

Lecture 6: In-class activity - SQL complex queries

Let's setup the SQL environment

```
In [1]: #Install pysqlite3 for python and import pandas to use later
#!pip install pysqlite3
from sqlite3 import dbapi2 as sqlite3
print(sqlite3.sqlite_version)
import pandas as pd
from IPython.display import display, HTML
```

3.45.3

Let's define some helper functions for running queries and printing results

```
In [2]: dbname = "music_streaming4.db"

def printSqlResults(cursor, tblName):
    try:
        df = pd.DataFrame(cursor.fetchall(), columns=[i[0] for i in cursor.description])
        display(HTML("<b><font color=Green> " + tblName + "</font></b>" + df.to_html()))
    except:
        pass

def runSql(caption, query):
    conn = sqlite3.connect(dbname) # Connect to the database
    cursor = conn.cursor() # Create a cursor (think: it's like a "pointer")
    cursor.execute(query) # Execute the query
    printSqlResults(cursor, caption) # Print the results
    conn.close()

def runStepByStepSql(query, fromline):
    lines = query.strip().split('\n')
    for lineidx in range(fromline, len(lines)):
        partial_query = '\n'.join(lines[:lineidx])
        caption = 'Query till line:' + partial_query
        runSql(caption, partial_query + ';')
```

Let's setup a Schema and insert some data

```
In [3]: # Connect to database (creates the file if it doesn't exist)
"""
1. Connections: A connection represents a connection to a database through
which we can execute SQL queries. The dbname here specifies the database.
In SQLite, if the DB doesn't exist, it will be created.
2. Cursors: A cursor is an object associated with a database connection.
It allows you to execute SQL queries, fetch query results.
"""
conn = sqlite3.connect(dbname)
```

```

cursor = conn.cursor()

# Create the Users table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Users (
    user_id INTEGER PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);
""")

# Create the Songs table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Songs (
    song_id INTEGER PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    artist VARCHAR(100) NOT NULL,
    genre VARCHAR(100)
);
""")

# Create the Listens table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Listens (
    listen_id INTEGER PRIMARY KEY,
    user_id INTEGER NOT NULL,
    song_id INTEGER NOT NULL,
    rating FLOAT,
    listen_time TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (song_id) REFERENCES Songs(song_id)
);
""")

# Create the recommendations table
cursor.execute("""
CREATE TABLE IF NOT EXISTS Recommendations (
    user_id INTEGER NOT NULL,
    song_id INTEGER NOT NULL,
    recommendation_id not NULL,
    recommendation_time TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (song_id) REFERENCES Songs(song_id)
);
""")

# Commit changes and close the connection
conn.commit()
conn.close()

```

```

In [4]: # Connect to database again and insert sample data
conn = sqlite3.connect(dbname)
sqlite3.enable_callback_tracebacks(True)

cursor = conn.cursor()
cursor.execute("delete from Songs;")

```

```
cursor.execute("delete from Users;")
cursor.execute("delete from Listens;")
cursor.execute("delete from Recommendations;")

# Insert sample users
cursor.execute("""
INSERT INTO Users (user_id, name, email)
VALUES
    (1, 'Mickey', 'mickey@example.com'),
    (2, 'Minnie', 'minnie@example.com'),
    (3, 'Daffy', 'daffy@example.com'),
    (4, 'Pluto', 'pluto@example.com');
""")

# Insert sample songs from Taylor Swift, Ed Sheeran, Beatles
cursor.execute("""
INSERT INTO Songs (song_id, title, artist, genre)
VALUES
    (1, 'Evermore', 'Taylor Swift', 'Pop'),
    (2, 'Willow', 'Taylor Swift', 'Pop'),
    (3, 'Shape of You', 'Ed Sheeran', 'Rock'),
    (4, 'Photograph', 'Ed Sheeran', 'Rock'),
    (5, 'Shivers', 'Ed Sheeran', 'Rock'),
    (6, 'Yesterday', 'Beatles', 'Classic'),
    (7, 'Yellow Submarine', 'Beatles', 'Classic'),
    (8, 'Hey Jude', 'Beatles', 'Classic'),
    (9, 'Bad Blood', 'Taylor Swift', 'Rock'),
    (10, 'DJ Mix', 'DJ', NULL);
""")

# Insert sample listens
cursor.execute("""
INSERT INTO Listens (listen_id, user_id, song_id, rating)
VALUES
    (1, 1, 1, 4.5),
    (2, 1, 2, 4.2),
    (3, 1, 6, 3.9),
    (4, 2, 2, 4.7),
    (5, 2, 7, 4.6),
    (6, 2, 8, 3.9),
    (7, 3, 1, 2.9),
    (8, 3, 2, 4.9),
    (9, 3, 6, NULL);
""")

# Commit changes and close the connection
conn.commit()
conn.close()

runSql('Users', "select * from Users;")
runSql('Songs', "select * from Songs;")
runSql('Listens', "select * from Listens;")
```

