BIG DATA MANAGEMENT

POST GRADUATE DIPLOMA IN DATA ENGINEERING

ASSIGNMENT 2

SUBMITTED BY:

NIRAJ BHAGCHANDANI [G23AI2087]

SUBMISSION DATE: 10th December, 2024

DEPARTMENT OF AIDE
INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR

#1 Insert the above into the recommendations table

Code:

```
import sqlite3
import pandas as pd
from IPython.core.display import HTML

# Initialize database connection
db_connection = sqlite3.connect("music_streaming_1.db")  # Replace with your
database name
db_cursor = db_connection.cursor()

# Function to run SQL queries and display results
def execute_sql(description, query):
    try:
        db_cursor.execute(query)
        if db_cursor.description:  # Check if the query returns a result set
            result_df = pd.DataFrame(db_cursor.fetchall(), columns=[col[0]
for col in db_cursor.description])
            display(HTML(f"<b><font
color=Green>{description}</font></b>{result_df.to_html(index=False)}"))
        else:
            print(f"{description}: Query executed successfully.")  # For non-
result queries
        db_connection.commit()  # Commit changes for transactional
consistency
    except Exception as error:
        print(f"Error executing {description}: {error}")

# Drop the existing Recommendations table
drop_table_query = """
DROP TABLE IF EXISTS Recommendations;
"""
execute_sql("Drop Recommendations Table", drop_table_query)

# Recreate the Recommendations table with AUTOINCREMENT for recommendation_id
create_table_query = """
CREATE TABLE IF NOT EXISTS Recommendations (
    recommendation_id INTEGER PRIMARY KEY AUTOINCREMENT,
    recommendation_time TIMESTAMP,
    user_id INTEGER NOT NULL,
    song_id INTEGER NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (song_id) REFERENCES Songs(song_id)
);
"""
execute_sql("Create Recommendations Table", create_table_query)
```

```
# Query to insert recommendations into the Recommendations table
insert_recommendations_query = """
INSERT INTO Recommendations (recommendation_time, user_id, song_id)
SELECT CURRENT_TIMESTAMP, user_id, song_id
FROM (
    WITH song_similarity AS (
        SELECT u1.song_id AS song1, u2.song_id AS song2, COUNT(*) AS
common_users
        FROM Listens u1
        JOIN Listens u2
        ON u1.user_id = u2.user_id AND u1.song_id != u2.song_id
        GROUP BY u1.song_id, u2.song_id
        HAVING COUNT(*) > 1
    )
    SELECT DISTINCT L.user_id, song_similarity.song2 AS song_id
    FROM song_similarity
    JOIN Listens L
    ON L.song_id = song_similarity.song1
    WHERE song_similarity.song2 NOT IN (
        SELECT song_id FROM Listens WHERE Listens.user_id = L.user_id
    )
);
"""

# Execute and insert recommendations
execute_sql("Insert Recommendations into Recommendations Table",
insert_recommendations_query)
```

Output:

```
Drop Recommendations Table: Query executed successfully.
Create Recommendations Table: Query executed successfully.
Insert Recommendations into Recommendations Table: Query executed successfully.
```

Verify Recommendations

recommendation_id	recommendation_time	user_id	song_id
1	2024-12-09 15:40:21	2	1
2	2024-12-09 15:40:21	2	6

#2 Generate the recommendaions for Minnie

```
# Query to find Minnie's user_id
find_minnie_user_id_query = """
SELECT user_id
FROM Users
```

```
WHERE name = 'Minnie';
"""

# Call the function to execute the query
execute_sql("Find Minnie's User ID", find_minnie_user_id_query)

# Query to generate song recommendations for Minnie
generate_minnie_recommendations_query = """
INSERT INTO Recommendations (recommendation_time, user_id, song_id)
SELECT CURRENT_TIMESTAMP, 2, song_id
FROM (
    WITH song_similarity AS (
        SELECT u1.song_id AS song1, u2.song_id AS song2, COUNT(*) AS
common_users
        FROM Listens u1
        JOIN Listens u2
        ON u1.user_id = u2.user_id AND u1.song_id != u2.song_id
        GROUP BY u1.song_id, u2.song_id
        HAVING COUNT(*) > 1
    )
    SELECT DISTINCT song_similarity.song2 AS song_id
    FROM song_similarity
    JOIN Listens L
    ON L.song_id = song_similarity.song1
    WHERE L.user_id = 2
    AND song_similarity.song2 NOT IN (
        SELECT song_id FROM Listens WHERE user_id = 2
    )
);
"""

# Function to execute the query for generating recommendations
execute_sql("Generate Song Recommendations for Minnie",
generate_minnie_recommendations_query)
execute_sql("Verify Recommendations for Minnie", f"""
SELECT *
FROM Recommendations
WHERE user_id = 2;
""")
```

Output:

Find Minnie's User ID

user_id
2

Generate Song Recommendations for Minnie: Query executed successfully.

