**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 |
| 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 |
| 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 |