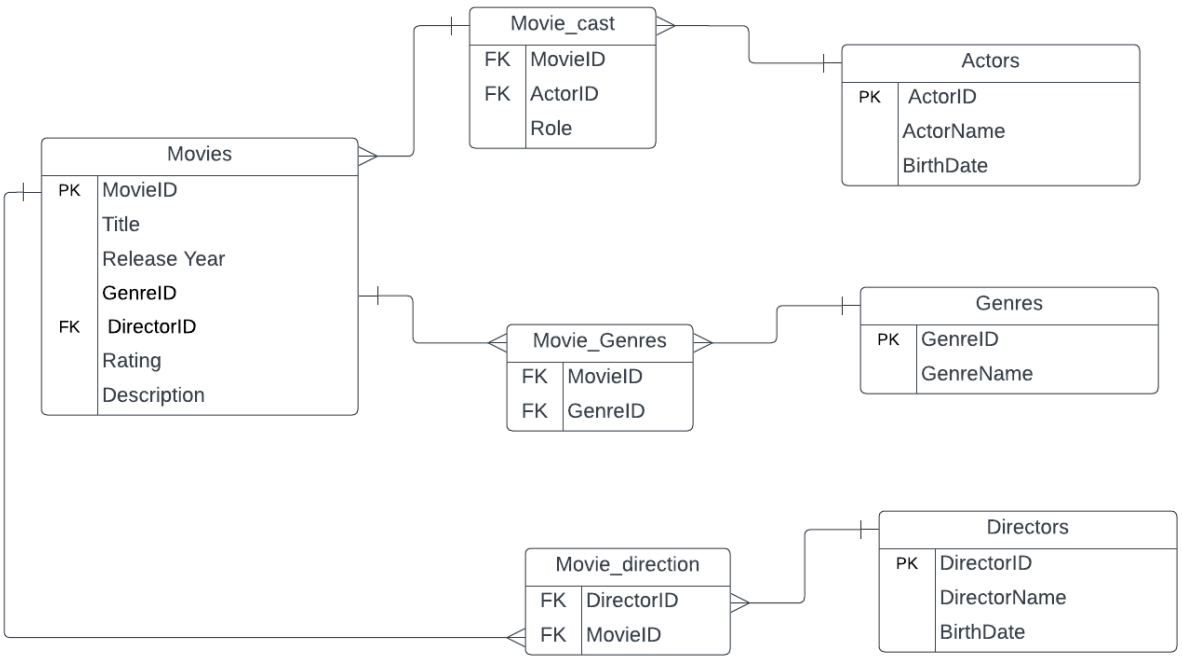
Given the following database tables:

- **Movies**: MovieID, Title, ReleaseYear, GenreID, DirectorID, RaAng, DescripAon
- **Genres**: GenreID, GenreName
- **Directors**: DirectorID, DirectorName, BirthDate
- **Actors**: ActorID, ActorName, BirthDate

1. Design the ER model and show the relaAonships clearly between tables (e.g. one-many relaAonships and many-many relaAonships). **You may need to add more tables (e.g. to cover the many-many rela<onships), so feel free to add tables and keys as needed.**

Answer:

