

## Task 4

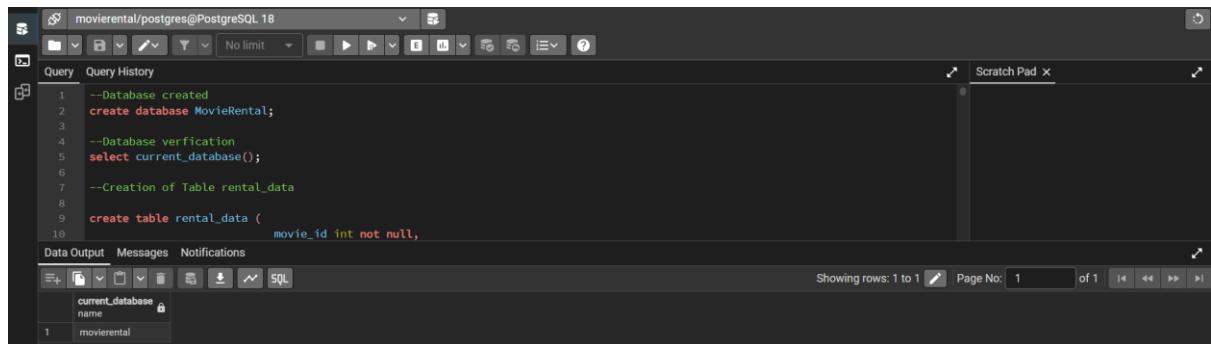
### Project: Movie Rental Analysis System (using Redshift or PostgreSQL)

#### Objective:

Perform advanced analysis on movie rental data using OLAP operations.

GITHUB: [https://github.com/xrahulcrx/Movie\\_Rental\\_Analysis\\_System](https://github.com/xrahulcrx/Movie_Rental_Analysis_System)

#### Database Creation:



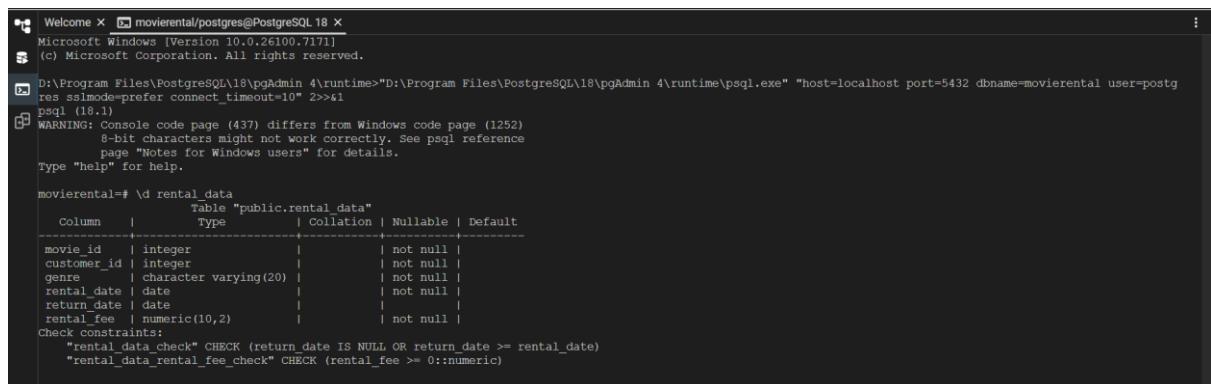
```
--Database created
create database MovieRental;

--Database verification
select current_database();

--Creation of Table rental_data
create table rental_data (
    movie_id int not null,
    customer_id int not null,
    genre character varying(20),
    rental_date date,
    return_date date,
    rental_fee numeric(10,2)
);
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The code above is run, and the results show the database 'MovieRental' has been created and is now the current database. A table 'rental\_data' has been successfully created with the specified columns.

#### Create table rental\_data with columns:



```
Welcome to Microsoft Windows [Version 10.0.26100.7171]
(c) Microsoft Corporation. All rights reserved.