Verify Recommendations for Minnie

recommendation_id	recommendation_time	user_id	song_id
1	2024-12-09 15:40:21	2	1
2	2024-12-09 15:40:21	2	6
3	2024-12-09 15:52:21	2	1
4	2024-12-09 15:52:21	2	6
5	2024-12-09 15:52:55	2	1
6	2024-12-09 15:52:55	2	6

#3 Re-do the generation of recommendations now on the basis of listen time

```
# Define the query to generate recommendations for Minnie based on listen
time
rec_for_minni = f"""
INSERT INTO Recommendations (recommendation_time, user_id, song_id)
SELECT CURRENT_TIMESTAMP, 2, song_id
FROM (
    WITH song_similarity AS (
        SELECT u1.song_id AS song1, u2.song_id AS song2, COUNT(*) AS
common_users
        FROM Listens u1
        JOIN Listens u2
        ON u1.user_id = u2.user_id AND u1.song_id != u2.song_id
        GROUP BY u1.song_id, u2.song_id
        HAVING COUNT(*) > 1
    )
    SELECT DISTINCT song_similarity.song2 AS song_id
    FROM song_similarity
    JOIN Listens L
    ON L.song_id = song_similarity.song1
    WHERE L.user_id = 2
    AND song_similarity.song2 NOT IN (
        SELECT song_id
        FROM Listens
        WHERE user_id = 2
    )
    ORDER BY L.listen_time DESC  -- Prioritize based on listen time
);
"""
```

```
# Execute the query to generate the recommendations
execute_sql("Generate Recommendations for Minnie Based on Listen Time",
rec_for_minni)
```

Query – 2

```
# Verify the recommendations for Minnie based on listen time
veri_mini_recomm = f"""
SELECT *
FROM Recommendations
WHERE user_id = 2;
"""

execute_sql("Verify Recommendations for Minnie Based on Listen Time",
veri_mini_recomm)

# Fetch song details for Minnie's recommendations based on listen time
mini_song_details = """
SELECT R.recommendation_id, R.user_id, S.title, S.artist, S.genre
FROM Recommendations R
JOIN Songs S
ON R.song_id = S.song_id
WHERE R.user_id = 2;
"""

execute_sql("Song Details for Minnie's Recommendations with Listen Time",
mini_song_details)
```

Verify Recommendations for Minnie Based on Listen Time

recommendation_id	recommendation_time	user_id	song_id
1	2024-12-09 15:40:21	2	1
2	2024-12-09 15:40:21	2	6
3	2024-12-09 15:52:21	2	1
4	2024-12-09 15:52:21	2	6
5	2024-12-09 15:52:55	2	1
6	2024-12-09 15:52:55	2	6
7	2024-12-09 15:54:04	2	1
8	2024-12-09 15:54:04	2	6

Song Details for Minnie's Recommendations with Listen Time

recommendation_id	user_id	title	artist	genre
1	2	Evermore	Taylor Swift	Pop
2	2	Yesterday	Beatles	Classic
3	2	Evermore	Taylor Swift	Pop

recommendation_id	user_id	title	artist	genre
4	2	Yesterday	Beatles	Classic
5	2	Evermore	Taylor Swift	Pop
6	2	Yesterday	Beatles	Classic
7	2	Evermore	Taylor Swift	Pop
8	2	Yesterday	Beatles	Classic

#4 Generate new recommendations

```
# New query to generate recommendations for Minnie based on rating and listen
time
gen_new_rec = f"""
INSERT INTO Recommendations (recommendation_time, user_id, song_id)
SELECT CURRENT_TIMESTAMP, {2}, S.song_id
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 = {2}
)
AND (L.rating >= 4.5 OR L.listen_time IS NOT NULL)
GROUP BY S.song_id
ORDER BY L.rating DESC, L.listen_time DESC;
"""

# Execute the query to generate new recommendations for Minnie
execute_sql("Generate New Recommendations for Minnie Based on Rating and
Listen Time", gen_new_rec)

execute_sql("Verify New Recommendations for Minnie", f"""
SELECT *
FROM Recommendations
WHERE user_id = {2};
""")
```

Output:

Verify New Recommendations for Minnie

recommendation_id	recommendation_time	user_id	song_id
1	2024-12-09 15:40:21	2	1
2	2024-12-09 15:40:21	2	6
3	2024-12-09 15:52:21	2	1
4	2024-12-09 15:52:21	2	6

recommendation_id	recommendation_time	user_id	song_id
5	2024-12-09 15:52:55	2	1
6	2024-12-09 15:52:55	2	6
7	2024-12-09 15:54:04	2	1
8	2024-12-09 15:54:04	2	6
9	2024-12-09 15:56:57	2	1
10	2024-12-09 15:56:57	2	6
11	2024-12-09 15:57:17	2	1
12	2024-12-09 15:57:17	2	6