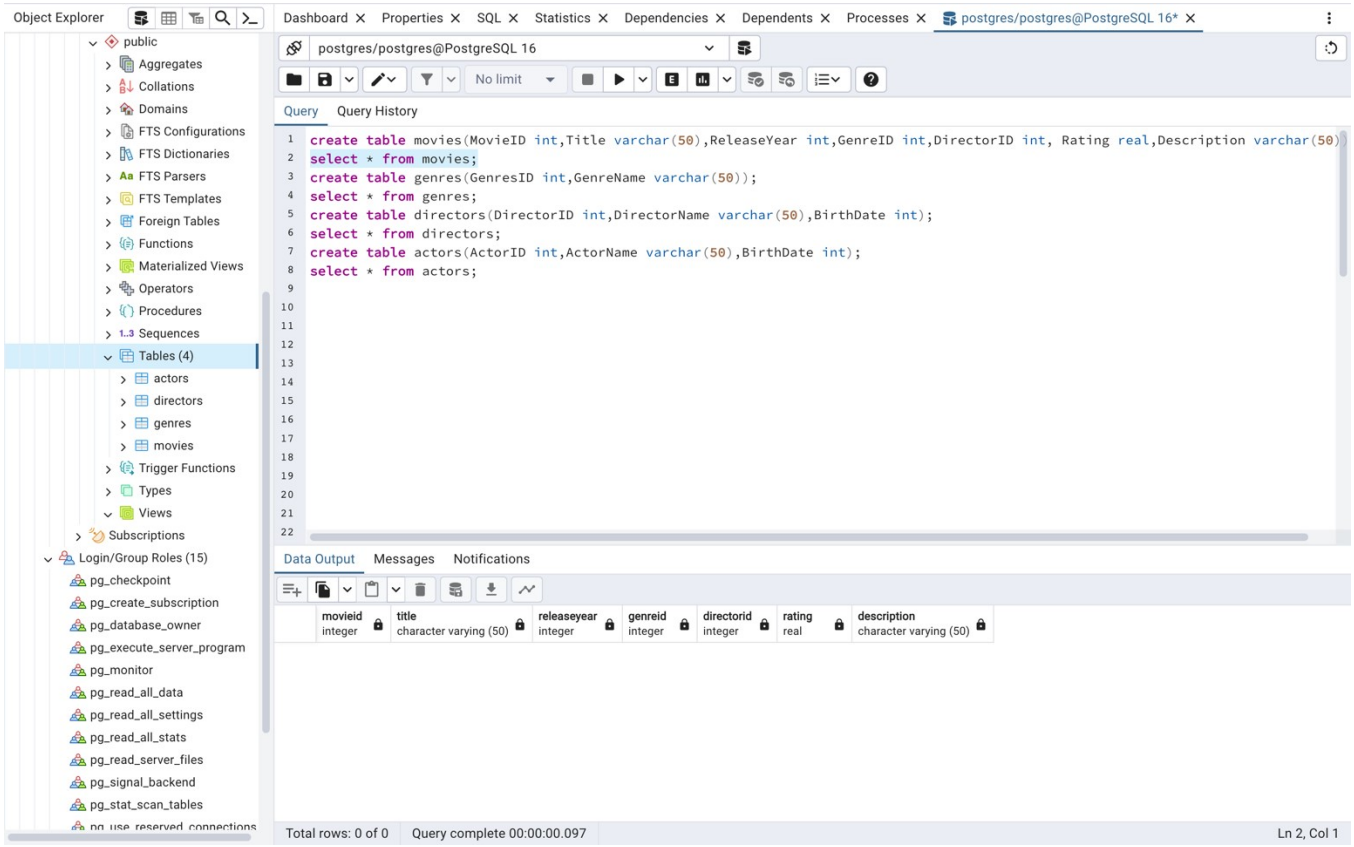


2. Create the database and then create the above tables using SQL.

Answer:

create table movies(MovieID int,Title varchar(50),ReleaseYear int,GenreID int,DirectorID int,

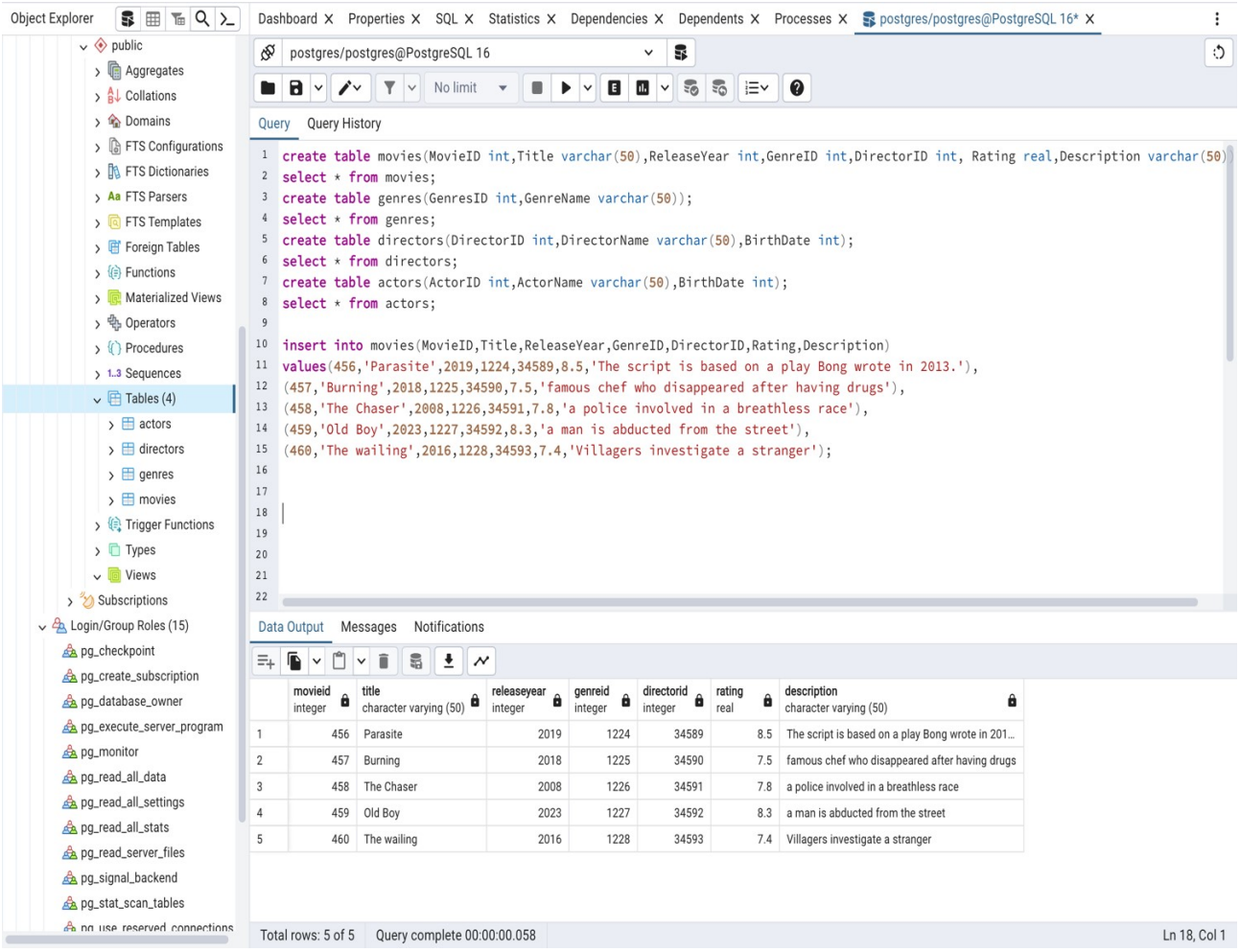
RaAng real,DescripAon varchar(50));
select * from movies;
create table genres(GenresID int,GenreName varchar(50));
select * from genres;
create table directors(DirectorID int,DirectorName varchar(50),BirthDate int);
select * from directors;
create table actors(ActorID int,ActorName varchar(50),BirthDate int);
select * from actors;



3. Populate the tables with real movies data (e.g. your favorite movies) with at least 5 movies. You may do it manually using SQL or using any programming language to import the data (e.g. Python).