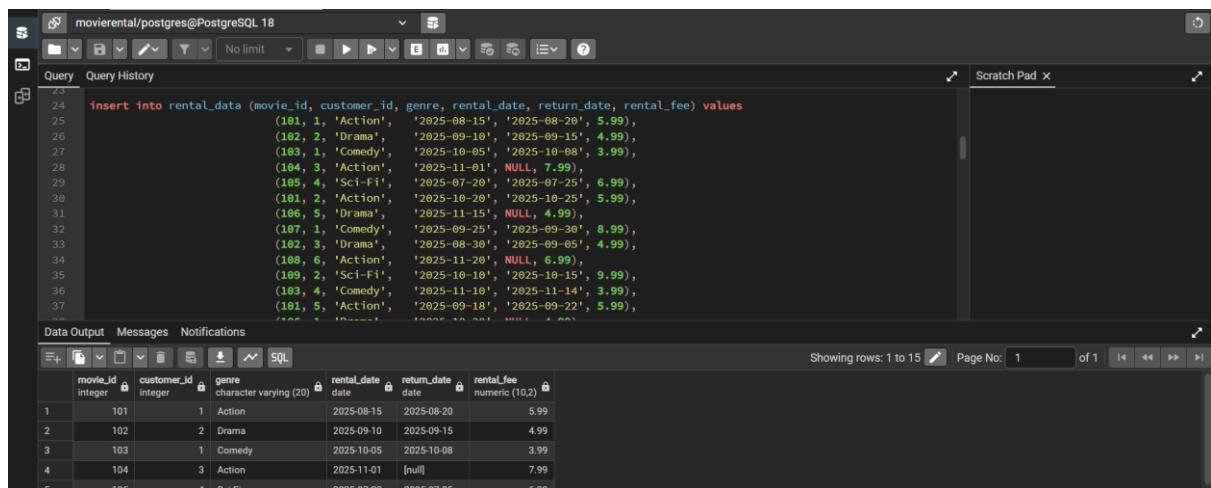
D:\Program Files\PostgreSQL\18\pgAdmin 4\runtime>D:\Program Files\PostgreSQL\18\pgAdmin 4\runtime\psql.exe "host=localhost port=5432 dbname=movierental user=postgres sslmode=prefer connect_timeout=10" 2>>1
psql (18.1)
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See pgsql reference
page "Notes for Windows users" for details.
Type "help" for help.

movierental=# \d rental_data
           Table "public.rental_data"
 Column | Type | Collation | Nullable | Default
--------+-----+-----+-----+-----+
 movie_id | integer | not null |
 customer_id | integer | not null |
 genre | character varying(20) | not null |
 rental_date | date | not null |
 return_date | date | not null |
 rental_fee | numeric(10,2) | not null |
Check constraints:
  "rental_data_check" CHECK (return_date IS NULL OR return_date >= rental_date)
  "rental_data_rental_fee_check" CHECK (rental_fee >= 0::numeric)
```

The screenshot shows a terminal window running psql. It lists the columns and constraints for the 'rental\_data' table. The table has columns for movie\_id, customer\_id, genre, rental\_date, return\_date, and rental\_fee. It includes two check constraints: one for the rental\_date column and another for the rental\_fee column.

#### Data Creation:

Insert 10–15 sample rental records.



```
insert into rental_data (movie_id, customer_id, genre, rental_date, return_date, rental_fee) values
(101, 1, 'Action', '2025-08-15', '2025-08-26', 5.99),
(102, 2, 'Drama', '2025-09-10', '2025-09-15', 4.99),
(103, 1, 'Comedy', '2025-10-05', '2025-10-08', 3.99),
(104, 3, 'Action', '2025-11-01', 'NULL', 7.99),
(105, 4, 'Sci-Fi', '2025-07-20', '2025-07-25', 6.99),
(101, 2, 'Action', '2025-10-20', '2025-10-25', 5.99),
(106, 5, 'Drama', '2025-11-15', 'NULL', 4.99),
(107, 1, 'Comedy', '2025-09-25', '2025-09-30', 8.99),
(102, 3, 'Drama', '2025-08-30', '2025-09-05', 4.99),
(108, 6, 'Action', '2025-11-20', 'NULL', 6.99),
(109, 2, 'Sci-Fi', '2025-10-10', '2025-10-15', 9.99),
(103, 4, 'Comedy', '2025-11-10', '2025-11-14', 3.99),
(101, 5, 'Action', '2025-09-18', '2025-09-22', 5.99),
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The code above inserts 15 sample rental records into the 'rental\_data' table. The table structure is defined in the previous screenshot. The inserted data includes various movie IDs, customer IDs, genres, rental dates, return dates, and rental fees.

## OLAP Operations:

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

movierental/postgres@PostgreSQL 18

Query History

```
50
51 -- a.) Drill Down: Analyze rentals from genre to individual movie level.
52
53 select movie_id, genre, count(*) as rentals, round(sum(rental_fee), 2) as revenue
54 from rental_data
55 group by genre, movie_id
56 order by genre, revenue desc;
57
```

Data Output

	movie_id	genre	rentals	revenue
1	101	Action	3	17.97
2	104	Action	1	7.99
3	108	Action	1	6.99
4	107	Comedy	1	8.99
5	103	Comedy	2	7.98
6	106	Drama	2	9.98
7	102	Drama	2	9.98
8	110	Horror	1	1.99
9	109	Sci-Fi	1	6.99
10	105	Sci-Fi	1	6.99

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

```
-- b) Rollup: Summarize total rental fees by genre and then overall.  
--  
select coalesce(genre, 'All Genres') as genre_list, count(*) as rentals, round(sum(rental_fee), 2) as revenue  
from rental_data  
group by rollup (genre)  
order by grouping(genre), revenue desc nulls last;  
|
```

Data Output Messages Notifications

Showing rows: 1 to 6 Page No: 1 of 1

	genre_list	rentals	revenue
1	Action	5	32.95
2	Drama	4	19.96
3	Sci-Fi	2	16.98
4	Comedy	3	16.97
5	Horror	1	1.99
6	All Genres	15	88.85

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

The screenshot shows the pgAdmin interface with a query editor and a results viewer.

