SQUADCAST -- Interview Question: Data Analysis and Manipulation

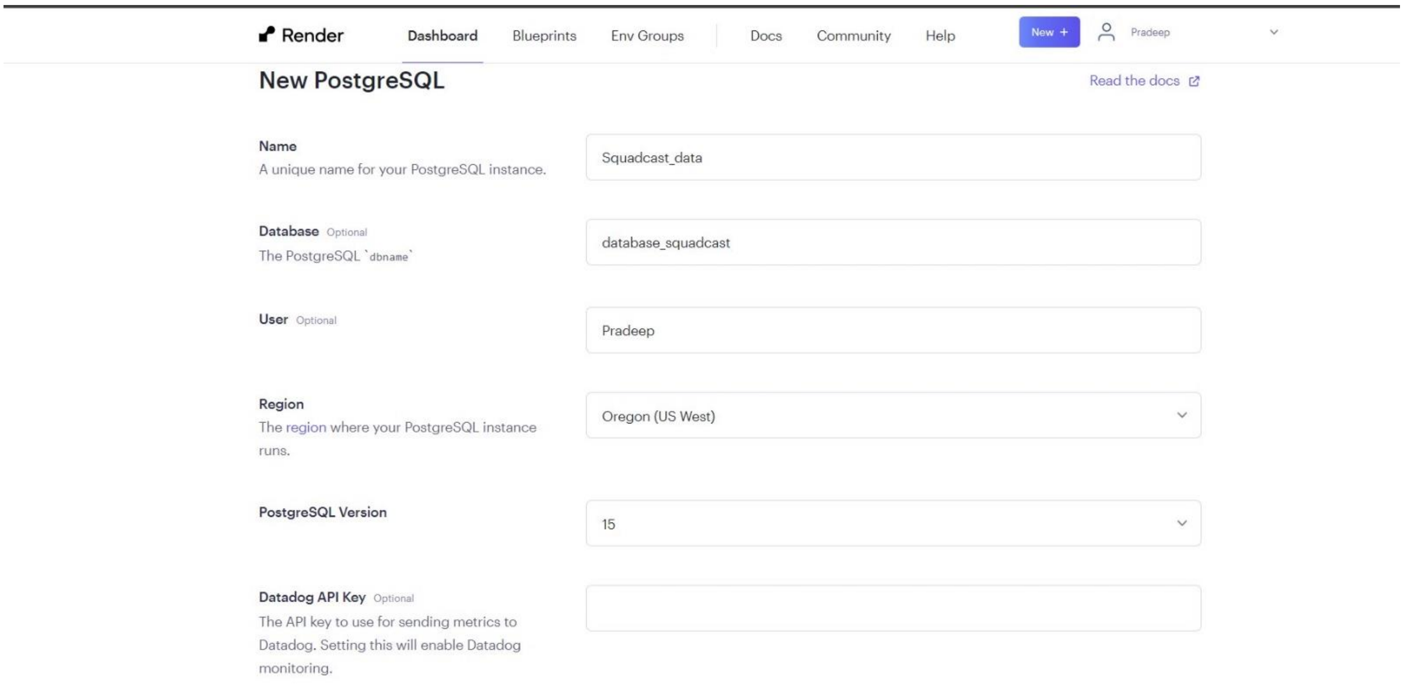
You are provided with CSV files containing movie and rating data. The task is to perform various insights and analyses on this dataset. The process involves:

Task Overview:

1. **Data Import:** - Set up a PostgreSQL database. (You can setup a free PostgreSQL instance from Render) - Create tables to store movie and rating data from the CSV files (You can download the CSV files from here) - Import the CSV data into the respective tables in the PostgreSQL database.

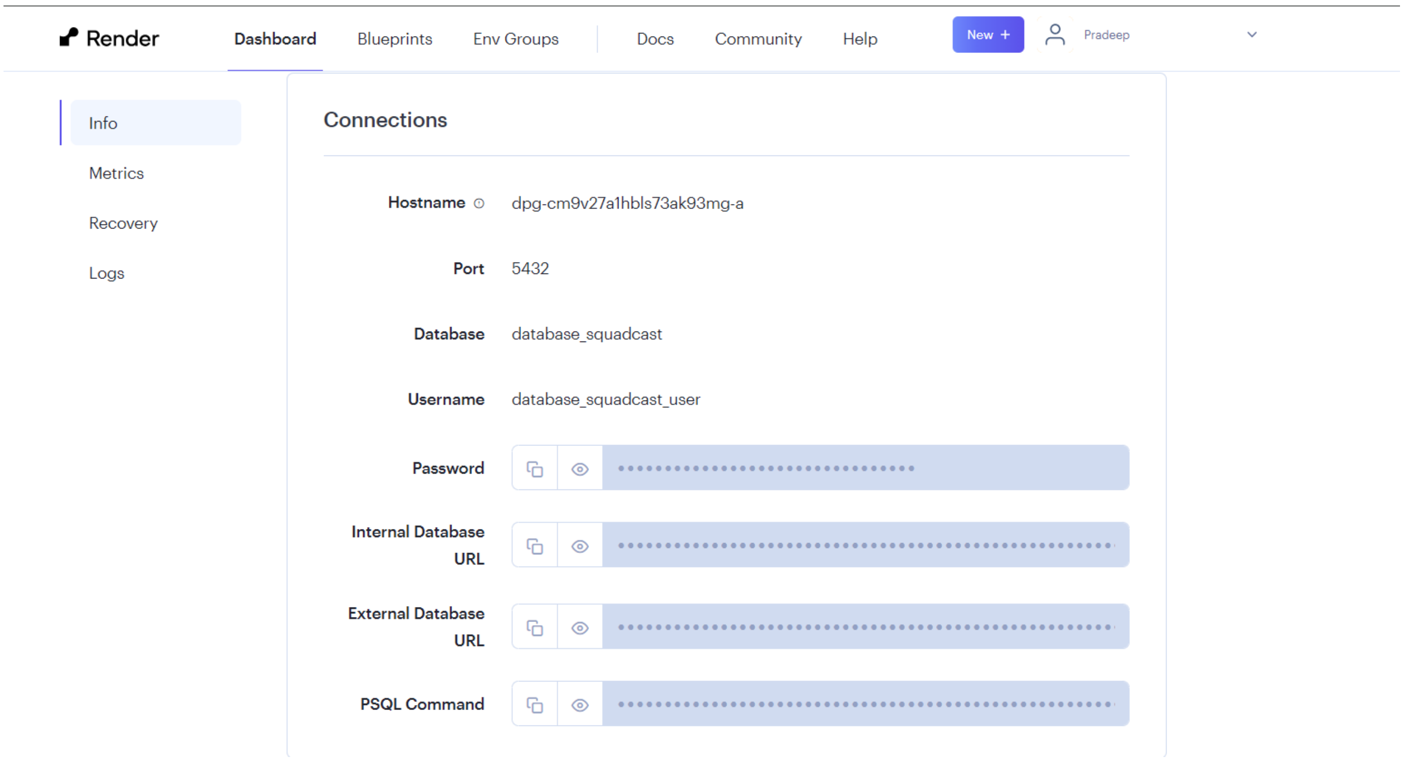
Step 1: Set Up a PostgreSQL Database

Go to Render: Create a hosting Server/ Database instance.



New PostgreSQL

Name A unique name for your PostgreSQL instance.	Squadcast_data
Database Optional The PostgreSQL `dbname`	database_squadcast
User Optional	Pradeep
Region The region where your PostgreSQL instance runs.	Oregon (US West)
PostgreSQL Version	15
Datadog API Key Optional The API key to use for sending metrics to Datadog. Setting this will enable Datadog monitoring.	



Connections

Hostname	dpg-cm9v27a1hbls73ak93mg-a
Port	5432
Database	database_squadcast
Username	database_squadcast_user
Password	••••••••••••••••••••••••••••••••
Internal Database URL	••
External Database URL	••
PSQL Command	••

Step 2: Now After Creating a database hosting server, we want a management tool so that we can Connect our Database instance which is on render and perform further Tasks.

I have Downloaded **pgAdmin** Application to connect our database, one popular tool for managing PostgreSQL databases.

Using pgAdmin:

Download and install pgAdmin:

Download and install pgAdmin from the official website:

1. Open pgAdmin:

   Open pgAdmin on your computer.

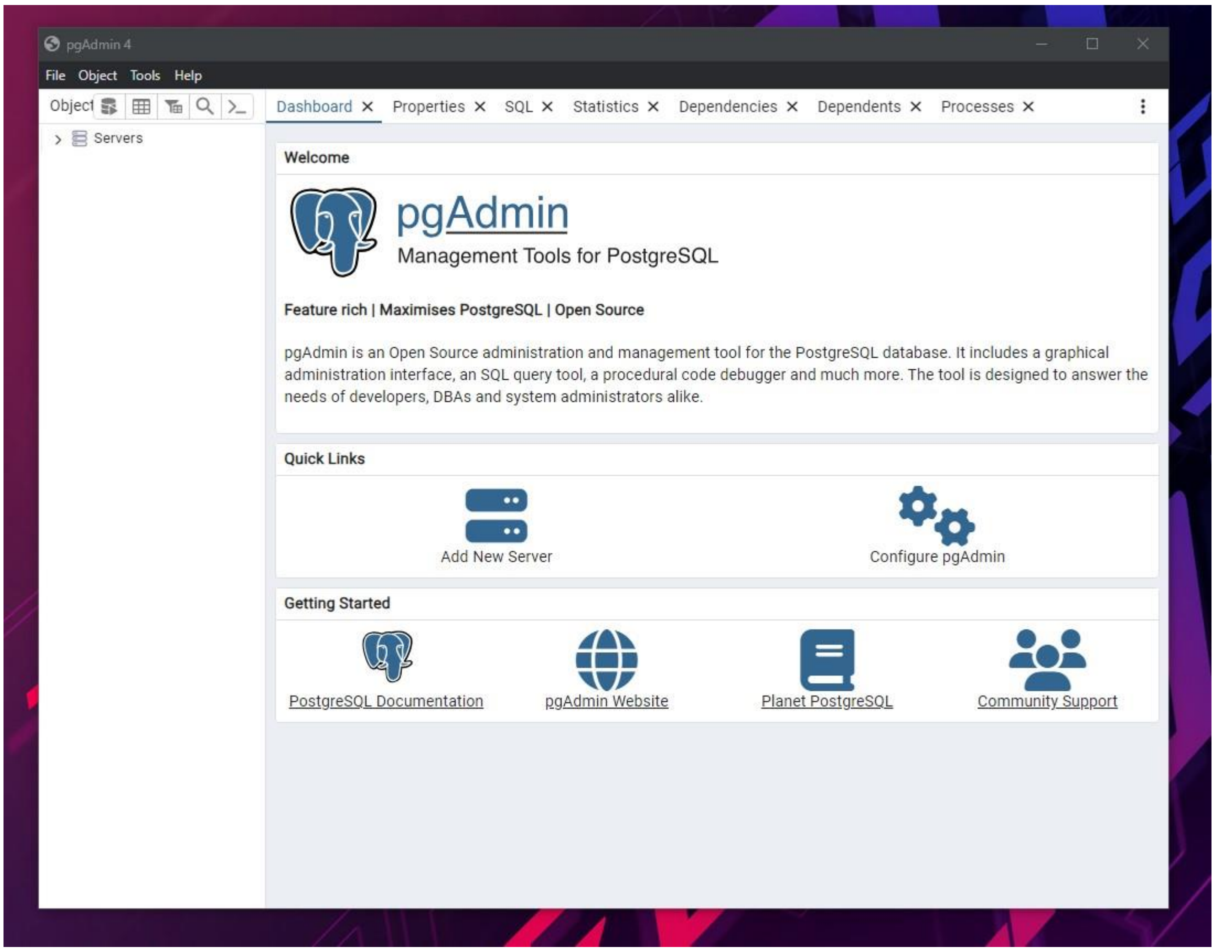
2. Add a Server:

   In pgAdmin, navigate to the "Browser" pane on the left.

   Right-click on "Servers" and choose "Create > Server..."

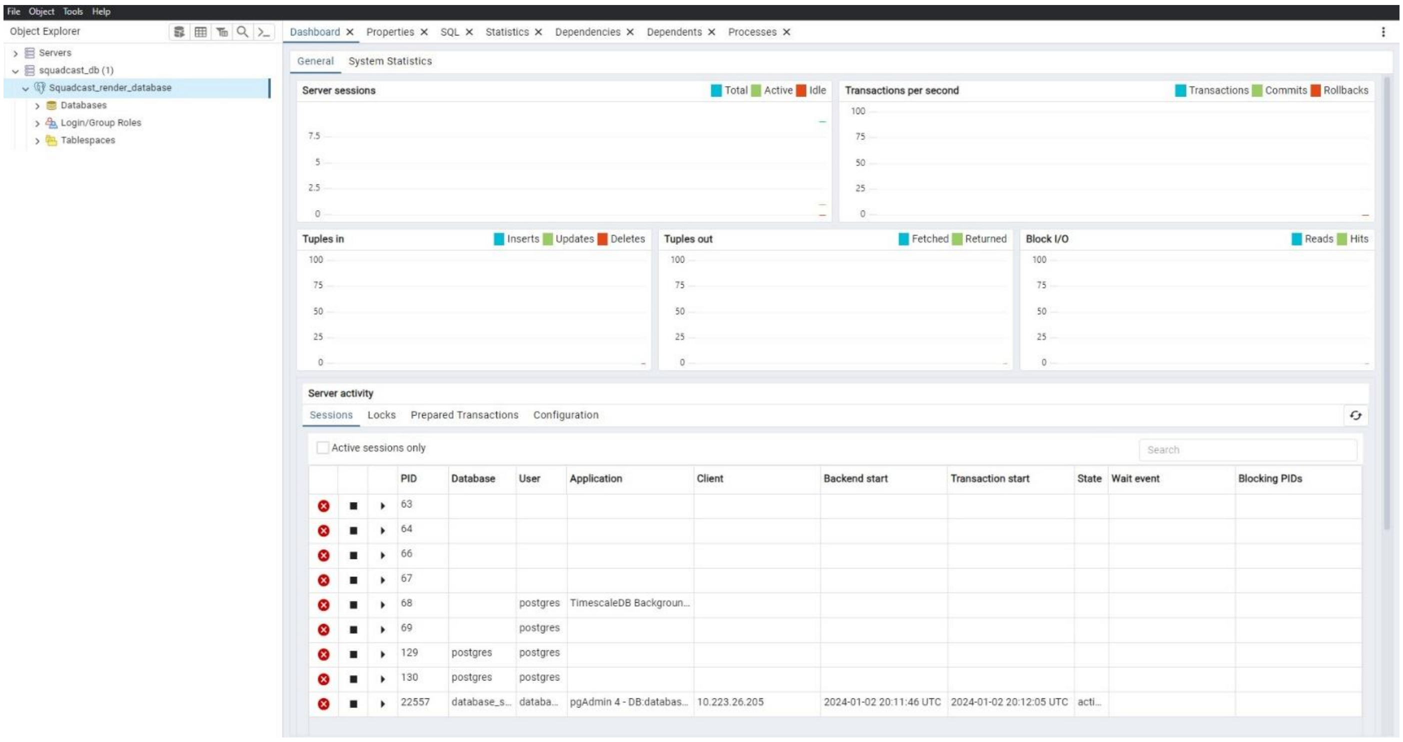
   Fill in Connection Details:

- Host: Your Render PostgreSQL database host.
- Port: Your Render PostgreSQL database port.
- Maintenance Database: The default database name you've created on Render.
- Username: Your Render PostgreSQL database username.
- Password: Your Render PostgreSQL database password.
- Click "Save."
- Connect to the Server:



Feature rich | Maximises PostgreSQL | Open Source

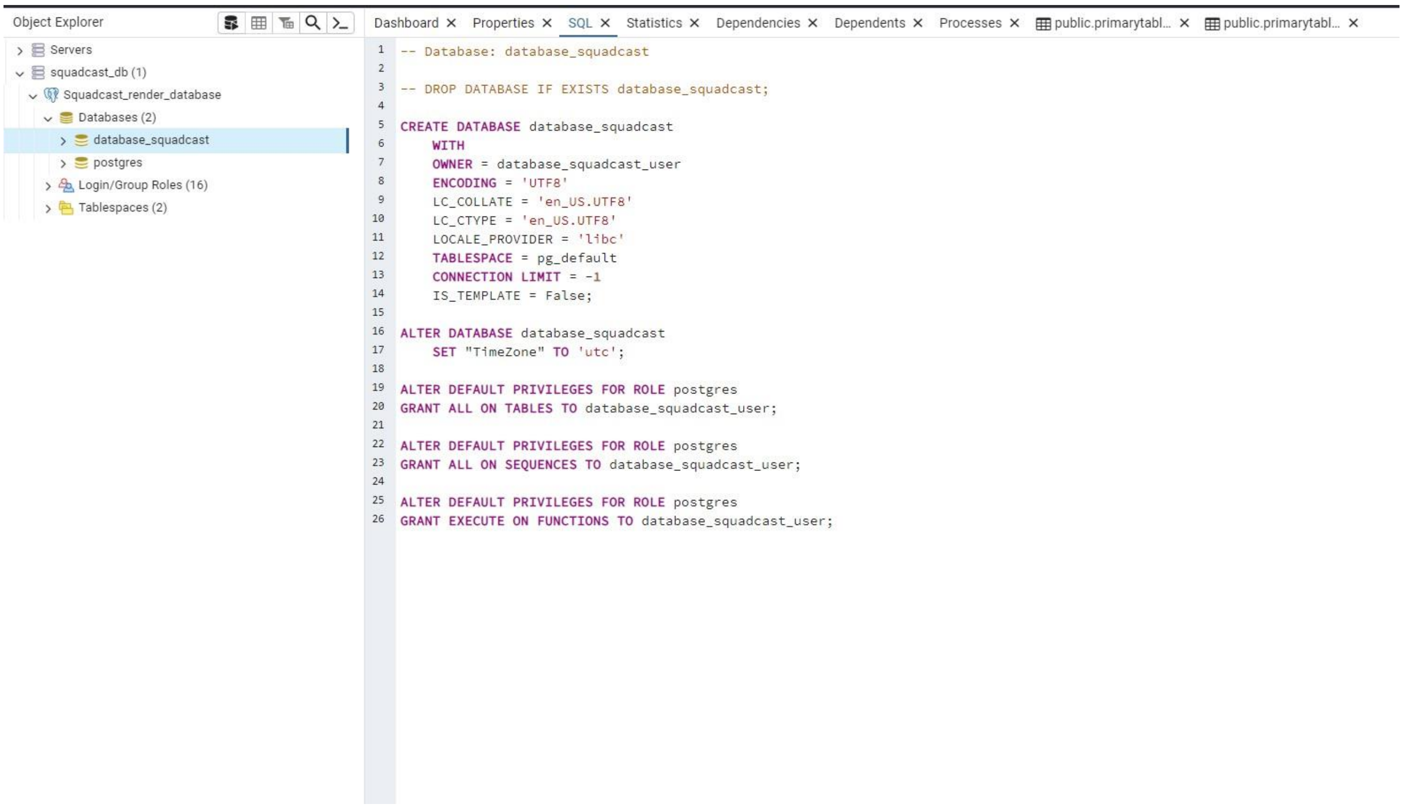
pgAdmin is an Open Source administration and management tool for the PostgreSQL database. It includes a graphical administration interface, an SQL query tool, a procedural code debugger and much more. The tool is designed to answer the needs of developers, DBAs and system administrators alike.



PID	Database	User	Application	Client	Backend start	Transaction start	State	Wait event	Blocking PIDs
63									
64									
66									
67									
68		postgres	TimescaleDB Backgroun...						
69		postgres							
129	postgres	postgres							
130	postgres	postgres							
22557	database_s...	databa...	pgAdmin 4 - DB:databas...	10.223.26.205	2024-01-02 20:11:46 UTC	2024-01-02 20:12:05 UTC	acti...		

Now, after being connected to your PostgreSQL database instance on Render using pgAdmin.

Grant the Super User permissions to the database User to perform the further task.