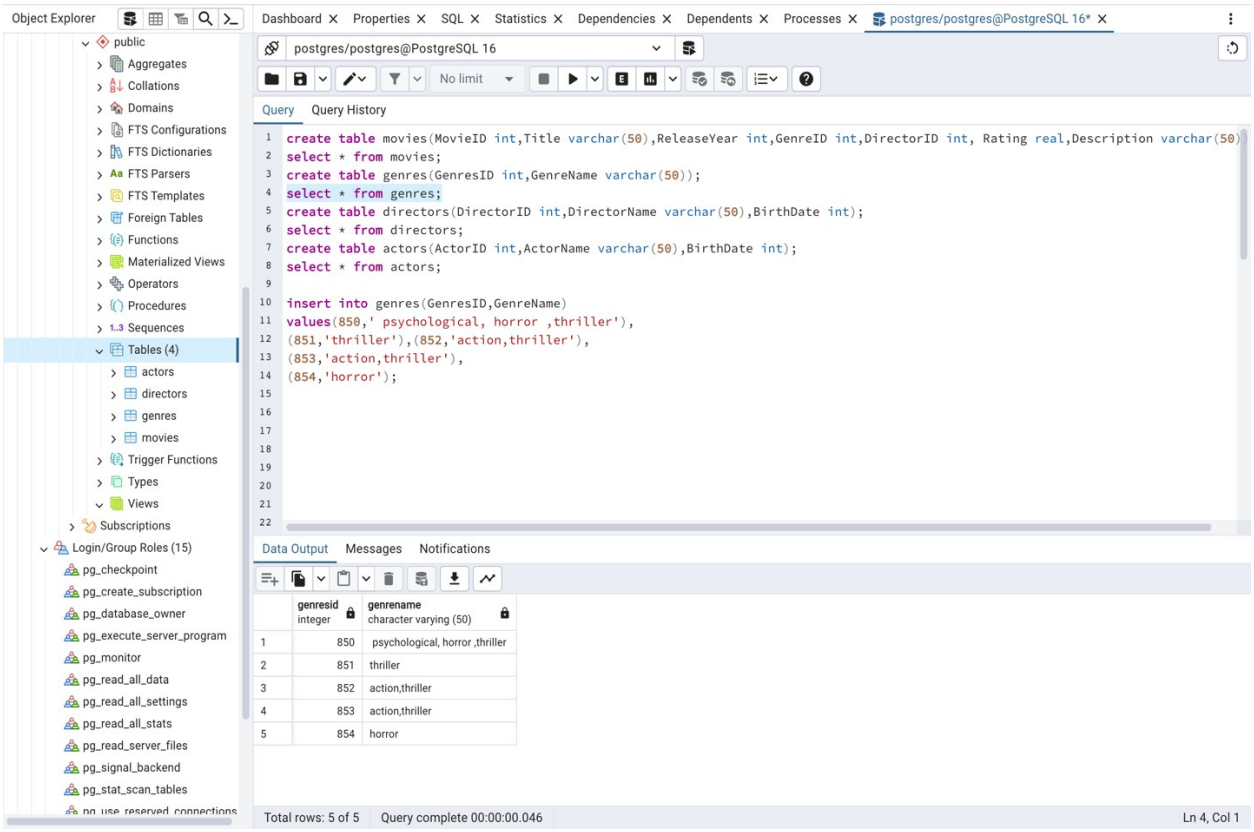
Answer:

a.insert into movies(MovieID,Title,ReleaseYear,GenreID,DirectorID,RaAng,DescripAon) values(456,'Parasite',2019,1224,34589,8.5,'The script is based on a play Bong wrote in 2013.'),
(457,'Burning',2018,1225,34590,7.5,'famous chef who disappeared a[er having drugs'),
(458,'The Chaser',2008,1226,34591,7.8,'a police involved in a breathless race'),
(459,'Old Boy',2023,1227,34592,8.3,'a man is abducted from the street'),
(460,'The wailing',2016,1228,34593,7.4,'Villagers invesAgate a stranger');



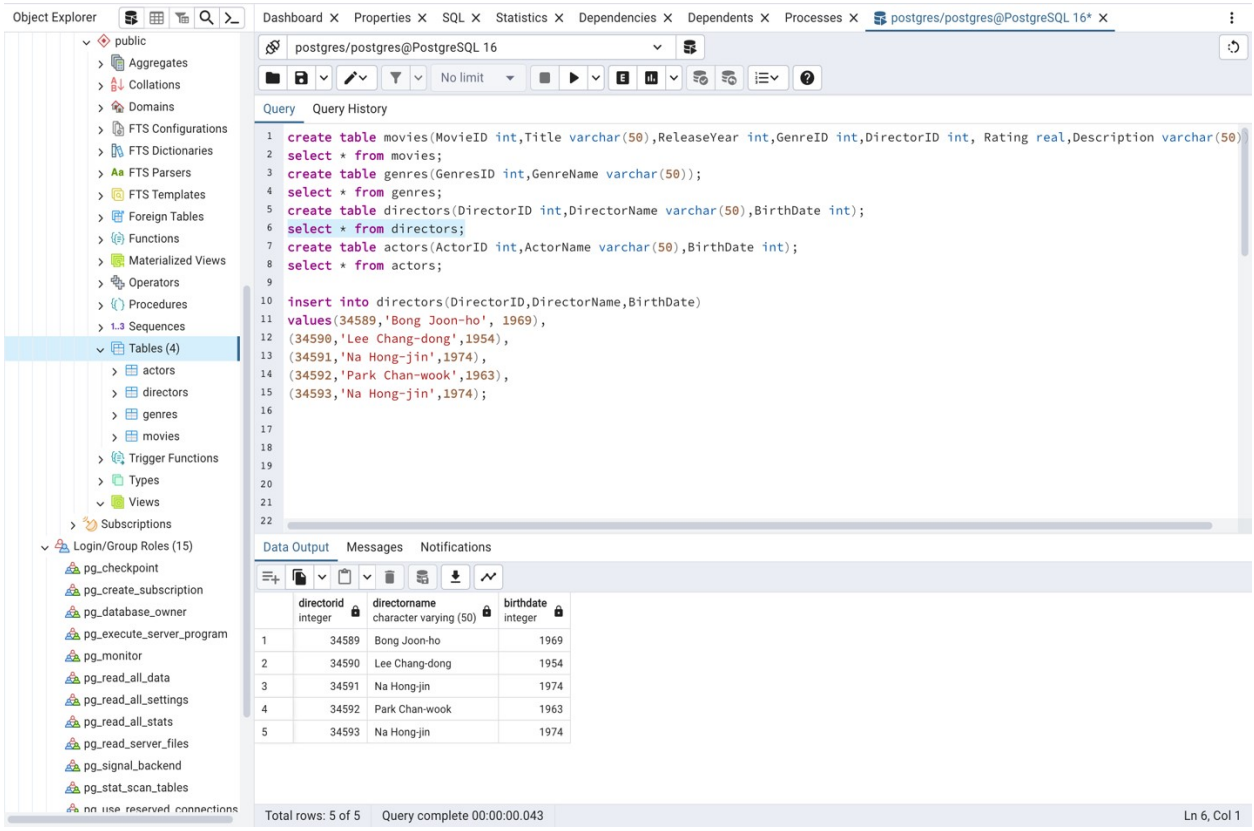
b.

```
insert into genres(GenresID,GenreName)
values(850,' psychological, horror ,thriller'),
(851,'thriller'),(852,'acAon,thriller'),
(853,'acAon,thriller'),
(854,'horror');
```



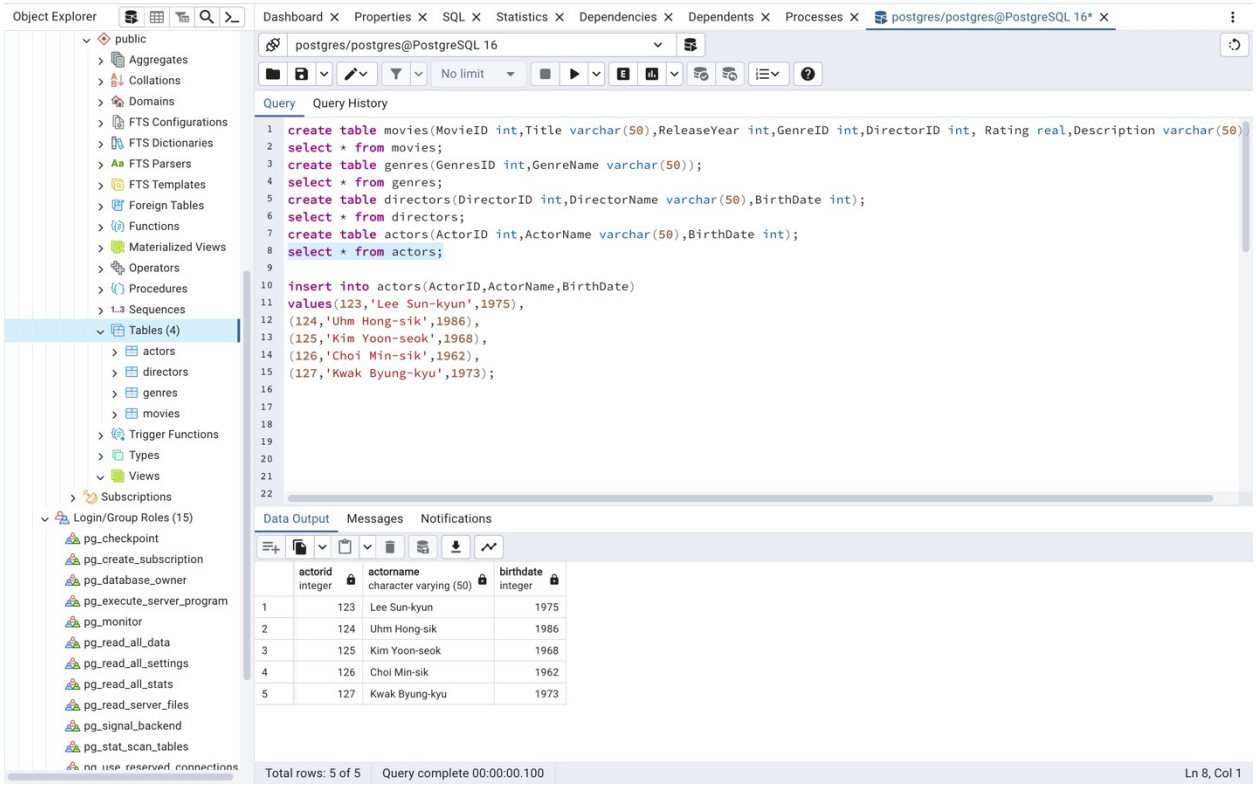
c.

```
insert into directors(DirectorID,DirectorName,BirthDate)
values(34589,'Bong Joon-ho', 1969),
(34590,'Lee Chang-dong',1954),
(34591,'Na Hong-jin',1974),
(34592,'Park Chan-wook',1963),
(34593,'Na Hong-jin',1974);
```



d.

```
insert into actors(ActorID,ActorName,BirthDate)
values(123,'Lee Sun-kyun',1975),
(124,'Uhm Hong-sik',1986),
(125,'Kim Yoon-seok',1968),
(126,'Choi Min-sik',1962),
(127,'Kwak Byung-kyu',1973);
```



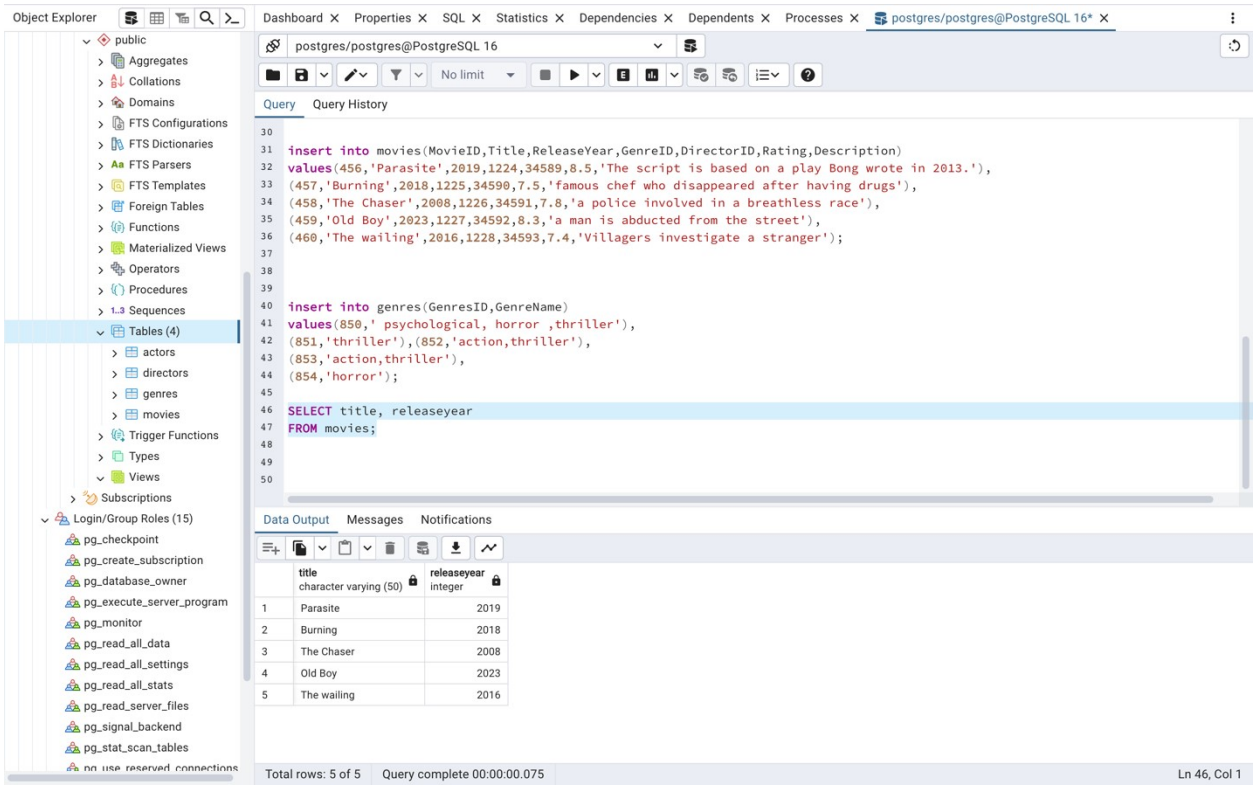
e.

4. Write SQL queries to answer the following:

a. Retrieve all movie Atles and their release years.

Answer:

```
SELECT Atle, releaseyear
FROM movies;
```



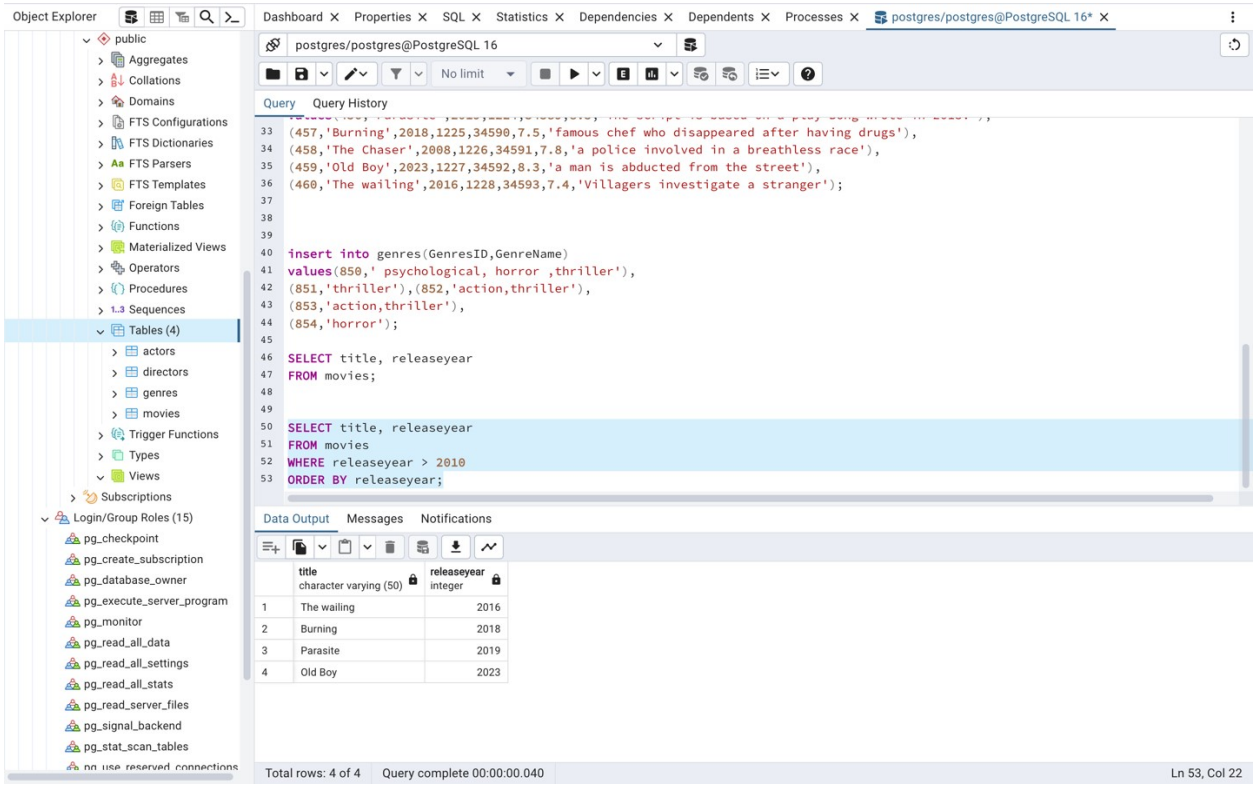
List the movies released a[er 2010, sorted by release year.

Answer:

```
SELECT Atle, releaseyear FROM
movies
WHERE releaseyear > 2010
```

b.

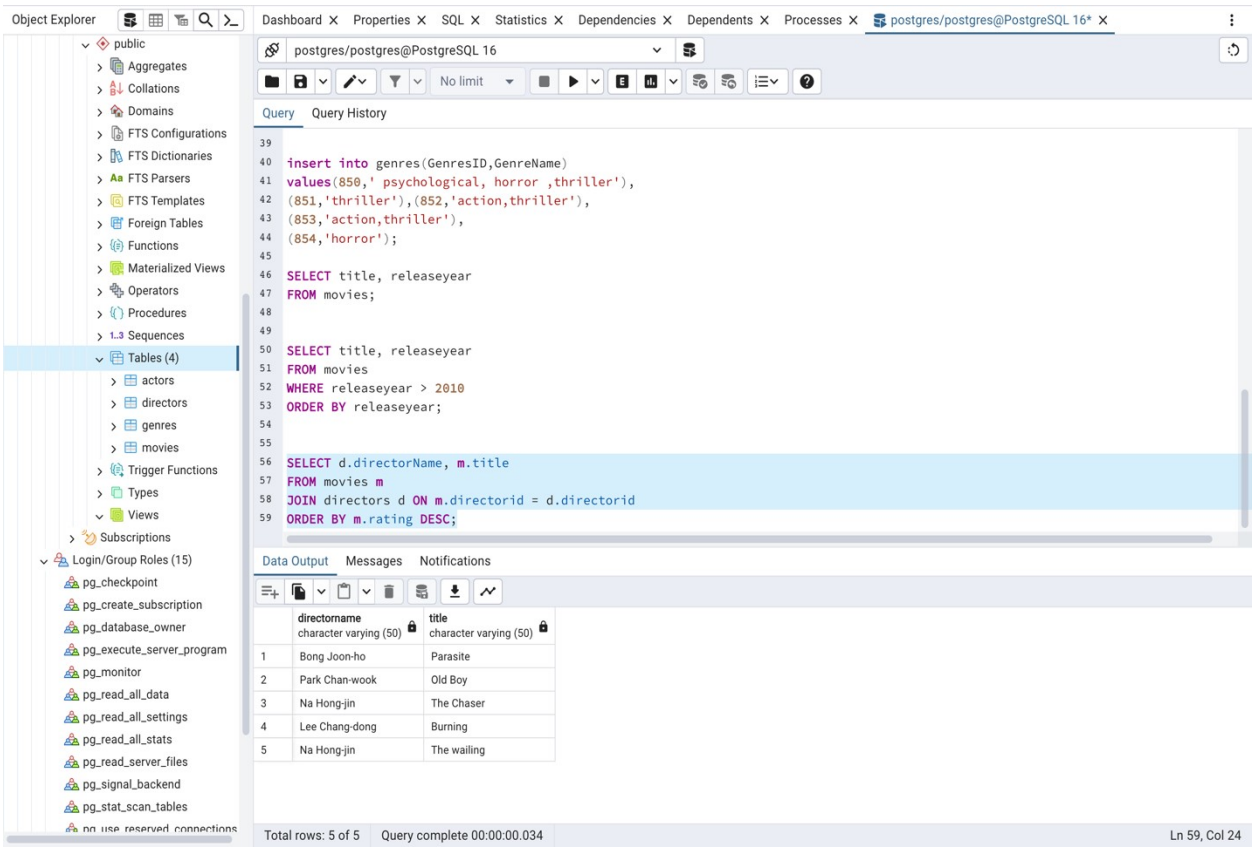
ORDER BY releaseyear;



Retrieve the top-rated movies along with their director.

Answer: SELECT d.directorName, m.Atle
FROM movies m
JOIN directors d ON m.directorid = d.directorid
ORDER BY m.raAng DESC;

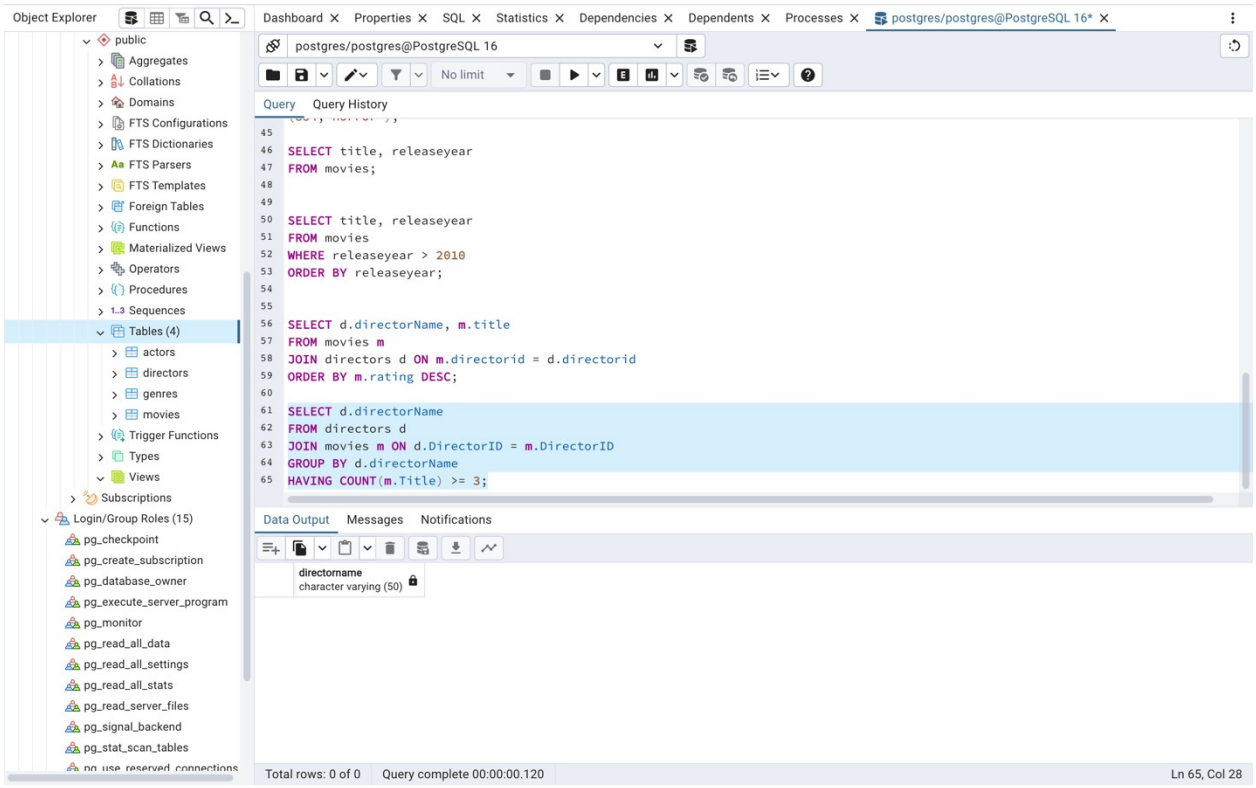
c.



Retrieve the directors who have directed at least three movies.

Answer:
```
SELECT d.directorName
FROM directors d
JOIN movies m ON d.DirectorID = m.DirectorID
GROUP BY d.directorName
HAVING COUNT(m.Title) >= 3;
```

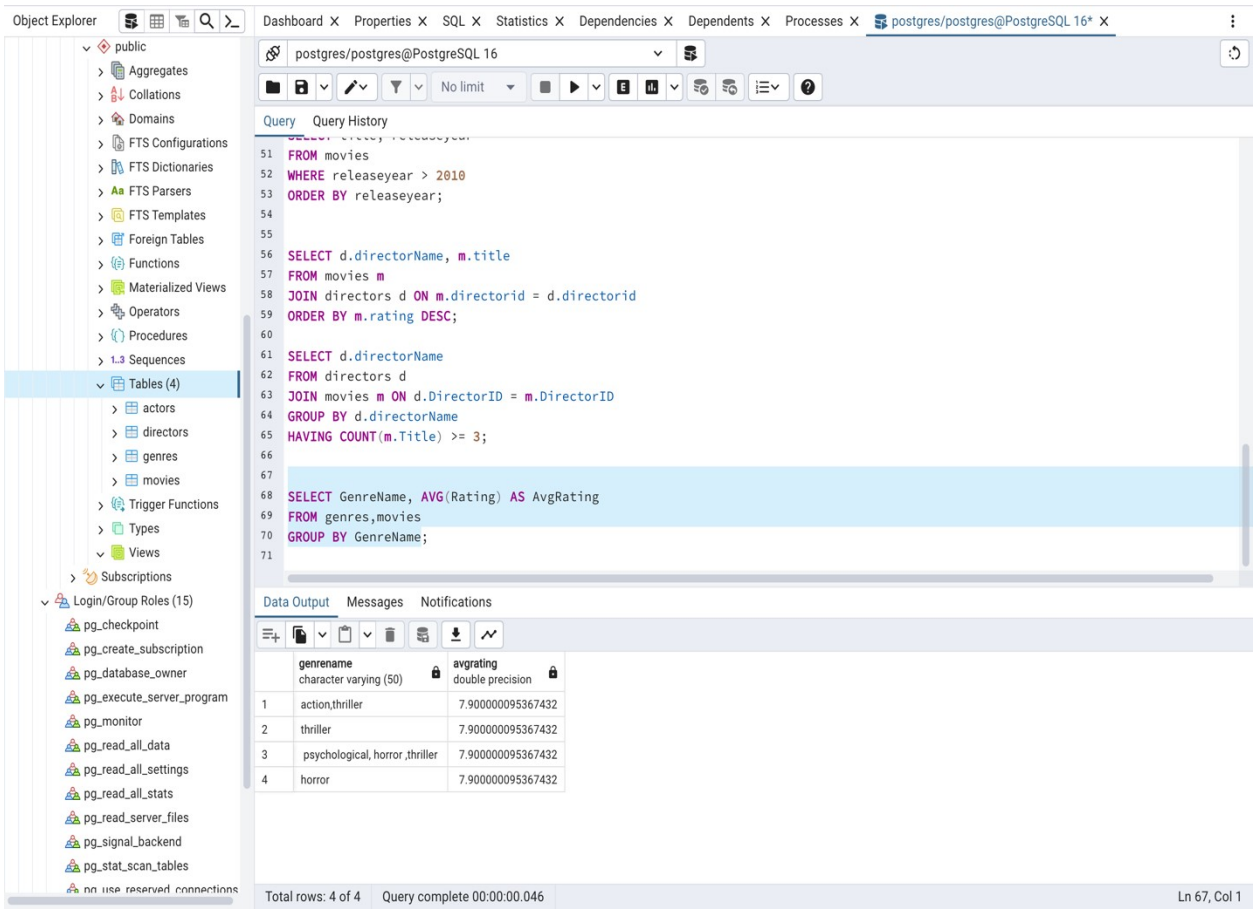
d.



Get the average raAng for each genre.

Answer: SELECT GenreName, AVG(RaAng) AS AvgRaAng
FROM genres,movies
GROUP BY GenreName;

e.


Object Explorer
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
Processes
postgres/postgres@PostgreSQL 16*
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (4)
actors
directors
genres
movies
Trigger Functions
Types
Views
Subscriptions
Login/Group Roles (15)
pg_checkpoint
pg_create_subscription
pg_database_owner
pg_execute_server_program
pg_monitor
pg_read_all_data
pg_read_all_settings
pg_read_all_stats
pg_read_server_files
pg_signal_backend
pg_stat_scan_tables
postgres/postgres@PostgreSQL 16
No limit
Query
Query History
FROM movies
WHERE releaseyear > 2010
ORDER BY releaseyear;
SELECT d.directorName, m.title
FROM movies m
JOIN directors d ON m.directorid = d.directorid
ORDER BY m.rating DESC;
SELECT d.directorName
FROM directors d
JOIN movies m ON d.DirectorID = m.DirectorID
GROUP BY d.directorName
HAVING COUNT(m.Title) >= 3;
SELECT GenreName, AVG(Rating) AS AvgRating
FROM genres,movies
GROUP BY GenreName;
Data Output
Messages
Notifications
genrename
character varying (50)
avgrating
double precision
1 action,thriller 7.900000095367432
2 thriller 7.900000095367432
3 psychological, horror ,thriller 7.900000095367432
4 horror 7.900000095367432
Total rows: 4 of 4
Query complete 00:00:00.046
Ln 67, Col 1

5. Suggest two indexes to be added to the database to enhance performance. One index to enhance an equality query and one index to enhance a range query. Explain clearly why you chose those indexes and show the SQL statements to create them.

Answer:

a. CREATE INDEX idx_Atle ON movies (Title);

Reason: There are several situaAons where you might need to look for a movie straight by its Atle, including when people look up films. The database can find the record that matches the requested movie Atle rapidly.

b. CREATE INDEX idx_movie_raAng ON movies (RaAng);

Reason: For example, to retrieve all movies with a raAng higher than a given number, range queries on the RaAng column can be helpful in locaAng movies inside a given raAng range. Improving the database engine's ability to rapidly discover rows with raAngs falling inside a given range helps speed up such queries by adding an index on RaAng.

You are free to choose the DBMS that you are comfortable with (e.g. SQLite, PostgreSQL, any other relaAonal database).

Submit a PDF report with clear screenshots and SQL scripts for every quesAon above.