**SQL Final Project Submission :**

**Name** : Snehal Yerkuntwar (snehalyr)

**LINK to onecompiler** : <https://onecompiler.com/postgresql/43tyqr3bb>

**NOTE** : In myCompiler , The Data Creation and Data Entry Part are common for all sub-tasks. The SQL codes and their explanation for all the (Questions and its Answer) tasks are in the same file. You need to scroll down to arrive at every question and its SQL query along with its explanation. The main query or code is Inactivated by “- -”.

So whenever you have to check the query, just remove “- -” before ‘SELECT /INSERT’ command and verify it on Output prompt, and then reapply “- -” so that it does not overlap on other queries.

You will find comment “ Task Begins from Here” and from there apply as mentioned above.

**Task 4 Project: Movie Rental Analysis System (using Redshift or PostgreSQL)**

**Objective:** Perform advanced analysis on movie rental data using OLAP operations.

The project will include the following tasks:

**Database Creation:**

Create a database named MovieRental.

Create table rental\_data with columns:

MOVIE\_ID (integer), CUSTOMER\_ID (integer),

GENRE (varchar), RENTAL\_DATE (date),

RETURN\_DATE (date),

RENTAL\_FEE (numeric)

**Data Creation:** Insert 10–15 sample rental records.

**OLAP Operations**:

a) Drill Down: Analyze rentals from genre to individual movie level.

b) Rollup: Summarize total rental fees by genre and then overall.

c) Cube: Analyze total rental fees across combinations of genre, rental date, and customer.

d) Slice: Extract rentals only from the ‘Action’ genre.

e) Dice: Extract rentals where GENRE = 'Action' or 'Drama' and RENTAL\_DATE is in the last 3 months

**Main Code: Task wise begins from here**

1. **Database and table creation**

-- Create the database

CREATE DATABASE MovieRental;

-- Use the newly created database

USE MovieRental;

-- Create the rental\_data table

CREATE TABLE rental\_data (

MOVIE\_ID INT,

CUSTOMER\_ID INT,

GENRE VARCHAR(50),

RENTAL\_DATE DATE,

RETURN\_DATE DATE,

RENTAL\_FEE NUMERIC(5, 2)

);

1. **Insert sample data**

INSERT INTO rental\_data (MOVIE\_ID, CUSTOMER\_ID, GENRE, RENTAL\_DATE, RETURN\_DATE, RENTAL\_FEE) VALUES

(1, 101, 'Action', '2025-05-10', '2025-05-15', 5.99),

(2, 102, 'Drama', '2025-05-12', '2025-05-18', 4.50),

(3, 103, 'Comedy', '2025-05-15', '2025-05-20', 3.75),

(4, 101, 'Action', '2025-06-01', '2025-06-05', 6.25),

(5, 104, 'Action', '2025-06-03', '2025-06-08', 5.99),

(6, 102, 'Drama', '2025-06-05', '2025-06-11', 4.50),

(7, 105, 'Sci-Fi', '2025-06-08', '2025-06-14', 7.00),

(8, 101, 'Comedy', '2025-07-01', '2025-07-06', 3.75),

(9, 103, 'Action', '2025-07-03', '2025-07-09', 6.25),

(10, 106, 'Thriller', '2025-07-05', '2025-07-12', 5.50),

(11, 104, 'Drama', '2025-07-08', '2025-07-15', 4.50),

(12, 105, 'Action', '2025-07-10', '2025-07-16', 5.99),

(13, 102, 'Comedy', '2025-07-12', '2025-07-19', 3.75),

(14, 106, 'Sci-Fi', '2025-07-15', '2025-07-22', 7.00),

(15, 101, 'Action', '2025-07-18', '2025-07-25', 6.25);

**OLAP operations:**

a**) Drill down: Analyze rentals from genre to individual movie level**

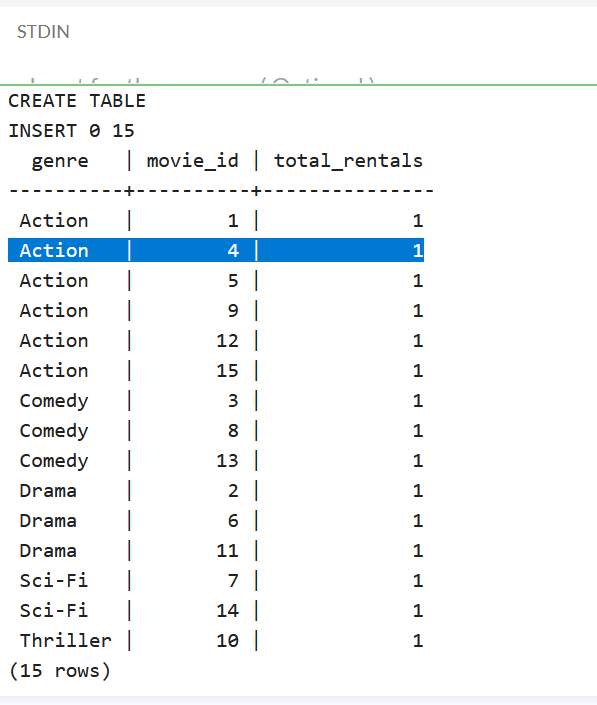
SELECT GENRE, MOVIE\_ID, COUNT(\*) AS total\_rentals

FROM rental\_data

GROUP BY GENRE, MOVIE\_ID

ORDER BY GENRE, MOVIE\_ID;

* This query shows the number of rentals for each movie within each genre.