**Query Tab:**

```
-- c) Cube: Analyze total rental fees across combinations of genre, rental date, and customer.
-- v1 : using date

select
    genre,
    date_trunc('month', rental_date)::date AS rental_month,
    customer_id,
    sum(rental_fee) as total_revenue
from rental_data
group by cube (genre, rental_month, customer_id)
order by grouping(genre), grouping(date_trunc('month', rental_date)::date), grouping(customer_id),
        genre, rental_month, customer_id;
```

**Data Output Tab:**

	genre	character varying (20)	rental_month	customer_id	total_revenue
1	Action	2025-08-01	1	5.99	
2	Action	2025-09-01	5	5.99	
3	Action	2025-10-01	2	5.99	
4	Action	2025-11-01	3	7.99	
5	Action	2025-11-01	6	6.99	
6	Comedy	2025-09-01	1	8.99	
7	Comedy	2025-10-01	1	3.99	
8	Comedy	2025-11-01	4	3.99	
9	Drama	2025-08-01	3	4.99	
10	Drama	2025-09-01	2	4.99	

movierental/postgres@PostgreSQL\_18

```

81 -- v2 using month year - optimised
82
83 select coalesce(genre, 'All Genre') as genre_list,
84       coalesce(to_char(date_trunc('Month', rental_date), 'Mon YYYY'), 'All months') as month_year,
85       coalesce(customer_id::text, 'All customer') as customer_id,
86       round(sum(rental_fee),2) as revenue
87 from rental_data
88 group by cube(genre, date_trunc('Month', rental_date), customer_id)
89 order by grouping(genre), grouping(date_trunc('Month', rental_date)), grouping(customer_id),
90         genre, date_trunc('Month', rental_date) nulls first, customer_id;

```

Data Output Messages Notifications

genre_list	month_year	customer_id	revenue
All Genre	Jul 2025	All customer	6.99
All Genre	Aug 2025	All customer	10.98
All Genre	Sep 2025	All customer	19.97
All Genre	Oct 2025	All customer	24.96
All Genre	Nov 2025	All customer	25.95
All Genre	All months	1	23.96
All Genre	All months	2	20.97
All Genre	All months	3	14.97
All Genre	All months	4	10.98
All Genre	All months	5	10.98
All Genre	All months	6	6.99
All Genre	All months	All customer	88.85

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

movierental/postgres@PostgreSQL\_18

```

92 -- d) Slice: Extract rentals only from the 'Action' genre.
93
94 select customer_id, movie_id, rental_date, rental_fee, coalesce(return_date::text, 'Yet to return') as return_date
95 from rental_data
96 where genre = 'Action'
97 order by rental_date desc;
98
99

```

Data Output Messages Notifications

customer_id	movie_id	rental_date	rental_fee	return_date
1	6	108	2025-11-20	6.99 Yet to return
2	3	104	2025-11-01	7.99 Yet to return
3	2	101	2025-10-20	5.99 2025-10-25
4	5	101	2025-09-18	5.99 2025-09-22
5	1	101	2025-08-15	5.99 2025-08-20

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

movierental/postgres@PostgreSQL\_18

```

101
102 -- e) Dice: Extract rentals where GENRE = 'Action' or 'Drama' and RENTAL_DATE is in the last 3 months.
103
104 select movie_id, customer_id, genre, rental_date, coalesce(return_date::text, 'Yet to return') as return_date, rental_fee
105 from rental_data
106 where genre in ('Action', 'Drama') and rental_date >= (current_date - interval '3 months')
107 order by genre, rental_date desc;
108
109
110
111
112
113

```

Data Output Messages Notifications

movie_id	customer_id	genre	rental_date	return_date	rental_fee
1	108	Action	2025-11-20	Yet to return	6.99
2	104	Action	2025-11-01	Yet to return	7.99
3	101	Action	2025-10-20	2025-10-25	5.99
4	101	Action	2025-09-18	2025-09-22	5.99
5	106	Drama	2025-11-15	Yet to return	4.99
6	106	Drama	2025-10-30	Yet to return	4.99
7	102	Drama	2025-09-10	2025-09-15	4.99