Users

user_id	name	email
1	Mickey	mickey@example.com
2	Minnie	minnie@example.com
3	Daffy	daffy@example.com
4	Pluto	pluto@example.com

Songs

song_id	title	artist	genre
1	Evermore	Taylor Swift	Pop
2	Willow	Taylor Swift	Pop
3	Shape of You	Ed Sheeran	Rock
4	Photograph	Ed Sheeran	Rock
5	Shivers	Ed Sheeran	Rock
6	Yesterday	Beatles	Classic
7	Yellow Submarine	Beatles	Classic
8	Hey Jude	Beatles	Classic
9	Bad Blood	Taylor Swift	Rock
10	DJ Mix	DJ	None

Listens

listen_id	user_id	song_id	rating	listen_time
1	1	1	4.5	None
2	1	2	4.2	None
3	1	6	3.9	None
4	2	2	4.7	None
5	2	7	4.6	None
6	2	8	3.9	None
7	3	1	2.9	None
8	3	2	4.9	None
9	3	6	NaN	None

Basic SQL queries (ORDER BY, GROUP BY, LIMIT, JOINS, LEFT JOINS)

```
In [5]: """ Goal: Learn basic forms of SELECT, FROM, WHERE, DISTINCT """

qry_classic_songs = """
-- Find the titles and artists of songs in the "Classic" genre.
SELECT Songs.title, Songs.artist
FROM Songs
WHERE Songs.genre = 'Classic';"""
runSql('Classic songs', qry_classic_songs)

qry_genres = """
-- List of all genres in the Songs table
SELECT genre
FROM Songs;"""
runSql('All genres in the Songs table', qry_genres)

qry_distinct = """
-- List of unique genres in the Songs table
SELECT DISTINCT genre
FROM Songs;"""
runSql('Unique genres in the Songs table', qry_distinct)

qry_taylor_count = """
-- Songs by Taylor Swift in different genres
SELECT genre, count(*) as num_songs
FROM Songs
where artist = 'Taylor Swift'
GROUP BY genre;"""
runSql('Count songs by Taylor Swift in different genres', qry_taylor_count)
```

Classic songs

title	artist
Yesterday	Beatles
Yellow Submarine	Beatles
Hey Jude	Beatles

All genres in the Songs table

genre
Pop
Pop
Rock
Rock
Rock
Classic
Classic
Classic
Rock
None

Unique genres in the Songs table

genre
Pop
Rock
Classic
None

Count songs by Taylor Swift in different genres

genre	num_songs
Pop	2
Rock	1

Experiment 1 with group by

```
In [6]: qry_group_by1 = """
SELECT genre, count(*) as num_songs
FROM Songs
GROUP BY genre;"""
runSql('Count songs by genre', qry_group_by1)
qry_group_by2 = """
SELECT genre, count(*) as num_songs
FROM Songs
GROUP BY artist, genre;"""
runSql('Count songs by genre-artist', qry_group_by2)
```

