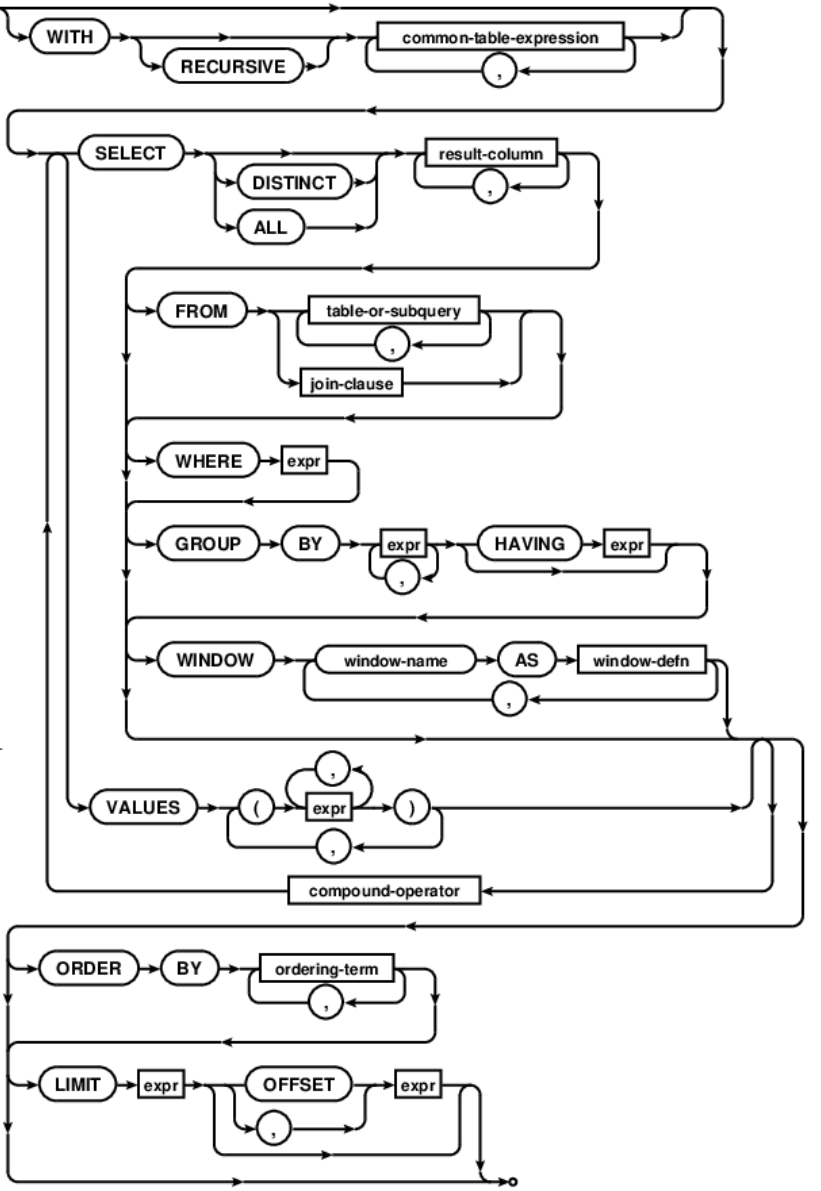
- There is more to SQL for data maintenance than inserting
  - DELETE Statement
    - Simple example: DELETE FROM table WHERE search_condition;
  - UPDATE data:
    - UPDATE table
    - SET column_1 = new_value_1,
    - column_2 = new_value_2
    - WHERE
    - search_condition
    - ORDER column_or_expression
    - LIMIT row_count OFFSET offset;

More Data maintenance

- Alter table
  - Can add or delete or rename columns and more
  - Fairly complex syntax, take comp405
- Drop table
  - Remove table (and all its data) from the database (schema)
  -

Using data

- Using data in an RDMS
- Aka welcome to select
  - Even in small sqlite the syntax is complex
  - This is the official graph for the finite sta machine for select



Simple Select

- The simplest version of Select
  - SELECT <columns> FROM <table>
  - Or more interesting:
    - SELECT <columns> FROM <table> WHERE <column name> = <value>
    -

Simple Select in python driver

- Lets see a simple select from one table:

```
def show_simple_select(cursor:sqlite3.Cursor):
    cutoff = float(input("What should the GPA cutoff be?"))
    #question to class-what about security issues here?
    #Discuss
    result = cursor.execute(f'SELECT * from STUDENTS WHERE\
     gpa < {cutoff}')
    for row in result:
        print(f'BannerId: {row[0]}\nName: {row[1]}\
              {row[2]}\nGPA:{row[4]}')
```

Parameterized select

- You want to parameterize most select statements with user data in them
- query = “Select * from shows where year < ? and rank > ?”
- args = (year, ranking) #where these come from user input
- cursor.execute(query, args)

More interesting Select

- More interesting data is found by joining two tables
  - Terminology
    - Inner join
    - Outer join
    - See comp405
  - But rule of thumb: only join tables that have foreign keys, otherwise you will have giant mess on your hands

More interesting select

- Example with lots to unpack and discuss

```
def show_select_with_join(cursor:sqlite3.Cursor):
    result = cursor.execute(f'''SELECT first_name, last_name, credits
    FROM STUDENTS
    INNER JOIN CLASS_LIST ON
    STUDENTS.banner_id = CLASS_LIST.banner_id
    WHERE (STUDENTS.credits < 60
        and CLASS_LIST.course_prefix = 'Comp'
        and CLASS_LIST.course_number = 490)''')
    for row in result:
        print(f'{row[0]} {row[1]} somehow registered for Comp490 with only
{row[2]} credits')
```

Note: inner join on foreign key

Where clause columns need not be in result set.

RDMS finish

- Just a quick taste introduction
- Podcast
- Soft Skills listening Assignment
  - Lets talk