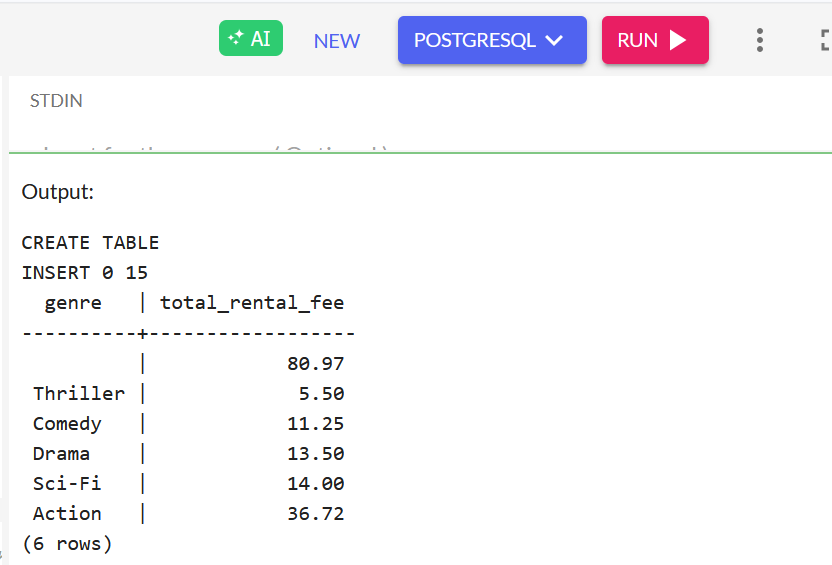


**b) Rollup: Summarize total rental fees by genre and then overall**

SELECT GENRE, SUM(RENTAL\_FEE) AS total\_rental\_fee

FROM rental\_data

GROUP BY ROLLUP(GENRE);



-- This query uses the ROLLUP operator to calculate the sum of rental fees for each genre and then an overall total rental fee.

c) **Cube: Analyze total rental fees across combinations of genre, rental date, and customer**

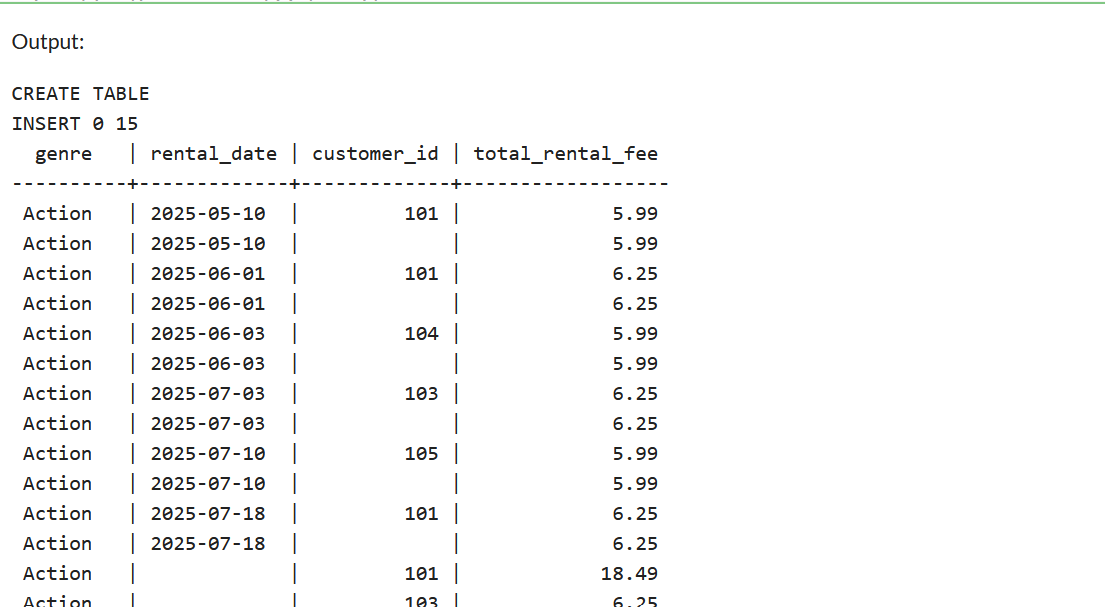
SELECT GENRE, RENTAL\_DATE, CUSTOMER\_ID, SUM(RENTAL\_FEE) AS total\_rental\_fee

FROM rental\_data

GROUP BY CUBE(GENRE, RENTAL\_DATE, CUSTOMER\_ID)

ORDER BY GENRE, RENTAL\_DATE, CUSTOMER\_ID;

* The CUBE operator generates aggregations for all possible combinations of the specified columns (genre, rental date, customer ID).