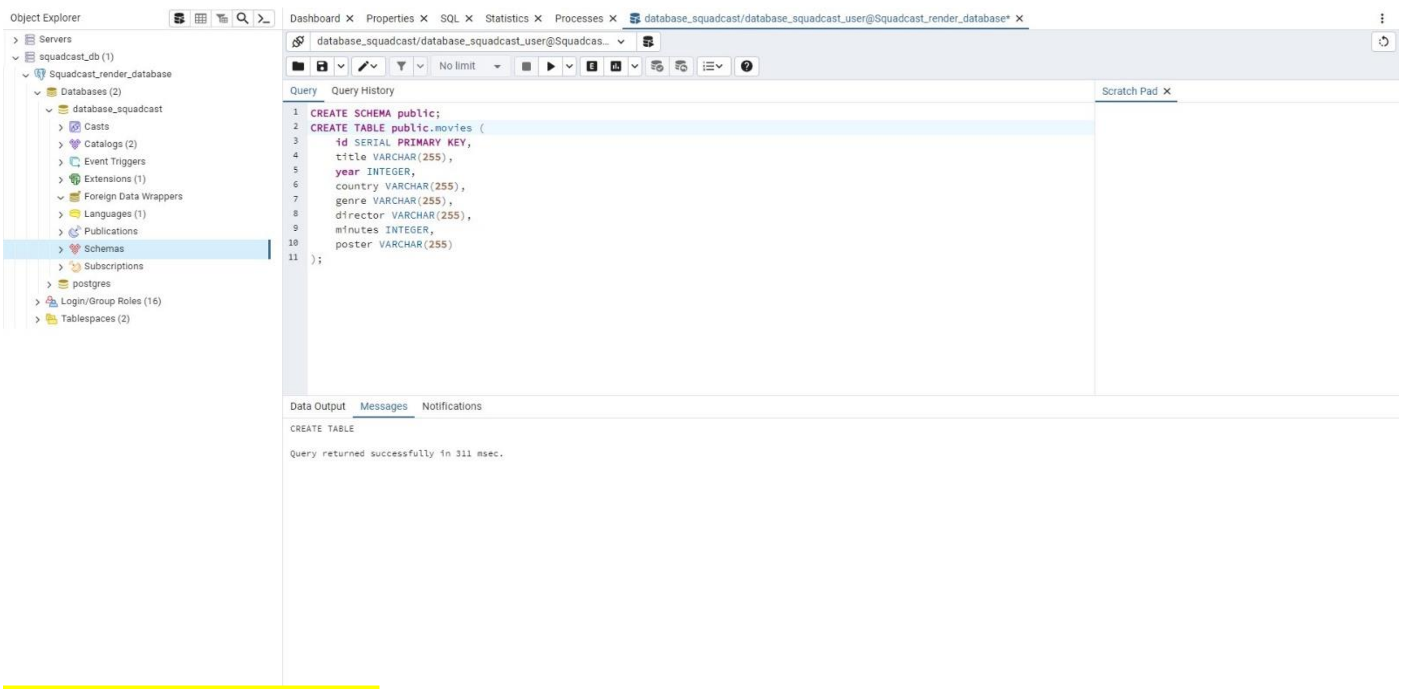

Object Explorer
Servers
squadcast_db (1)
Squadcast_render_database
Databases (2)
database_squadcast
postgres
Login/Group Roles (16)
Tablespaces (2)
Dashboard Properties SQL Statistics Dependencies Dependents Processes public.primarytabl... public.primarytabl...

```
-- Database: database_squadcast

-- DROP DATABASE IF EXISTS database_squadcast;

CREATE DATABASE database_squadcast
    WITH
    OWNER = database_squadcast_user
    ENCODING = 'UTF8'
    LC_COLLATE = 'en_US.UTF8'
    LC_CTYPE = 'en_US.UTF8'
    LOCALE_PROVIDER = 'libc'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;

ALTER DATABASE database_squadcast
    SET "TimeZone" TO 'utc';

ALTER DEFAULT PRIVILEGES FOR ROLE postgres
GRANT ALL ON TABLES TO database_squadcast_user;

ALTER DEFAULT PRIVILEGES FOR ROLE postgres
GRANT ALL ON SEQUENCES TO database_squadcast_user;

ALTER DEFAULT PRIVILEGES FOR ROLE postgres
GRANT EXECUTE ON FUNCTIONS TO database_squadcast_user;
```

STEP 3: Create a Schema in the Database So that we can easily create tables in the squadcast_db.

- For that Right Click on the database and select Query tool, and Write Command

  CREATE SCHEMA public   //public is SCHEMA name.
- And after that create first table name movies according to our Csv data.



```
CREATE SCHEMA public;
CREATE TABLE public.movies (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255),
    year INTEGER,
    country VARCHAR(255),
    genre VARCHAR(255),
    director VARCHAR(255),
    minutes INTEGER,
    poster VARCHAR(255)
);
```

```
CREATE TABLE

Query returned successfully in 311 msec.
```

CREATE TABLE public.movies (//movies – name of table

id PRIMARY KEY,   // id which is primary key – unique key.

title VARCHAR(255),

year INTEGER,

country VARCHAR(255),

genre VARCHAR(255),

director VARCHAR(255),

minutes INTEGER ;

- id: Assumes identifier as the primary key.
- title, country, genre, director, and poster: Columns with the VARCHAR data type to store textual information.
- year and minutes: Columns with the INTEGER data type to store numeric information.

After that CREATE Ratings table according to ratings.csv file

CREATE TABLE ratings (

rating_id

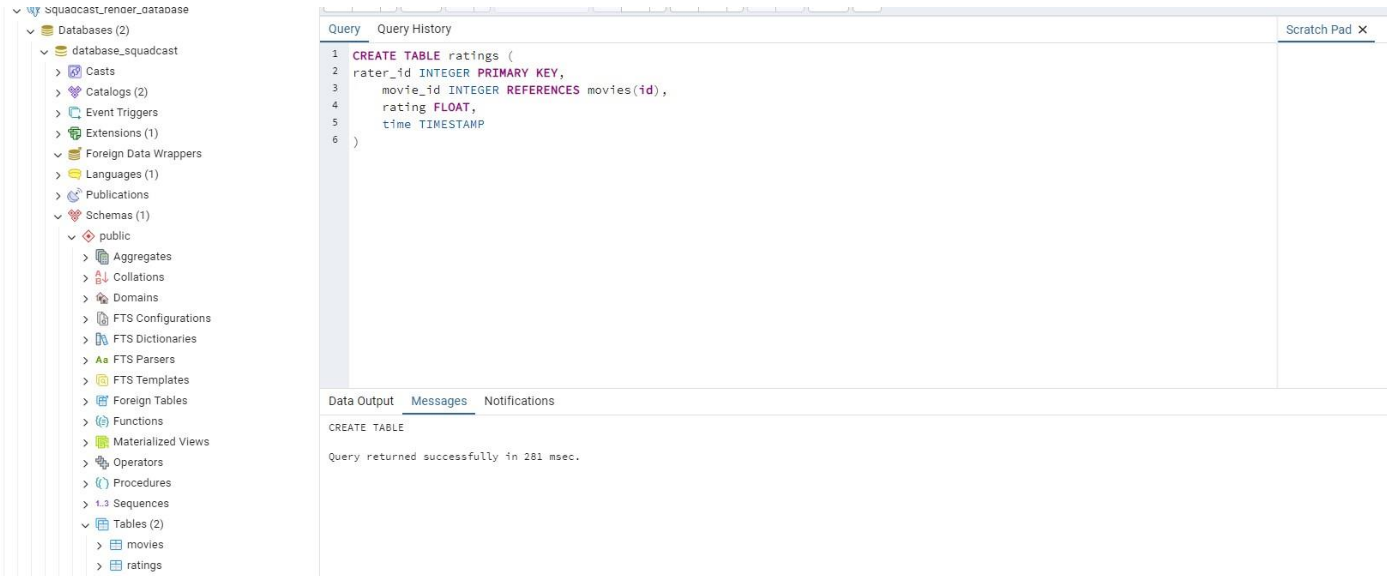
rater_id INTEGER,

movie_id INTEGER REFERENCES movies(id),

rating INTEGER,

time TIMESTAMP

- rating_id: Assumes an automatically incrementing identifier as the primary key.
- rater_id and movie_id: Columns with the INTEGER data type. The movie_id column is set as a foreign key referencing the id column in the "movies" table.
- rating: Column with the FLOAT data type to store a floating-point value for the rating.
- time: Column with the TIMESTAMP data type to store the timestamp of the rating.

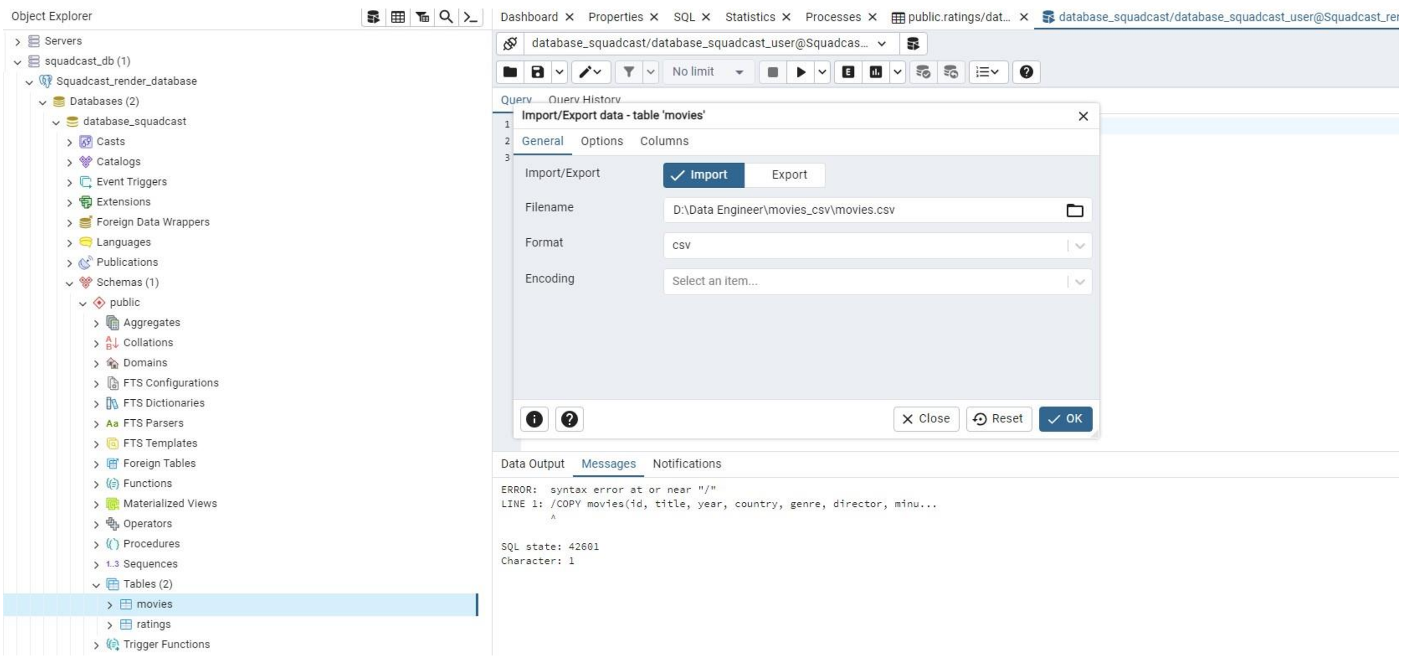


```
CREATE TABLE ratings (
rater_id INTEGER PRIMARY KEY,
    movie_id INTEGER REFERENCES movies(id),
    rating FLOAT,
    time TIMESTAMP
)
```

```
CREATE TABLE

Query returned successfully in 281 msec.
```

STEP 4: Now After Creating both the tables time to import the csv files which we have downloaded.

Select database => Under SCHEMA sections => Select table > movies => Right Click => Select Import data.



Import/Export: Import | Export
Filename: D:\Data Engineer\movies_csv\movies.csv
Format: csv
Encoding: Select an item...

```
ERROR:  syntax error at or near "/"
LINE 1: /COPY movies(id, title, year, country, genre, director, minu...
        ^

SQL state: 42601
Character: 1
```

Now Select File Path where you store eg: D\Data_Engineer\movies_csv\movies.csv

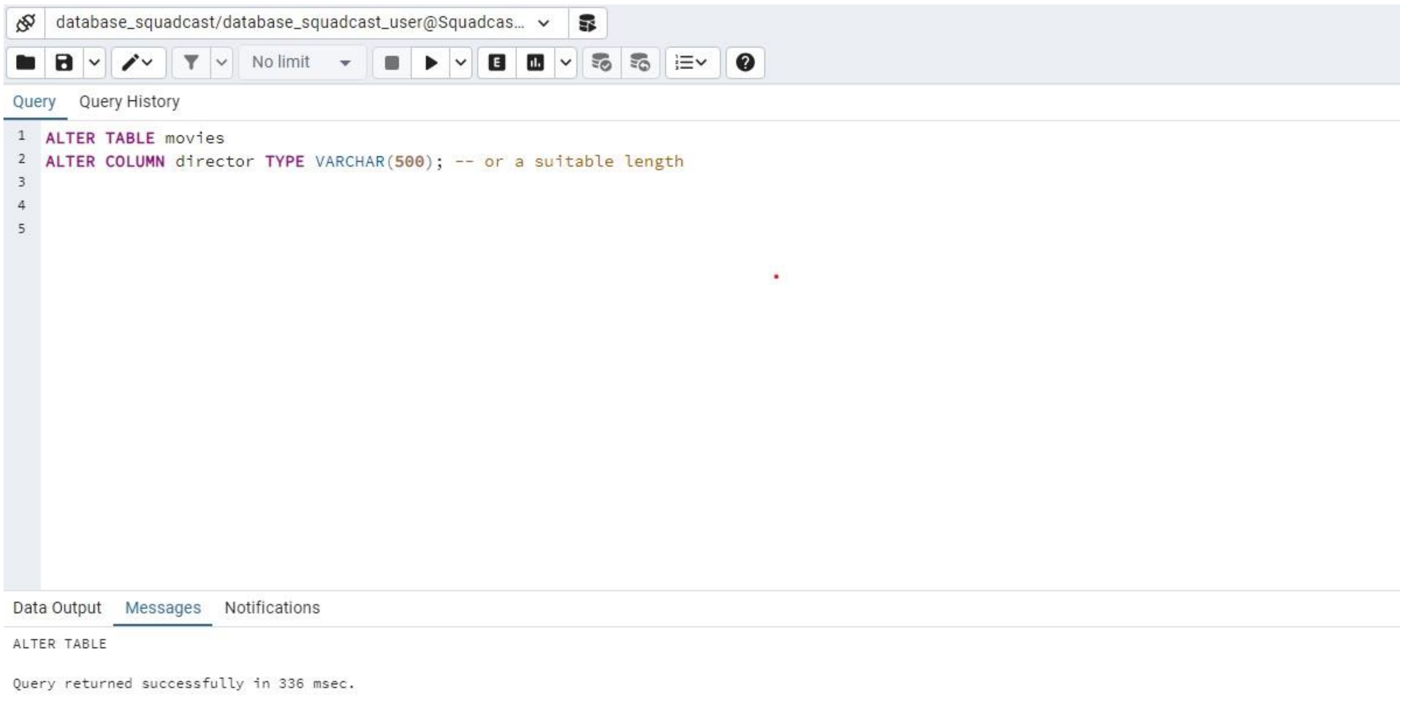
For Encoding level, you can set it to "UTF8," it indicates that the database is encoded in UTF-8, which is a widely used character encoding for supporting a broad range of characters and symbols.

Here Iam getting one length error while importing the data that :

ERROR: value too long for type character varying(255)

CONTEXT: COPY movies, line 2527, column director: "Olivier Assayas, Fracdacric Auburtin, Emmanuel Benbihy, Gurinder Chadha, Sylvain Chomet, Ethan Coen,..."

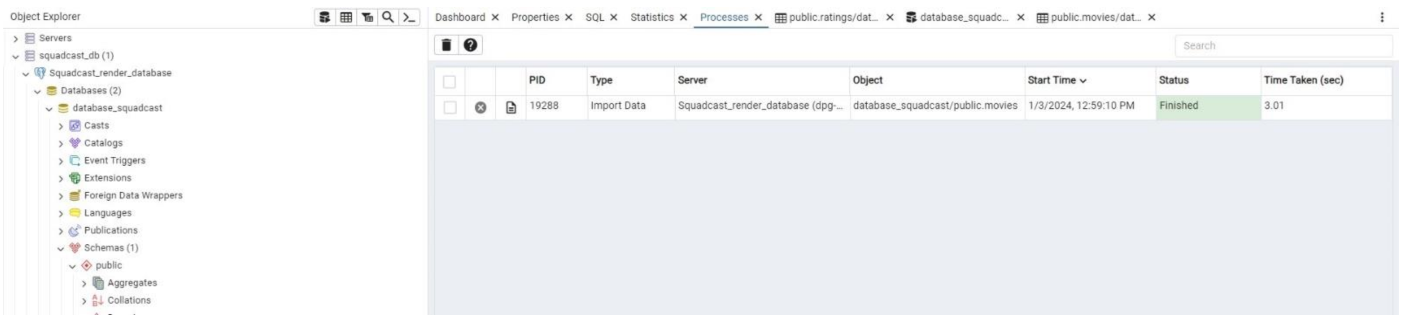
So, to Fix this I Alter by table column and Set type VARCHAR(500) for Suitable length.



```sql
ALTER TABLE movies
ALTER COLUMN director TYPE VARCHAR(500); -- or a suitable length
```

```
ALTER TABLE

Query returned successfully in 336 msec.
```

After that Import Has Successful.



			PID	Type	Server	Object	Start Time	Status	Time Taken (sec)
☐	⊗	🗎	19288	Import Data	Squadcast_render_database (dpg-...	database_squadcast/public.movies	1/3/2024, 12:59:10 PM	Finished	3.01

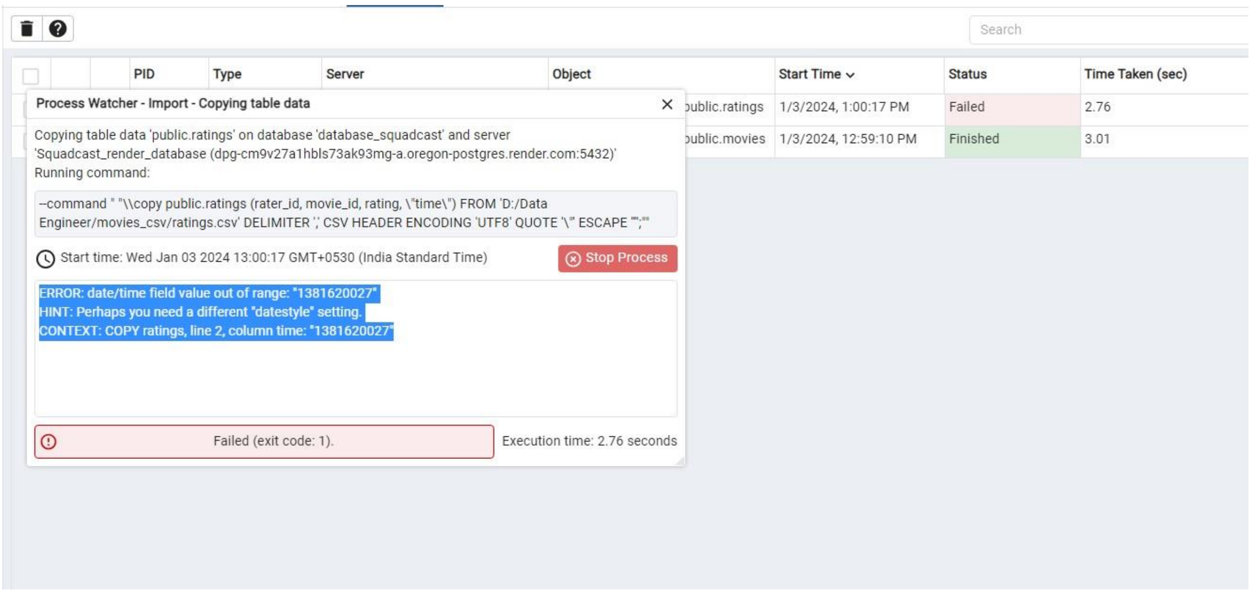
Now Under Rating table select Import data.

Again, while Importing I get an Error :

ERROR: date/time field value out of range: "1381620027"

HINT: Perhaps you need a different "datestyle" setting.

CONTEXT: COPY ratings, line 2, column time: "1381620027"



			PID	Type	Server	Object	Start Time	Status	Time Taken (sec)
						public.ratings	1/3/2024, 1:00:17 PM	Failed	2.76
						public.movies	1/3/2024, 12:59:10 PM	Finished	3.01

Process Watcher - Import - Copying table data

Copying table data 'public.ratings' on database 'database_squadcast' and server 'Squadcast_render_database (dpg-cm9v27a1hbls73ak93mg-a.oregon-postgres.render.com:5432)'
Running command:

```
--command " "\\copy public.ratings (rater_id, movie_id, rating, \"time\") FROM 'D:/Data
Engineer/movies_csv/ratings.csv' DELIMITER ',' CSV HEADER ENCODING 'UTF8' QUOTE '\"' ESCAPE '''';""
```

Start time: Wed Jan 03 2024 13:00:17 GMT+0530 (India Standard Time)

```
ERROR: date/time field value out of range: "1381620027"
HINT: Perhaps you need a different "datestyle" setting.
CONTEXT: COPY ratings, line 2, column time: "1381620027"
```

Failed (exit code: 1).

Execution time: 2.76 seconds

So, to fix this Error I have to change the time format of our ratings.csv file by default error "date/time field value out of range" suggests that there is an issue with the date value being inserted into the "time" column in your "ratings" table. The value "1381620027" seems to be a timestamp in Unix epoch format (seconds since January 1, 1970).

So, I must change this epoch time format to UST format which is Human readable.

This can be done by altering the time column in ratings.csv file and convert the time column to SQL timestamp format so I used one formula in csv for successful data importation in the database.