Count songs by genre

genre	num_songs
None	1
Classic	3
Pop	2
Rock	4

Count songs by genre-artist

genre	num_songs
Classic	3
None	1
Rock	3
Pop	2
Rock	1

Experiment 2 with group by: Unsafe! Why? Ed Sheeran has 3 songs in Rock and Taylor Swift has 1 in Rock. When we GROUP BY genre, what artist should you return for Rock genre? SQL engines will throw an error, or return a random artist in SELECT. Unsafe.

```
In [7]: qry_group_by_unsafe = """
SELECT artist, genre, count(*) as num_songs
FROM Songs
GROUP BY genre;"""
runSql('Count songs by genre [unsafe/wrong]', qry_group_by_unsafe)

"""Tip: Always make sure SELECT includes only columns in GROUP BY, or aggregate
of the GROUP BY or extraneous columns (e.g., SUM, COUNT, AVG, etc.)
Below we add artist to GROUP BY as well"""
qry_group_by_safe = """
SELECT genre, count(*) as num_songs
FROM Songs
GROUP BY artist, genre;"""
runSql('Count songs by artist-genre. Select artists-genre', qry_group_by_safe)
```

Count songs by genre [unsafe/wrong]

artist	genre	num_songs
DJ	None	1
Beatles	Classic	3
Taylor Swift	Pop	2
Ed Sheeran	Rock	4

Count songs by artist-genre. Select artists-genre

genre	num_songs
Classic	3
None	1
Rock	3
Pop	2
Rock	1

TO DO: Write a query that calculates average ratings of all songs. Only include songs with Listens

```
In [26]: %%time
avg_ratings = """
SELECT Songs.title, sum(Listens.rating)/count(Songs.title) as average from S
runSql('Average ratings with join ', avg_ratings)
```

Average ratings with join

title	average
Evermore	3.70
Hey Jude	3.90
Willow	4.60
Yellow Submarine	4.60
Yesterday	1.95

CPU times: user 4.36 ms, sys: 4.86 ms, total: 9.22 ms
Wall time: 5.78 ms

```
In [27]: %%time
avg_withoutnull_ratings = """
SELECT Songs.title, sum(Listens.rating)/count(Songs.title) as average from S
WHERE Listens.rating is not null GROUP BY Songs.title"""
runSql('Average ratings with join (without null)', avg_withoutnull_ratings)
```

Average ratings with join (without null)

title	average
Evermore	3.7
Hey Jude	3.9
Willow	4.6
Yellow Submarine	4.6
Yesterday	3.9

CPU times: user 5.36 ms, sys: 4.1 ms, total: 9.46 ms
Wall time: 5.87 ms


```
In [28]: %%time
avg_left_ratings_funtion = """
SELECT Songs.title, avg(Listens.rating) as average from Songs join Listens c
runSql('Average ratings with inbuilt function', avg_left_ratings_funtion)
```

Average ratings with inbuilt function

title	average
Evermore	3.7
Hey Jude	3.9
Willow	4.6
Yellow Submarine	4.6
Yesterday	3.9

CPU times: user 3.94 ms, sys: 3.52 ms, total: 7.46 ms
Wall time: 4.34 ms

```
In [29]: %%time
avg_left_ratings = """
SELECT Songs.title, avg(Listens.rating) as average from Songs left join List
runSql('Average ratings with left join', avg_left_ratings)
```

Average ratings with left join

title	average
Bad Blood	NaN
DJ Mix	NaN
Evermore	3.7
Hey Jude	3.9
Photograph	NaN
Shape of You	NaN
Shivers	NaN
Willow	4.6
Yellow Submarine	4.6
Yesterday	3.9

CPU times: user 4.62 ms, sys: 4.38 ms, total: 9 ms
Wall time: 6.51 ms

```
In [30]: %%time
avg_left_ratings_notnull = """
SELECT Songs.title, avg(Listens.rating) as average from Songs left join List
WHERE Listens.rating is not null GROUP BY Songs.title"""
runSql('Average ratings with left join without null', avg_left_ratings_notnu
```

Average ratings with left join without null

title	average
Evermore	3.7
Hey Jude	3.9
Willow	4.6
Yellow Submarine	4.6
Yesterday	3.9

CPU times: user 4.26 ms, sys: 5.74 ms, total: 10 ms

Wall time: 6.54 ms

In []: