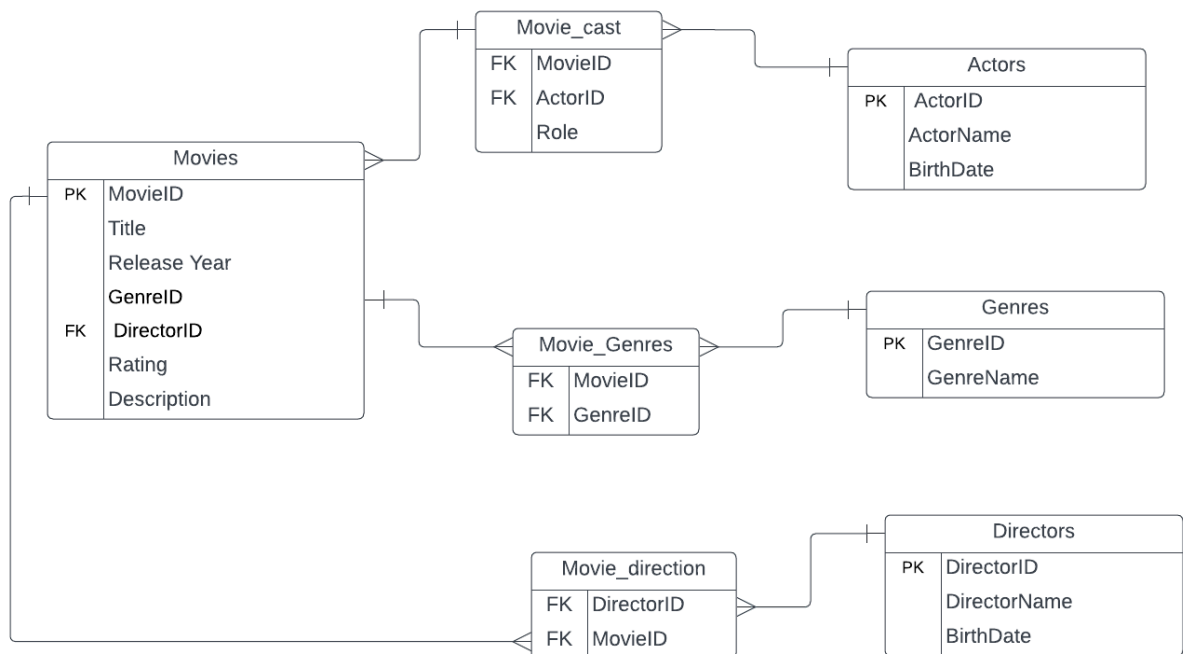


Given the following database tables:

- **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 relationships clearly between tables (e.g. one-many relationships and many-many relationships). **You may need to add more tables (e.g. to cover the many-many relationships), so feel free to add tables and keys as needed.**

Answer:



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

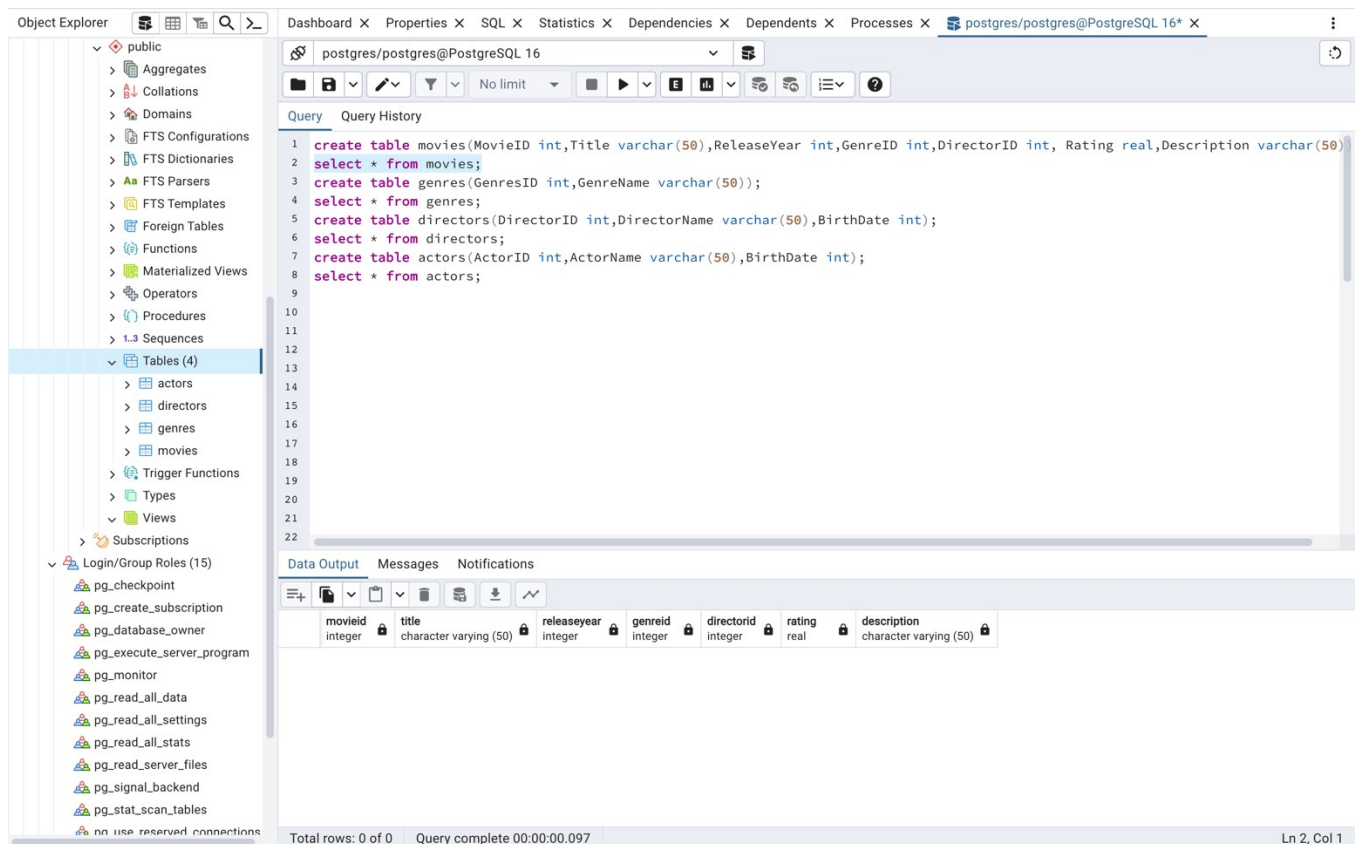
Answer:

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;

```



- 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');

The screenshot shows a PostgreSQL management tool interface. On the left is the 'Object Explorer' tree with 'Tables (4)' selected. The main pane displays a SQL query with the following content:

```

1 create table movies(MovieID int,Title varchar(50),ReleaseYear int,GenreID int,DirectorID int, Rating real,Description varchar(50))
2 select * from movies;
3 create table genres(GenreID int,GenreName varchar(50));
4 select * from genres;
5 create table directors(DirectorID int,DirectorName varchar(50),BirthDate int);
6 select * from directors;
7 create table actors(ActorID int,ActorName varchar(50),BirthDate int);
8 select * from actors;
9
10 insert into movies(MovieID,Title,ReleaseYear,GenreID,DirectorID,Rating,Description)
11 values(456,'Parasite',2019,1224,34589,8.5,'The script is based on a play Bong wrote in 2013.'),
12 (457,'Burning',2018,1225,34590,7.5,'famous chef who disappeared after having drugs'),
13 (458,'The Chaser',2008,1226,34591,7.8,'a police involved in a breathless race'),
14 (459,'Old Boy',2023,1227,34592,8.3,'a man is abducted from the street'),
15 (460,'The wailing',2016,1228,34593,7.4,'Villagers investigate a stranger');
16
17
18
19
20
21
22

```

Below the query editor, the 'Data Output' tab shows the results of the query. The results are displayed in a table with 5 rows and 7 columns:

	movieid integer	title character varying (50)	releaseyear integer	genreid integer	directorid integer	rating real	description character varying (50)
1	456	Parasite	2019	1224	34589	8.5	The script is based on a play Bong wrote in 201...
2	457	Burning	2018	1225	34590	7.5	famous chef who disappeared after having drugs
3	458	The Chaser	2008	1226	34591	7.8	a police involved in a breathless race
4	459	Old Boy	2023	1227	34592	8.3	a man is abducted from the street
5	460	The wailing	2016	1228	34593	7.4	Villagers investigate a stranger

At the bottom of the interface, it shows 'Total rows: 5 of 5' and 'Query complete 00:00:00.058'. The status bar at the bottom right indicates 'Ln 18, Col 1'.

b.

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

The screenshot shows the PostgreSQL pgAdmin interface. On the left, the 'Object Explorer' pane displays the database structure, with 'Tables (4)' expanded under the 'public' schema. The main pane shows a SQL query being executed. The query includes table creation, data selection, and an insert statement. Below the query, the 'Data Output' pane displays the results of the insert operation as a table with 5 rows.

```
1 create table movies(MovieID int,Title varchar(50),ReleaseYear int,GenreID int,DirectorID int, Rating real,Description varchar(50))
2 select * from movies;
3 create table genres(GenresID int,GenreName varchar(50));
4 select * from genres;
5 create table directors(DirectorID int,DirectorName varchar(50),BirthDate int);
6 select * from directors;
7 create table actors(ActorID int,ActorName varchar(50),BirthDate int);
8 select * from actors;
9
10 insert into genres(GenresID,GenreName)
11 values(850,' psychological, horror ,thriller'),
12 (851,'thriller'),(852,'action,thriller'),
13 (853,'action,thriller'),
14 (854,'horror');
```

genresid	genrename
1	850 psychological, horror ,thriller
2	851 thriller
3	852 action,thriller
4	853 action,thriller
5	854 horror

Total rows: 5 of 5 Query complete 00:00:00.0046 Ln 4, Col 1

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);
```

The screenshot shows a PostgreSQL management tool interface. On the left is the 'Object Explorer' pane with a tree view of database objects. The 'Tables (4)' folder is expanded, showing tables: actors, directors, genres, and movies. The 'directors' table is selected. The main pane displays a SQL query in the 'Query' tab. The query creates tables for movies, genres, directors, and actors, and then inserts data into the directors table. Below the query, the 'Data Output' tab shows the results of the insert operation as a table with 5 rows. The status bar at the bottom indicates 'Total rows: 5 of 5' and 'Query complete 00:00:00.043'.

```
1 create table movies(MovieID int,Title varchar(50),ReleaseYear int,GenreID int,DirectorID int, Rating real,Description varchar(50))
2 select * from movies;
3 create table genres(GenresID int,GenreName varchar(50));
4 select * from genres;
5 create table directors(DirectorID int,DirectorName varchar(50),BirthDate int);
6 select * from directors;
7 create table actors(ActorID int,ActorName varchar(50),BirthDate int);
8 select * from actors;
9
10 insert into directors(DirectorID,DirectorName,BirthDate)
11 values(34589,'Bong Joon-ho', 1969),
12 (34590,'Lee Chang-dong',1954),
13 (34591,'Na Hong-jin',1974),
14 (34592,'Park Chan-wook',1963),
15 (34593,'Na Hong-jin',1974);
16
17
18
19
20
21
22
```

directorid	directorname	birthdate
integer	character varying (50)	integer
1	34589 Bong Joon-ho	1969
2	34590 Lee Chang-dong	1954
3	34591 Na Hong-jin	1974
4	34592 Park Chan-wook	1963
5	34593 Na Hong-jin	1974

Total rows: 5 of 5 Query complete 00:00:00.043 Ln 6, Col 1

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);
```

The screenshot shows a PostgreSQL database management interface. On the left, the 'Object Explorer' pane displays the database structure, including 'public' schema, 'Aggregates', 'Collations', 'Domains', 'FTS Configurations', 'FTS Dictionaries', 'FTS Parsers', 'FTS Templates', 'Foreign Tables', 'Functions', 'Materialized Views', 'Operators', 'Procedures', 'Sequences', 'Tables (4)', 'Trigger Functions', 'Types', 'Views', 'Subscriptions', and 'Login/Group Roles (15)'. The 'Tables (4)' folder is expanded, showing 'actors', 'directors', 'genres', and 'movies'. The main pane displays a SQL query and its results. The query is as follows:

```
1 create table movies(MovieID int,Title varchar(50),ReleaseYear int,GenreID int,DirectorID int, Rating real,Description varchar(50));
2 select * from movies;
3 create table genres(GenresID int,GenreName varchar(50));
4 select * from genres;
5 create table directors(DirectorID int,DirectorName varchar(50),BirthDate int);
6 select * from directors;
7 create table actors(ActorID int,ActorName varchar(50),BirthDate int);
8 select * from actors;
9
10 insert into actors(ActorID,ActorName,BirthDate)
11 values(123,'Lee Sun-kyun',1975),
12 (124,'Uhm Hong-sik',1986),
13 (125,'Kim Yoon-seok',1968),
14 (126,'Choi Min-sik',1962),
15 (127,'Kwak Byung-kyu',1973);
16
17
18
19
20
21
22
```

The 'Data Output' pane shows the results of the query, displaying a table with 5 rows and 3 columns: 'actorid', 'actormname', and 'birthdate'.

actorid	actormname	birthdate
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

The status bar at the bottom indicates 'Total rows: 5 of 5' and 'Query complete 00:00:00.100'. The current position is 'Ln 8, Col 1'.

e.

4. Write SQL queries to answer the following:

a. Retrieve all movie Atles and their release years.

Answer:

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

The screenshot shows a PostgreSQL database interface. On the left is the 'Object Explorer' showing a tree of database objects. The 'Tables (4)' folder is expanded, showing 'actors', 'directors', 'genres', and 'movies'. The 'movies' table is selected. The main window displays a SQL query in the 'Query' tab. The query is as follows:

```
30
31 insert into movies(MovieID,Title,ReleaseYear,GenreID,DirectorID,Rating,Description)
32 values(456,'Parasite',2019,1224,34589,8.5,'The script is based on a play Bong wrote in 2013.'),
33 (457,'Burning',2018,1225,34598,7.5,'famous chef who disappeared after having drugs'),
34 (458,'The Chaser',2008,1226,34591,7.8,'a police involved in a breathless race'),
35 (459,'Old Boy',2023,1227,34592,8.3,'a man is abducted from the street'),
36 (460,'The wailing',2016,1228,34593,7.4,'Villagers investigate a stranger');
37
38
39
40 insert into genres(GenresID,GenreName)
41 values(850,' psychological , horror ,thriller'),
42 (851,'thriller'),(852,'action,thriller'),
43 (853,'action,thriller'),
44 (854,'horror');
45
46 SELECT title, releaseyear
47 FROM movies;
48
49
50
```

Below the query, the 'Data Output' tab shows the results of the query. The results are displayed in a table with two columns: 'title' and 'releaseyear'. The table contains five rows of data:

title	releaseyear
Parasite	2019
Burning	2018
The Chaser	2008
Old Boy	2023
The wailing	2016

The status bar at the bottom indicates 'Total rows: 5 of 5' and 'Query complete 00:00:00.075'.

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

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


b.

ORDER BY releaseyear;

The screenshot shows a PostgreSQL database interface with the following components:

- Object Explorer:** A tree view on the left showing the database structure. The 'Tables (4)' folder is expanded, showing tables: actors, directors, genres, and movies. The 'movies' table is selected.
- Query Editor:** A central pane showing a SQL query. The query is as follows:

```
33 (457,'Burning',2018,1225,34590,7.5,'famous chef who disappeared after having drugs'),  
34 (458,'The Chaser',2008,1226,34591,7.8,'a police involved in a breathless race'),  
35 (459,'Old Boy',2023,1227,34592,8.3,'a man is abducted from the street'),  
36 (460,'The wailing',2016,1228,34593,7.4,'Villagers investigate a stranger');  
37  
38  
39  
40 insert into genres(GenresID,GenreName)  
41 values(850,' psychological, horror ,thriller'),  
42 (851,'thriller'),(852,'action,thriller'),  
43 (853,'action,thriller'),  
44 (854,'horror');  
45  
46 SELECT title, releaseyear  
47 FROM movies;  
48  
49  
50 SELECT title, releaseyear  
51 FROM movies  
52 WHERE releaseyear > 2010  
53 ORDER BY releaseyear;
```
- Data Output:** A table at the bottom showing the results of the query. It has two columns: 'title' (character varying (50)) and 'releaseyear' (integer). The results are:

title	releaseyear
The wailing	2016
Burning	2018
Parasite	2019
Old Boy	2023
- Status Bar:** At the bottom, it shows 'Total rows: 4 of 4' and 'Query complete 00:00:00.040'.

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.

The screenshot shows a PostgreSQL management tool interface. On the left is the 'Object Explorer' tree with 'Tables (4)' expanded, showing 'actors', 'directors', 'genres', and 'movies'. The 'Query' editor on the right contains the following SQL code:

```
39
40 insert into genres(GenresID,GenreName)
41 values(850,'psychological, horror ,thriller'),
42 (851,'thriller'),(852,'action,thriller'),
43 (853,'action,thriller'),
44 (854,'horror');
45
46 SELECT title, releaseyear
47 FROM movies;
48
49
50 SELECT title, releaseyear
51 FROM movies
52 WHERE releaseyear > 2010
53 ORDER BY releaseyear;
54
55
56 SELECT d.directorName, m.title
57 FROM movies m
58 JOIN directors d ON m.directorid = d.directorid
59 ORDER BY m.rating DESC;
```

Below the query editor is the 'Data Output' tab, which displays the results of the last query in a table:

	directorname	title
1	Bong Joon-ho	Parasite
2	Park Chan-wook	Old Boy
3	Na Hong-jin	The Chaser
4	Lee Chang-dong	Burning
5	Na Hong-jin	The wailing

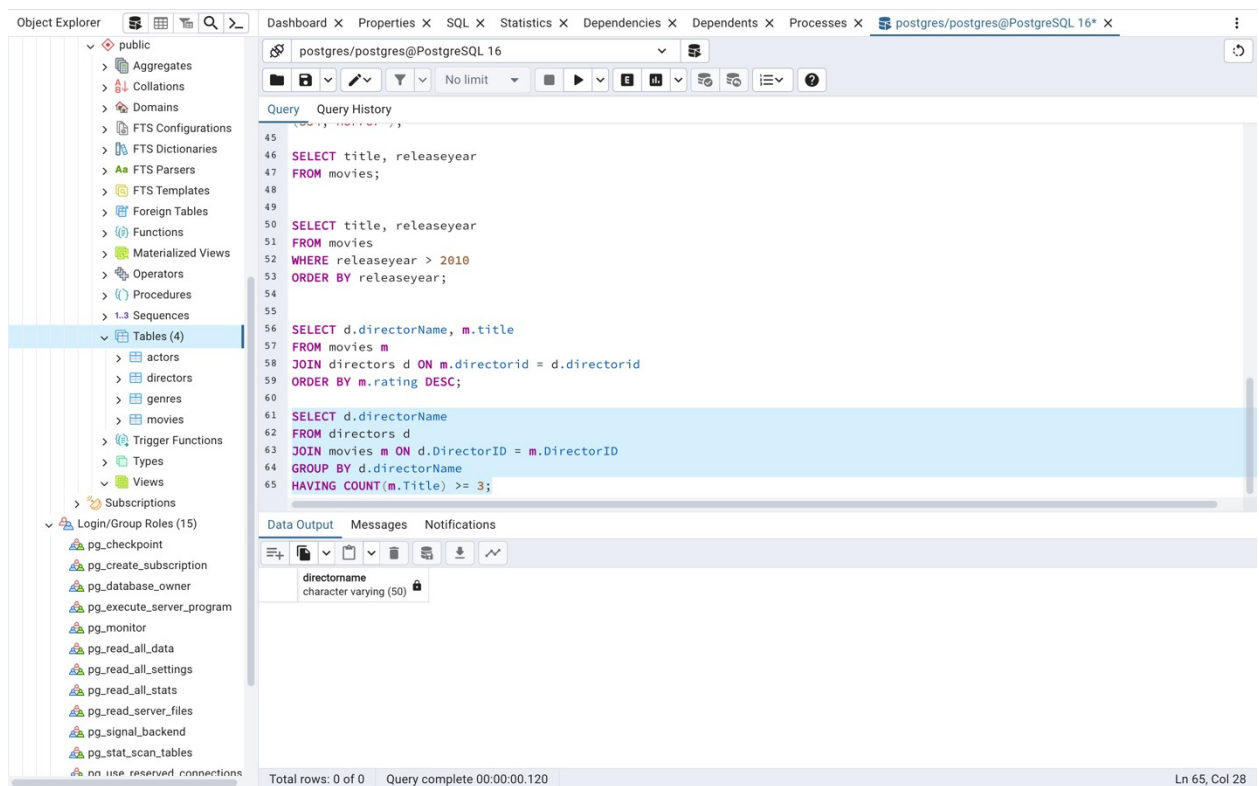
The status bar at the bottom indicates 'Total rows: 5 of 5' and 'Query complete 00:00:00.034'.

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.

The screenshot shows a PostgreSQL IDE interface. On the left is the 'Object Explorer' pane with a tree view of the database schema. The 'public' schema is expanded, showing 'Tables (4)' which includes 'actors', 'directors', 'genres', and 'movies'. The main editor pane displays a SQL query with line numbers 51 to 71. The query consists of two parts: a query to find movies by director and a query to calculate the average rating by genre. The 'Data Output' pane at the bottom shows the results of the second query, a table with 4 rows and 2 columns: 'genrename' and 'avgrating'. The status bar at the bottom indicates 'Total rows: 4 of 4' and 'Query complete 00:00:00.046'.

Object Explorer

- 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
 - no use reserved connections

Dashboard X Properties X SQL X Statistics X Dependencies X Dependents X Processes X postgres/postgres@PostgreSQL 16* X

postgres/postgres@PostgreSQL 16

Query Query History

```
51 FROM movies
52 WHERE releaseyear > 2010
53 ORDER BY releaseyear;
54
55
56 SELECT d.directorName, m.title
57 FROM movies m
58 JOIN directors d ON m.directorid = d.directorid
59 ORDER BY m.rating DESC;
60
61 SELECT d.directorName
62 FROM directors d
63 JOIN movies m ON d.DirectorID = m.DirectorID
64 GROUP BY d.directorName
65 HAVING COUNT(m.Title) >= 3;
66
67
68 SELECT GenreName, AVG(Rating) AS AvgRating
69 FROM genres,movies
70 GROUP BY GenreName;
71
```

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 situations where you might need to look for a movie straight by its title, including when people look up films. The database can find the record that matches the requested movie title rapidly.

- b. `CREATE INDEX idx_movie_raAng ON movies (RaAng);`

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

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

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