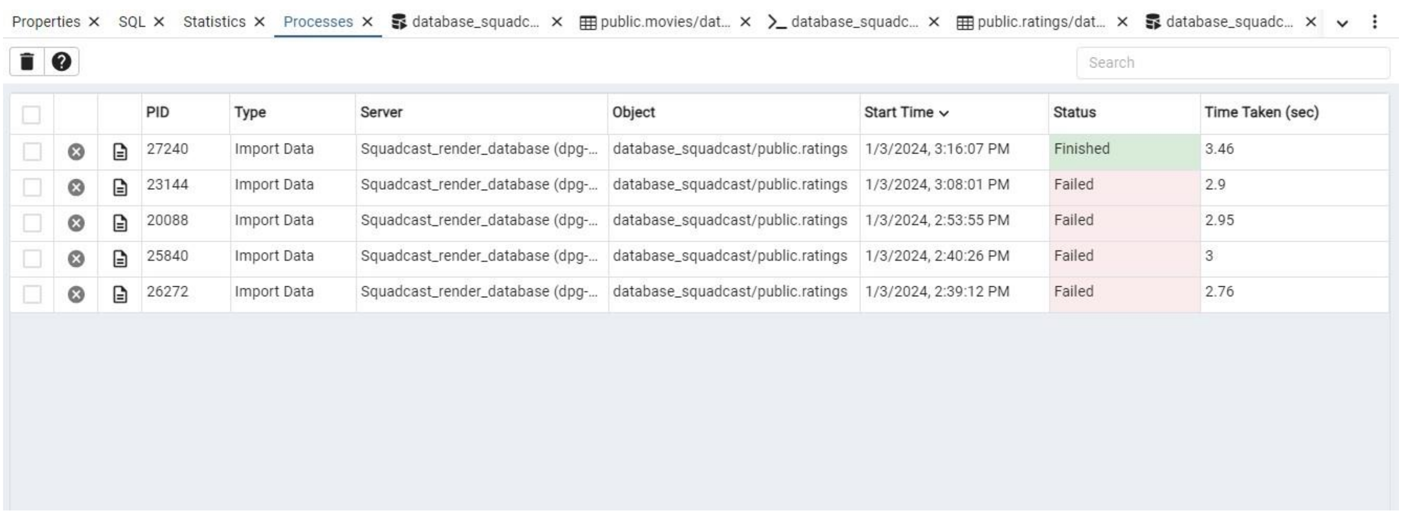
Now Add the formula like below

=(((B1/60)/60)/24)+DATE(1970,1,1)

Now format the cell like below or required format(Custom format)

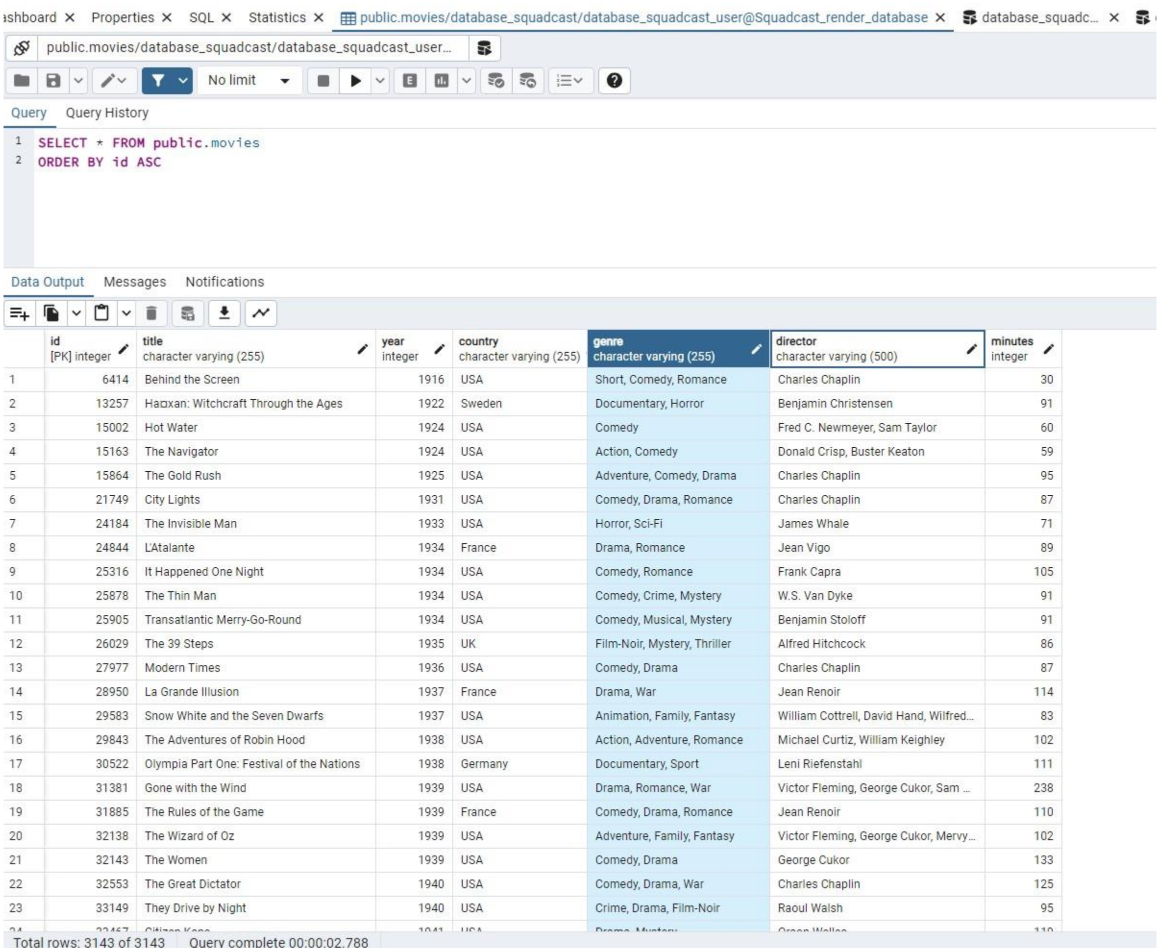
m/d/yyyy h:mm:ss.000

After that … Finally data has been imported in ratings table



			PID	Type	Server	Object	Start Time	Status	Time Taken (sec)
			27240	Import Data	Squadcast_render_database (dpg-...	database_squadcast/public.ratings	1/3/2024, 3:16:07 PM	Finished	3.46
			23144	Import Data	Squadcast_render_database (dpg-...	database_squadcast/public.ratings	1/3/2024, 3:08:01 PM	Failed	2.9
			20088	Import Data	Squadcast_render_database (dpg-...	database_squadcast/public.ratings	1/3/2024, 2:53:55 PM	Failed	2.95
			25840	Import Data	Squadcast_render_database (dpg-...	database_squadcast/public.ratings	1/3/2024, 2:40:26 PM	Failed	3
			26272	Import Data	Squadcast_render_database (dpg-...	database_squadcast/public.ratings	1/3/2024, 2:39:12 PM	Failed	2.76

Here is the Overview of both the tables.



ashboard
Properties
SQL
Statistics
public.movies/database_squadcast/database_squadcast_user@Squadcast_render_database
database_squadc...
public.movies/database_squadcast/database_squadcast_user...
No limit
Query
Query History
Data Output
Messages
Notifications

```
SELECT * FROM public.movies
ORDER BY id ASC
```

	id [PK] integer	title character varying (255)	year integer	country character varying (255)	genre character varying (255)	director character varying (500)	minutes integer
1	6414	Behind the Screen	1916	USA	Short, Comedy, Romance	Charles Chaplin	30
2	13257	Hαxan: Witchcraft Through the Ages	1922	Sweden	Documentary, Horror	Benjamin Christensen	91
3	15002	Hot Water	1924	USA	Comedy	Fred C. Newmeyer, Sam Taylor	60
4	15163	The Navigator	1924	USA	Action, Comedy	Donald Crisp, Buster Keaton	59
5	15864	The Gold Rush	1925	USA	Adventure, Comedy, Drama	Charles Chaplin	95
6	21749	City Lights	1931	USA	Comedy, Drama, Romance	Charles Chaplin	87
7	24184	The Invisible Man	1933	USA	Horror, Sci-Fi	James Whale	71
8	24844	L'Atalante	1934	France	Drama, Romance	Jean Vigo	89
9	25316	It Happened One Night	1934	USA	Comedy, Romance	Frank Capra	105
10	25878	The Thin Man	1934	USA	Comedy, Crime, Mystery	W.S. Van Dyke	91
11	25905	Transatlantic Merry-Go-Round	1934	USA	Comedy, Musical, Mystery	Benjamin Stoloff	91
12	26029	The 39 Steps	1935	UK	Film-Noir, Mystery, Thriller	Alfred Hitchcock	86
13	27977	Modern Times	1936	USA	Comedy, Drama	Charles Chaplin	87
14	28950	La Grande Illusion	1937	France	Drama, War	Jean Renoir	114
15	29583	Snow White and the Seven Dwarfs	1937	USA	Animation, Family, Fantasy	William Cottrell, David Hand, Wilfred...	83
16	29843	The Adventures of Robin Hood	1938	USA	Action, Adventure, Romance	Michael Curtiz, William Keighley	102
17	30522	Olympia Part One: Festival of the Nations	1938	Germany	Documentary, Sport	Leni Riefenstahl	111
18	31381	Gone with the Wind	1939	USA	Drama, Romance, War	Victor Fleming, George Cukor, Sam ...	238
19	31885	The Rules of the Game	1939	France	Comedy, Drama, Romance	Jean Renoir	110
20	32138	The Wizard of Oz	1939	USA	Adventure, Family, Fantasy	Victor Fleming, George Cukor, Mervy...	102
21	32143	The Women	1939	USA	Comedy, Drama	George Cukor	133
22	32553	The Great Dictator	1940	USA	Comedy, Drama, War	Charles Chaplin	125
23	33149	They Drive by Night	1940	USA	Crime, Drama, Film-Noir	Raoul Walsh	95
24	33467	Citizen Kane	1941	USA	Drama, Mystery	Orson Welles	119

Total rows: 3143 of 3143 Query complete 00:00:02.788

So, the First Phase of task is Complete now comes the

2. Insights and Analysis:

- Use any scripting language of your choice

(e.g., Python, JavaScript, etc.) to perform the

following insights:

So, I used **JavaScript** as scripting language because I feel,

Confident in it.

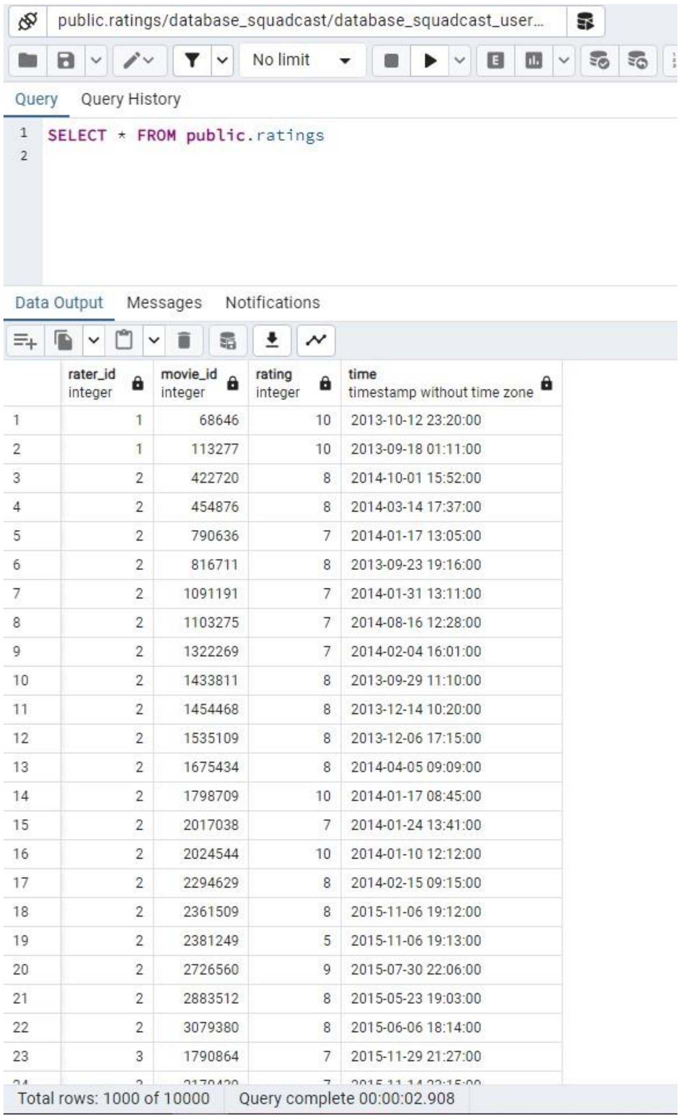
So Iam using online platform Repl.it.

Repl.it allows you to write, run, and share

JavaScript code directly from your browser.

Here are the steps to perform the analysis using

JavaScript on Repl.it:


public.ratings/database_squadcast/database_squadcast_user...
No limit
Query
Query History

```
SELECT * FROM public.ratings
```

Data Output
Messages
Notifications

	rater_id integer	movie_id integer	rating integer	time timestamp without time zone
1	1	68646	10	2013-10-12 23:20:00
2	1	113277	10	2013-09-18 01:11:00
3	2	422720	8	2014-10-01 15:52:00
4	2	454876	8	2014-03-14 17:37:00
5	2	790636	7	2014-01-17 13:05:00
6	2	816711	8	2013-09-23 19:16:00
7	2	1091191	7	2014-01-31 13:11:00
8	2	1103275	7	2014-08-16 12:28:00
9	2	1322269	7	2014-02-04 16:01:00
10	2	1433811	8	2013-09-29 11:10:00
11	2	1454468	8	2013-12-14 10:20:00
12	2	1535109	8	2013-12-06 17:15:00
13	2	1675434	8	2014-04-05 09:09:00
14	2	1798709	10	2014-01-17 08:45:00
15	2	2017038	7	2014-01-24 13:41:00
16	2	2024544	10	2014-01-10 12:12:00
17	2	2294629	8	2014-02-15 09:15:00
18	2	2361509	8	2015-11-06 19:12:00
19	2	2381249	5	2015-11-06 19:13:00
20	2	2726560	9	2015-07-30 22:06:00
21	2	2883512	8	2015-05-23 19:03:00
22	2	3079380	8	2015-06-06 18:14:00
23	3	1790864	7	2015-11-29 21:27:00

Total rows: 1000 of 10000 Query complete 00:00:02.908

Open Repl.it:

Visit Repl.it, and create a new JavaScript (Node.js) repl.

Now In Console write **npm install;** // it downloads require node modules

After that make sure we have Node.js installed on our machine and that the 'pg' module is installed in your project.

Command : ' **npm install pg** '

To Retrieve data from PostgreSQL database set up your script to connect using the 'pg' module and retrieve data..

Step : Okay So now to connect our database :

```
const { Client } = require("pg");
const client = new Client({
  user: "database_squadcast_user",
  host: "dpg-cm9v27a1hbls73ak93mg-a.oregon-postgres.render.com",
  database: "database_squadcast",
  password: "eZDRW57lrRcHG2w73caRfYfE5Lka1xQV",
  port: 5432,
  ssl: {
    rejectUnauthorized: false,
  },
  connectionTimeoutMillis: 5000, // Adjust timeout
  query_timeout: 5000, //
});
```

- ***username***: Replace this with your Render database username.
- ***host***: Replace this with the host address or URL of your Render database.
- ***database_name***: Replace this with the name of your Render database.
- ***password***: Replace this with your Render database password.
- ***port***: Replace this with the port number used by your Render database.

Here's a breakdown:

require('pg'): This is a Node.js function used to include external modules in our code. In this case, it's importing the 'pg' module, which is a PostgreSQL client for Node.js.

const { Client }: This is object restructuring syntax in JavaScript. It is used to extract the Client class from the 'pg' module and assign it to a variable named Client. The Client class is a part of the 'pg' module and is used to interact with PostgreSQL databases.

So, the entire line is essentially saying, "Import the 'pg' module, and extract the Client class from it, assigning it to a variable named Client."

This line is commonly used when working with PostgreSQL databases in Node.js. It allows you to create a PostgreSQL client instance (Client) that can be used to connect to and interact with a PostgreSQL database.

So now :

A) Top 5 Movie Titles: Sort and print the top 5 movie titles based on the following criteria:

● Duration

● Year of Release

● Average rating (consider movies with minimum 5 ratings)

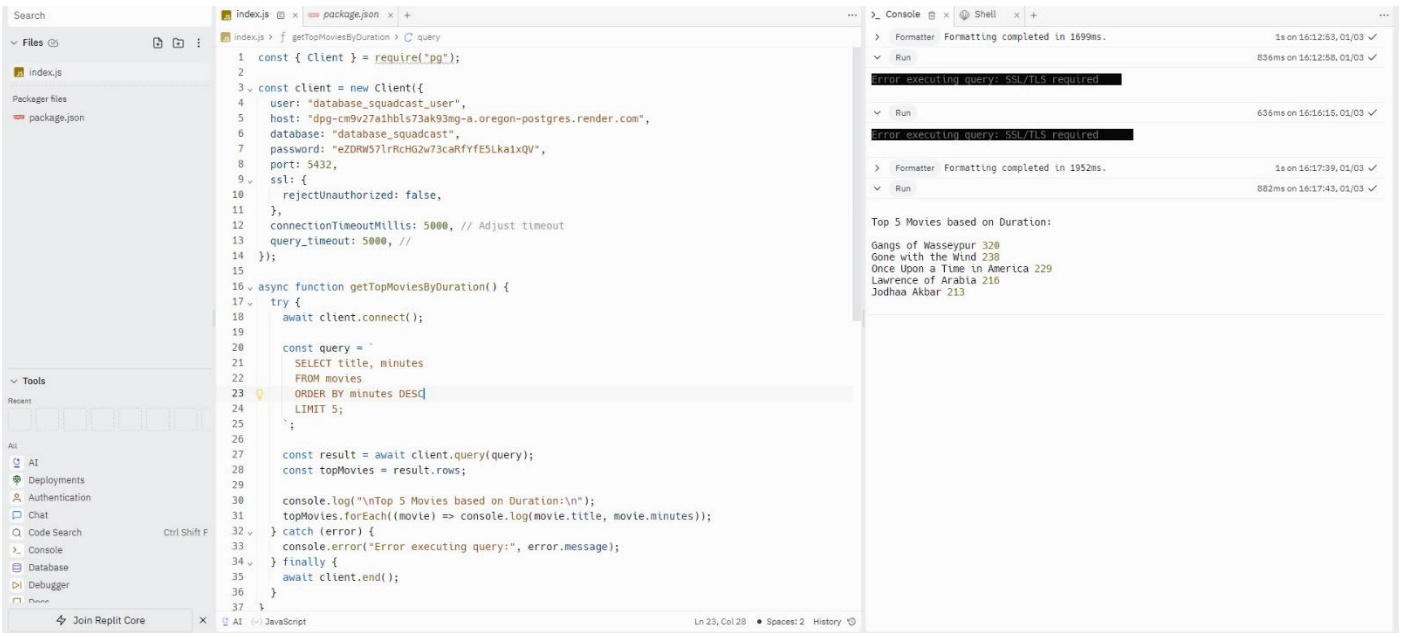
● Number of ratings given

1) **Duration :**

Query: "SELECT title, minutes FROM movies ORDER BY minutes DESC LIMIT 5;"

Explanation: This query retrieves the top 5 movies based on their duration (in minutes). It selects the movie title and duration from the "movies" table, orders the result by duration in descending order, and limits the result to 5 rows.

```
const { Client } = require("pg");
const client = new Client({
  user: "database_squadcast_user",
  host: "dpg-cm9v27a1hbls73ak93mg-a.oregon-postgres.render.com",
  database: "database_squadcast",
  password: "eZDRW57lrRcHG2w73caRfYfE5Lka1xQV",
  port: 5432,
  ssl: {
    rejectUnauthorized: false,
  },
  connectionTimeoutMillis: 5000, // Adjust timeout
  query_timeout: 5000, //
});

// Define SQL query for top duration
const topDurationQuery = "SELECT title, minutes FROM movies ORDER BY minutes DESC LIMIT 5;"

// Execute the query
async function executeTopDurationQuery() {
  const result = await client.query(topDurationQuery);

  // Print the results
  console.log("\nTop 5 Movies based on Duration:\n");
  result.rows.forEach(row => console.log(row));

  // Close the database connection
  client.end();
}

executeTopDurationQuery();
```



```
const { Client } = require("pg");

const client = new Client({
  user: "database_squadcast_user",
  host: "dpg-cm9v27a1hbls73ak93mg-a.oregon-postgres.render.com",
  database: "database_squadcast",
  password: "eZDRW57lrRcHG2w73caRfYfE5Lka1xQV",
  port: 5432,
  ssl: {
    rejectUnauthorized: false,
  },
  connectionTimeoutMillis: 5000, // Adjust timeout
  query_timeout: 5000, //
});

async function getTopMoviesByDuration() {
  try {
    await client.connect();

    const query = `
      SELECT title, minutes
      FROM movies
      ORDER BY minutes DESC
      LIMIT 5;
    `;

    const result = await client.query(query);
    const topMovies = result.rows;

    console.log("\nTop 5 Movies based on Duration:\n");
    topMovies.forEach((movie) => console.log(movie.title, movie.minutes));
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    await client.end();
  }
}
```

```
Formatter Formatting completed in 1699ms.        1s on 16:12:53, 01/03 ✓
Run                                              836ms on 16:12:58, 01/03 ✓
Error executing query: SSL/TLS required
Run                                              636ms on 16:16:15, 01/03 ✓
Error executing query: SSL/TLS required
Formatter Formatting completed in 1952ms.        1s on 16:17:39, 01/03 ✓
Run                                              882ms on 16:17:43, 01/03 ✓

Top 5 Movies based on Duration:

Gangs of Wasseypur 320
Gone with the Wind 238
Once Upon a Time in America 229
Lawrence of Arabia 216
Jodhaa Akbar 213
```