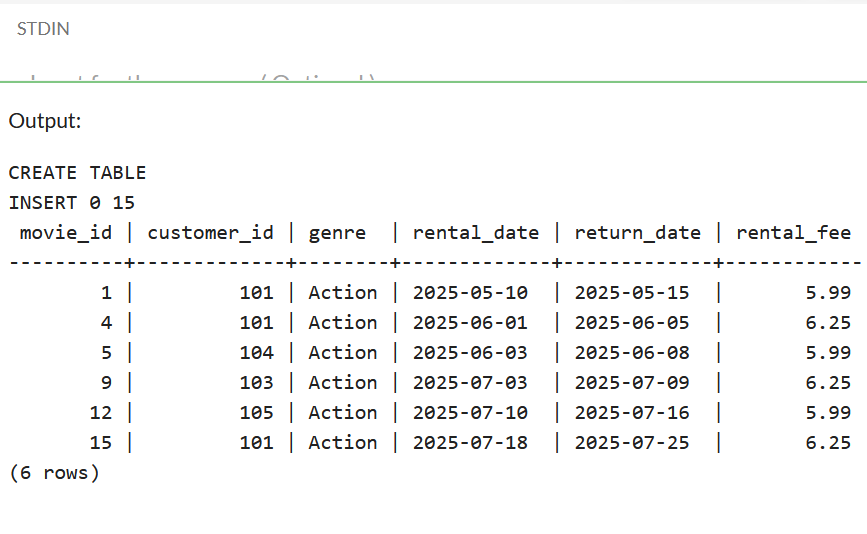
d**) Slice: Extract rentals only from the ‘Action’ genre**

SELECT \*

FROM rental\_data

WHERE GENRE = 'Action';

* This query "slices" the data to show only rentals belonging to the 'Action' genre.



**e) Dice: Extract rentals where GENRE = 'Action' or 'Drama' and RENTAL\_DATE is in the last 3 months**

SELECT \*

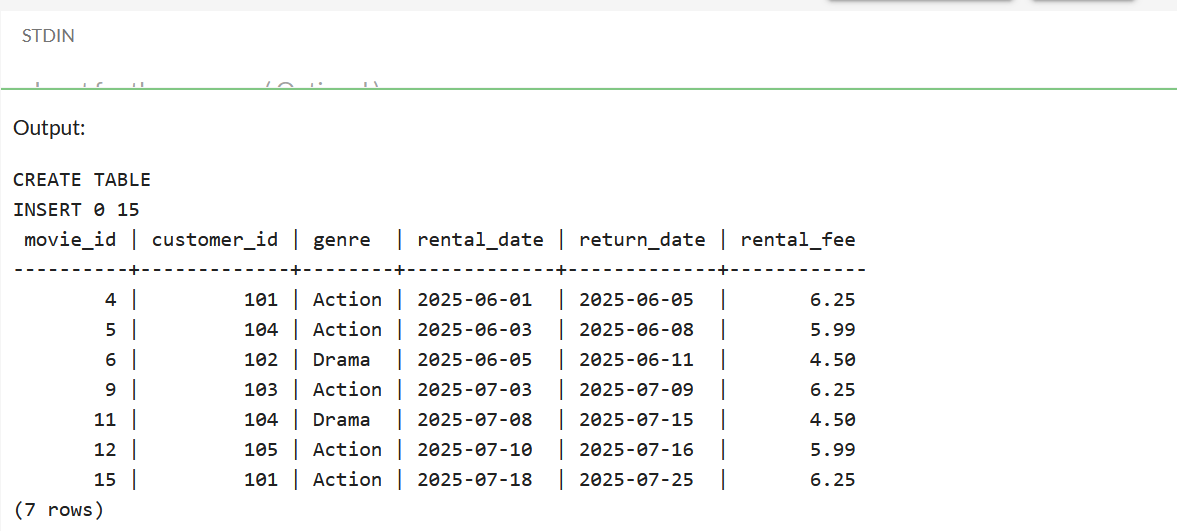
FROM rental\_data

WHERE GENRE IN ('Action', 'Drama')

AND RENTAL\_DATE >= NOW() - INTERVAL '3 months';

---- This query "dices" the data by filtering for rentals from either the 'Action' or 'Drama' genre and those that occurred within the last three months.

--- In PostgreSQL, you can directly subtract time intervals using the INTERVAL keyword. NOW() returns the current date and time.

****