Query – 2

```
execute_sql("Song Details for New Recommendations", """
SELECT R.recommendation_id, R.user_id, S.title, S.artist, S.genre
FROM Recommendations R
JOIN Songs S
ON R.song_id = S.song_id
WHERE R.user_id = 2;
""")
```

Song Details for New Recommendations

recommendation_id	user_id	title	artist	genre
1	2	Evermore	Taylor Swift	Pop
2	2	Yesterday	Beatles	Classic
3	2	Evermore	Taylor Swift	Pop
4	2	Yesterday	Beatles	Classic
5	2	Evermore	Taylor Swift	Pop
6	2	Yesterday	Beatles	Classic
7	2	Evermore	Taylor Swift	Pop
8	2	Yesterday	Beatles	Classic
9	2	Evermore	Taylor Swift	Pop
10	2	Yesterday	Beatles	Classic
11	2	Evermore	Taylor Swift	Pop
12	2	Yesterday	Beatles	Classic

Query – 3

```
# Genre-based recommendation query for Minnie
grec_query = f"""
SELECT DISTINCT S.song_id, S.title, S.genre
FROM Songs S
```

```
JOIN Listens L
ON S.song_id = L.song_id
WHERE S.genre = (
    SELECT genre
    FROM Songs
    JOIN Listens
    ON Songs.song_id = Listens.song_id
    WHERE user_id = {2}
    GROUP BY genre
    ORDER BY COUNT(*) DESC
    LIMIT 1
)
AND S.song_id NOT IN (
    SELECT song_id
    FROM Listens
    WHERE user_id = {2}
);
"""

# Execute the genre-based recommendation query
execute_sql("Genre-Based Recommendations", grec_query)
```

Output:

Genre-Based Recommendations

song_id	title	genre
6	Yesterday	Classic

Query – 4

```
new_rec_query = f"""
INSERT INTO Recommendations (recommendation_time, user_id, song_id)
SELECT CURRENT_TIMESTAMP, L.user_id, S.song_id
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 = L.user_id
)
AND (L.rating >= 4.5 OR L.listen_time IS NOT NULL)  -- Prioritize high
ratings or recent listens
GROUP BY L.user_id, S.song_id
ORDER BY L.rating DESC, L.listen_time DESC;
"""
execute_sql("Generate New Recommendations for Minnie", new_rec_query)
```

Query – 5

```
execute_sql("Verify New Recommendations", f"""
SELECT *
FROM Recommendations;
""")
```

Output:

```
Generate New Recommendations for Minnie: Query executed successfully.
```

Query – 6

```
execute_sql("Verify New Recommendations", f"""
SELECT R.recommendation_id, S.title, S.artist, S.genre
FROM Recommendations R
JOIN Songs S
ON R.song_id = S.song_id;
""")
```

Output:

Verify New Recommendations

recommendation_id	recommendation_time	user_id	song_id
1	2024-12-09 15:40:21	2	1
2	2024-12-09 15:40:21	2	6
3	2024-12-09 15:52:21	2	1
4	2024-12-09 15:52:21	2	6
5	2024-12-09 15:52:55	2	1
6	2024-12-09 15:52:55	2	6
7	2024-12-09 15:54:04	2	1
8	2024-12-09 15:54:04	2	6
9	2024-12-09 15:56:57	2	1
10	2024-12-09 15:56:57	2	6
11	2024-12-09 15:57:17	2	1
12	2024-12-09 15:57:17	2	6

#5 What are the differences with the static method on #2 above

```
dy_rec_wsong_nm = """
```

```
WITH song_popularity AS (
    SELECT song_id, AVG(listen_time) AS avg_listen_time, COUNT(user_id) AS
total_listens
    FROM Listens
    GROUP BY song_id
    HAVING COUNT(user_id) > 1  -- Songs listened to by more than 1 user
),
user_recommendations AS (
    SELECT DISTINCT u.user_id, sp.song_id, s.title AS song_name
    FROM Users u
    CROSS JOIN song_popularity sp
    JOIN Songs s ON sp.song_id = s.song_id
    WHERE sp.song_id NOT IN (
        SELECT song_id
        FROM Listens
        WHERE user_id = u.user_id
    )
    ORDER BY sp.avg_listen_time DESC, sp.total_listens DESC
)
SELECT user_id, song_id, song_name FROM user_recommendations;
"""
runSql("Dynamic Recommendations with Song Names", dy_rec_wsong_nm)
```

Output:

Dynamic Recommendations with Song Names		
user_id	**song_id**	**song_name**
4	2	Willow
2	1	Evermore
2	6	Yesterday
4	1	Evermore
4	6	Yesterday