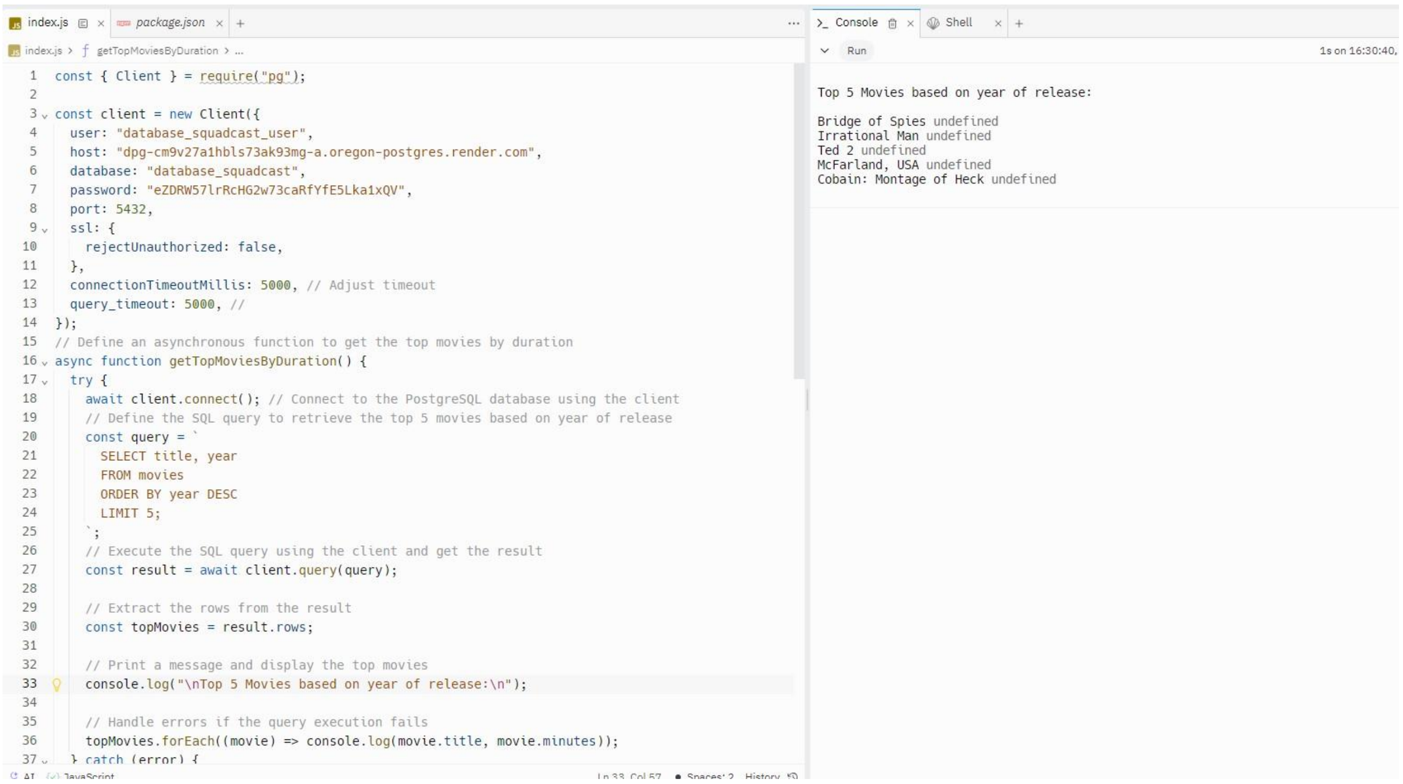
"SELECT title, minutes FROM movies ORDER BY minutes DESC LIMIT 5;"

2) Year of Release :

Query: "SELECT title, year FROM movies ORDER BY year DESC LIMIT 5;"

Explanation: This query retrieves the top 5 movies based on their year of release. It selects the movie title and release year from the "movies" table, orders the result by release year in descending order, and limits the result to 5 rows.

Top Average Rating (consider movies with minimum 5 ratings):



```javascript
const { Client } = require("pg");

const client = new Client({
  user: "database_squadcast_user",
  host: "dpg-cm9v27a1hbls73ak93mg-a.oregon-postgres.render.com",
  database: "database_squadcast",
  password: "eZDRW57lrRcHG2w73caRfYfE5Lka1xQV",
  port: 5432,
  ssl: {
    rejectUnauthorized: false,
  },
  connectionTimeoutMillis: 5000, // Adjust timeout
  query_timeout: 5000, //
});
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesByDuration() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
      SELECT title, year
      FROM movies
      ORDER BY year DESC
      LIMIT 5;
    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result
    const topMovies = result.rows;

    // Print a message and display the top movies
    console.log("\nTop 5 Movies based on year of release:\n");

    // Handle errors if the query execution fails
    topMovies.forEach((movie) => console.log(movie.title, movie.minutes));
  } catch (error) {
```

```
Top 5 Movies based on year of release:

Bridge of Spies undefined
Irrational Man undefined
Ted 2 undefined
McFarland, USA undefined
Cobain: Montage of Heck undefined
```

- SELECT title, year: This part specifies the columns we want to retrieve, which are the movie title and release year.
- FROM movies: This indicates that we are retrieving data from the "movies" table.
- ORDER BY year DESC: This orders the results based on the release year in descending order (DESC stands for descending). This means the movies with the latest release years will appear first.
- LIMIT 5: This limits the output to only the top 5 rows. Since we've ordered the results by release year, this gives us the top 5 movies with the latest release years.

3) Average rating (consider movies with minimum 5 ratings :

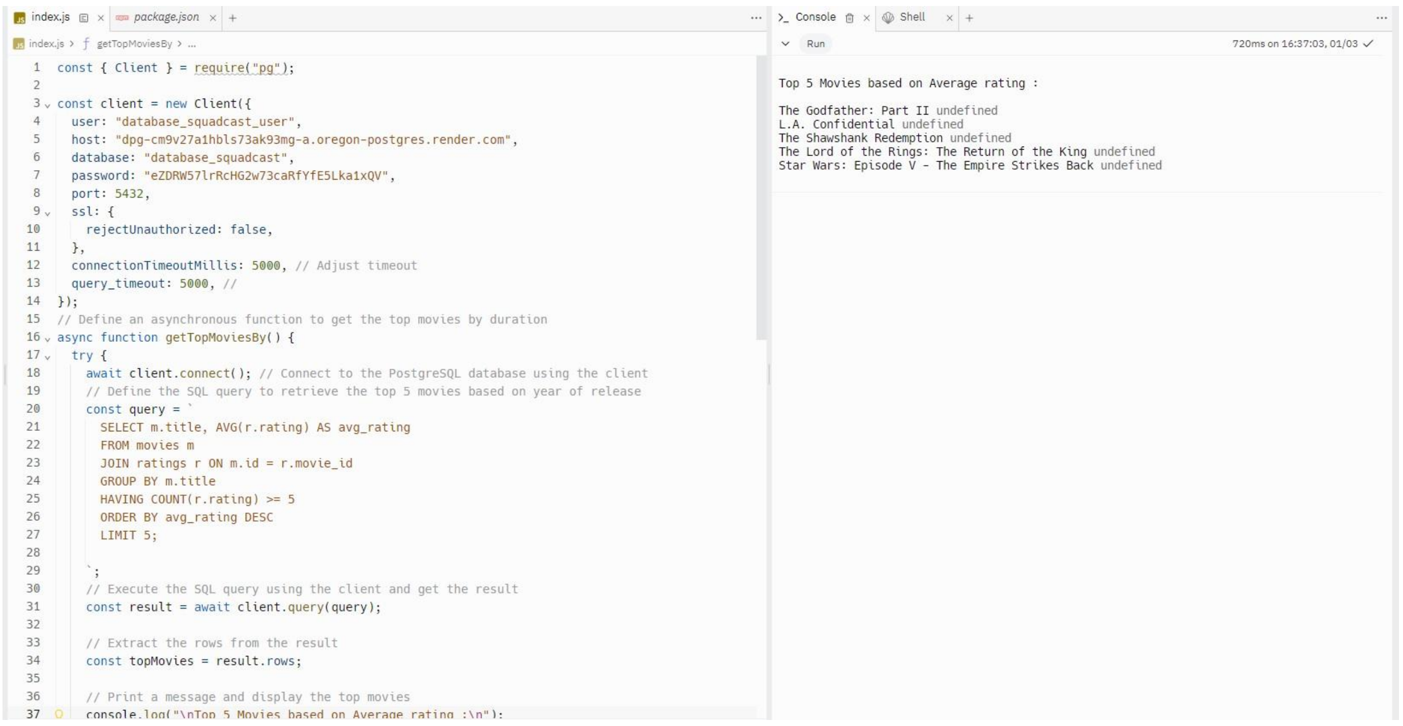
Query: "SELECT m.title, AVG(r.rating) AS avg_rating FROM movies m JOIN ratings r ON m.id = r.movie_id GROUP BY m.title HAVING COUNT(r.rating) >= 5 ORDER BY avg_rating DESC LIMIT 5;"

Explanation: This query calculates the average rating for each movie, considering only those movies with a minimum of 5 ratings. It selects the movie title and the average rating, joins the "movies" and "ratings" tables based on the movie ID, groups the result by movie title, filters out movies with fewer than 5 ratings using the HAVING clause, orders the result by average rating in descending order, and limits the result to 5 rows.

Top Number of Ratings Given:

- SELECT m.title, AVG(r.rating) AS avg_rating: Selects the movie title and calculates the average rating for each movie.
- FROM movies m: Specifies that the data is coming from the "movies" table and assigns it the alias m.
- JOIN ratings r ON m.id = r.movie_id: Joins the "movies" and "ratings" tables based on the movie ID.
- GROUP BY m.title: Groups the results by movie title.

- HAVING COUNT(r.rating) >= 5: Filters out movies that have less than 5 ratings.
- ORDER BY avg_rating DESC: Orders the results in descending order based on the average rating.
- LIMIT 5: Limits the output to the top 5 movies.



