In-class activity - Song recommendation

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(inde
  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 runSqlWithCommit(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.commit()
  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 SQLlite, 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

Query that calculates average ratings of all songs. Only includes songs with Listens

In [6]:
```
qry_join_songs_ratings="""
SELECT Songs.song_id, Songs.artist, Songs.title, AVG(Listens.rating) as avg_rating
FROM songs
JOIN Listens
ON Songs.song_id = Listens.song_id
```

```
GROUP BY Songs.song_id"""
runSql('Calculates average ratings for songs', qry_join_songs_ratings)
```

Calculates average ratings for songs

song_id	artist	title	avg_rating
1	Taylor Swift	Evermore	3.7
2	Taylor Swift	Willow	4.6
6	Beatles	Yesterday	3.9
7	Beatles	Yellow Submarine	4.6
8	Beatles	Hey Jude	3.9

TO DO: 1. Create a Recommendations table as shown in lecture slides. 2. Write a query to produce two song recommendations for Minnie, and insert into the Recommendations table. The recommendations should be the two songs with the highest average rating not listened by Minnie 3. Write a query to retrieve the song title and artist of the recommendations for Minnie.

In [7]:
```
qry_recommendations = """
INSERT INTO Recommendations (user_id, song_id, recommendation_id, recommendation_time
SELECT (SELECT user_id FROM Users WHERE name="Minnie"), song_id, 1, CURRENT_TIMESTAMP
FROM (
    SELECT s.song_id, AVG(l.rating) AS avg_rating
    FROM Songs s
    JOIN Listens l ON s.song_id = l.song_id
    WHERE s.song_id NOT IN (SELECT song_id FROM Listens WHERE user_id = (SELECT user_
    GROUP BY s.song_id
    ORDER BY avg_rating DESC
    LIMIT 2
) ;
"""
runSqlWithCommit('Insert Recommendations for Minnie', qry_recommendations)
```

In [8]:
```
qry_minnie_recommendations = """
SELECT s.title, s.artist
FROM Songs s
JOIN Recommendations r ON s.song_id = r.song_id
WHERE r.user_id = (SELECT user_id FROM Users WHERE name="Minnie");
"""
runSql('Minnie recommendations', qry_minnie_recommendations)
```

Minnie recommendations

title	artist
Yesterday	Beatles
Evermore	Taylor Swift

In [ ]:
```

```

In [ ]:
```

```