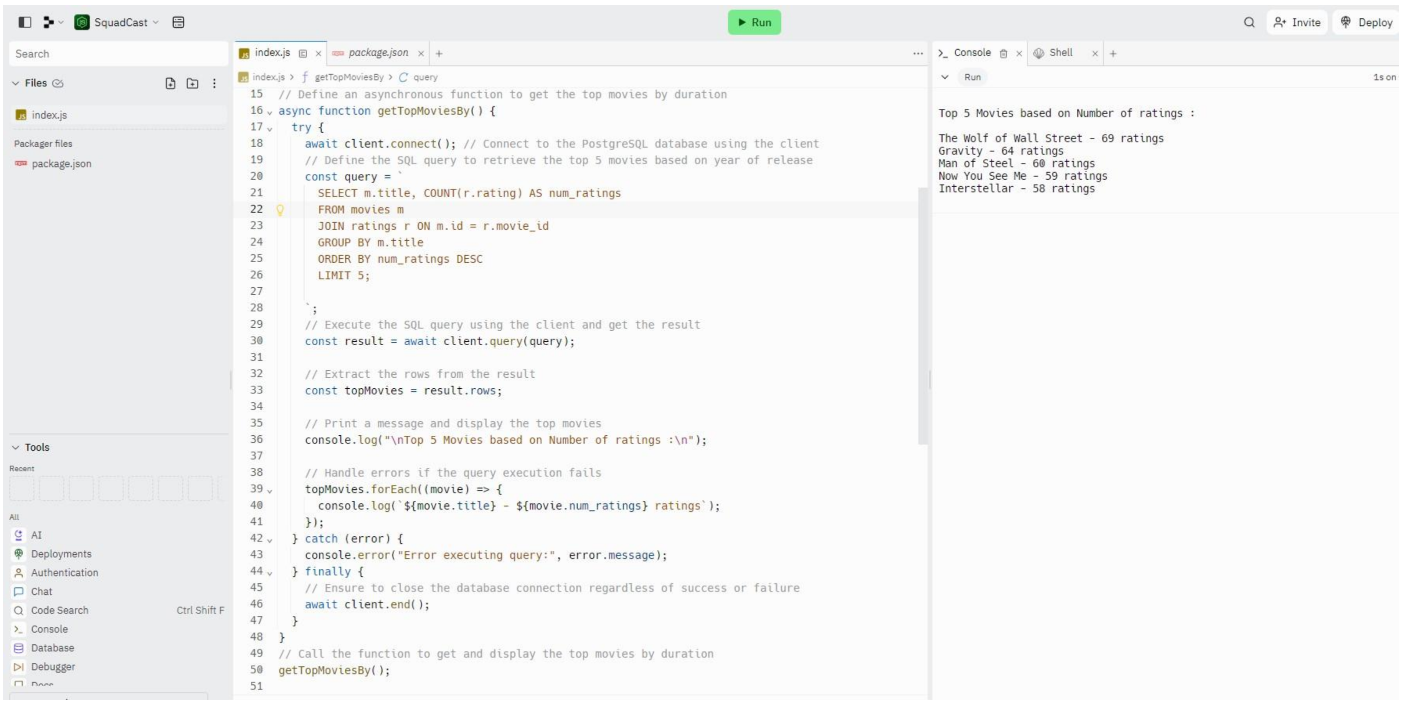
```javascript
const { Client } = require("pg");

const client = new Client({
  user: "database_squadcast_user",
  host: "dpg-cm9v27a1hbls73ak93mg-a.oregon-postgres.render.com",
  database: "database_squadcast",
  password: "eZDRW57lrRcHG2w73caRfYfE5Lka1xQV",
  port: 5432,
  ssl: {
    rejectUnauthorized: false,
  },
  connectionTimeoutMillis: 5000, // Adjust timeout
  query_timeout: 5000, //
});
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
      SELECT m.title, AVG(r.rating) AS avg_rating
      FROM movies m
      JOIN ratings r ON m.id = r.movie_id
      GROUP BY m.title
      HAVING COUNT(r.rating) >= 5
      ORDER BY avg_rating DESC
      LIMIT 5;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result
    const topMovies = result.rows;

    // Print a message and display the top movies
    console.log("\nTop 5 Movies based on Average rating :\n");
```

```
Top 5 Movies based on Average rating :

The Godfather: Part II undefined
L.A. Confidential undefined
The Shawshank Redemption undefined
The Lord of the Rings: The Return of the King undefined
Star Wars: Episode V - The Empire Strikes Back undefined
```

4) Number of ratings given

Query: "SELECT m.title, COUNT(r.rating) AS num_ratings FROM movies m JOIN ratings r ON m.id = r.movie_id GROUP BY m.title ORDER BY num_ratings DESC LIMIT 5;"

Explanation: This query retrieves the top 5 movies based on the total number of ratings they have received. It selects the movie title and the count of ratings, joins the "movies" and "ratings" tables based on the movie ID, groups the result by movie title, orders the result by the number of ratings in descending order, and limits the result to 5 rows.



```javascript
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
      SELECT m.title, COUNT(r.rating) AS num_ratings
      FROM movies m
      JOIN ratings r ON m.id = r.movie_id
      GROUP BY m.title
      ORDER BY num_ratings DESC
      LIMIT 5;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result
    const topMovies = result.rows;

    // Print a message and display the top movies
    console.log("\nTop 5 Movies based on Number of ratings :\n");

    // Handle errors if the query execution fails
    topMovies.forEach((movie) => {
      console.log(`${movie.title} - ${movie.num_ratings} ratings`);
    });
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
// Call the function to get and display the top movies by duration
getTopMoviesBy();
```

```
Top 5 Movies based on Number of ratings :

The Wolf of Wall Street - 69 ratings
Gravity - 64 ratings
Man of Steel - 60 ratings
Now You See Me - 59 ratings
Interstellar - 58 ratings
```

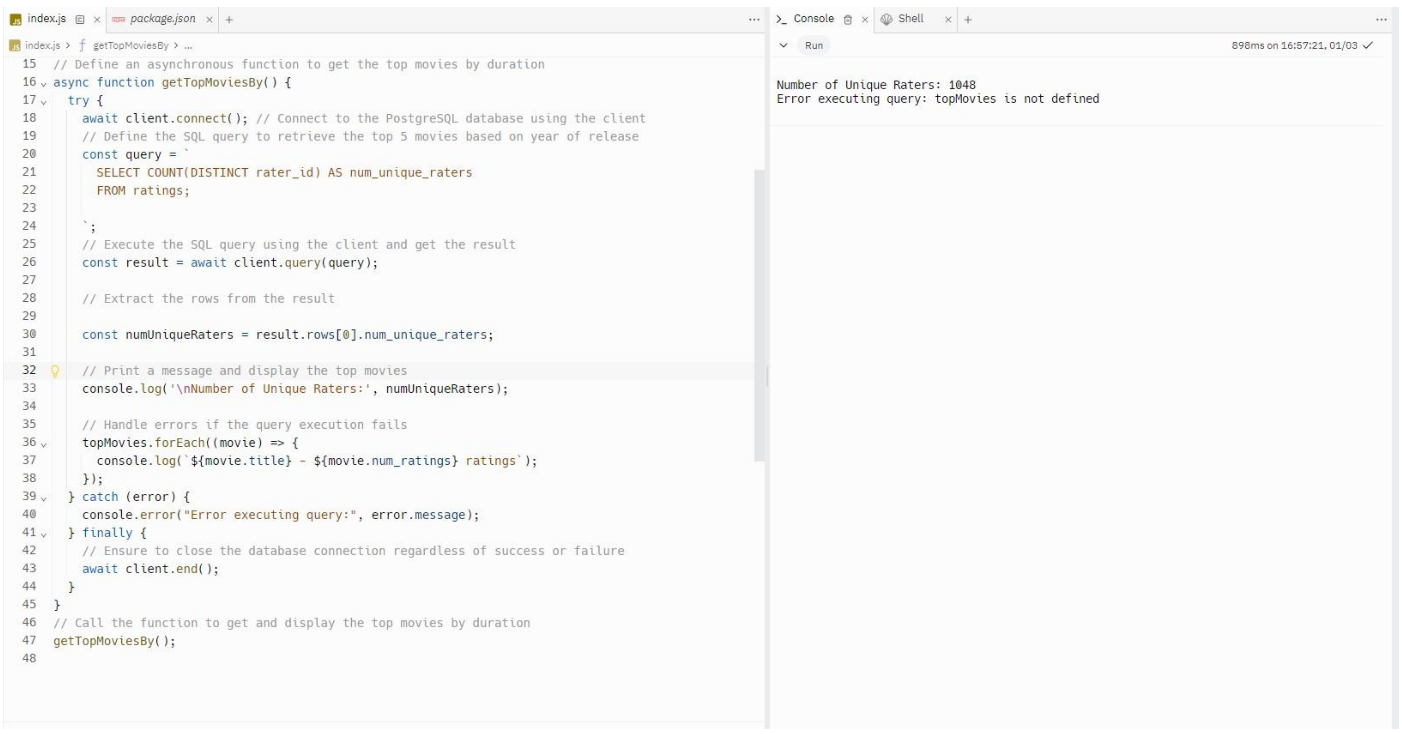
- SELECT m.title, COUNT(r.rating) AS num_ratings: Selects the movie title and counts the number of ratings for each movie.
- FROM movies m: Specifies that the data is coming from the "movies" table and assigns it the alias m.
- JOIN ratings r ON m.id = r.movie_id: Joins the "movies" and "ratings" tables based on the movie ID.
- GROUP BY m.title: Groups the results by movie title.
- ORDER BY num_ratings DESC: Orders the results in descending order based on the number of ratings.
- LIMIT 5: Limits the output to the top 5 movies.

b). Number of Unique Raters: Determine and print the count of unique rater IDs

Query : SELECT COUNT(DISTINCT rater_id) AS num_unique_raters FROM ratings;

Explanation :

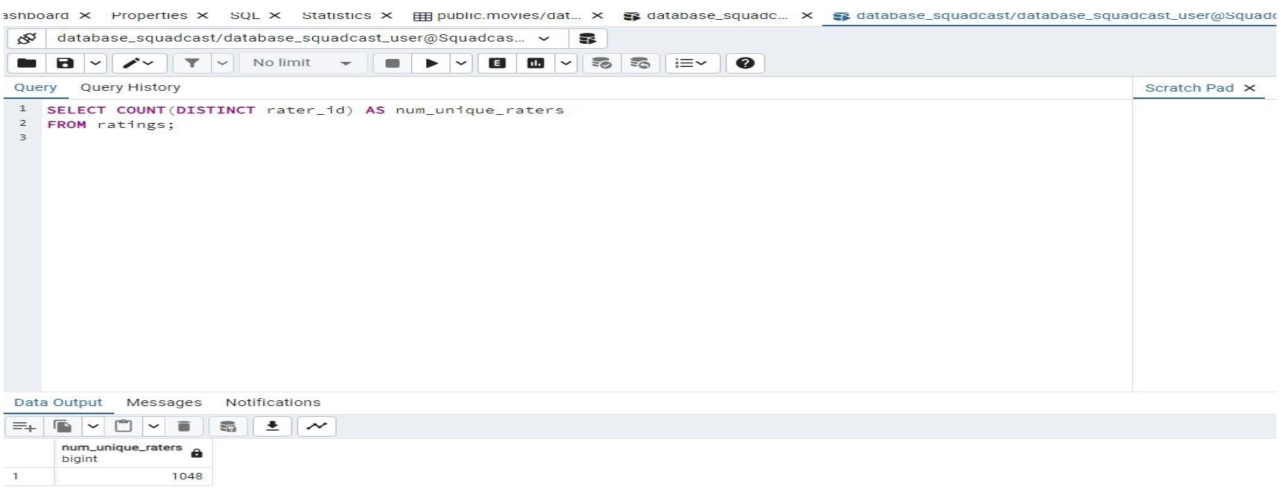
SELECT COUNT(DISTINCT rater_id) AS num_unique_raters: This command counts the number of distinct (unique) values in the rater_id column of the ratings table and aliases the result as num_unique_raters.



```
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
      SELECT COUNT(DISTINCT rater_id) AS num_unique_raters
      FROM ratings;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const numUniqueRaters = result.rows[0].num_unique_raters;

    // Print a message and display the top movies
    console.log('\nNumber of Unique Raters:', numUniqueRaters);

    // Handle errors if the query execution fails
    topMovies.forEach((movie) => {
      console.log(`${movie.title} - ${movie.num_ratings} ratings`);
    });
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
// Call the function to get and display the top movies by duration
getTopMoviesBy();
```

```
Number of Unique Raters: 1048
Error executing query: topMovies is not defined
```

For Cross Check I also write this Query in pgAdmin management tool :



```
SELECT COUNT(DISTINCT rater_id) AS num_unique_raters
FROM ratings;
```

	num_unique_raters bigint
1	1048

c. Top 5 Rater IDs: Sort and print the top 5 rater IDs based on:

● Most movies rated

Query : SELECT rater_id, COUNT(DISTINCT movie_id) AS num_movies_rated

FROM ratings

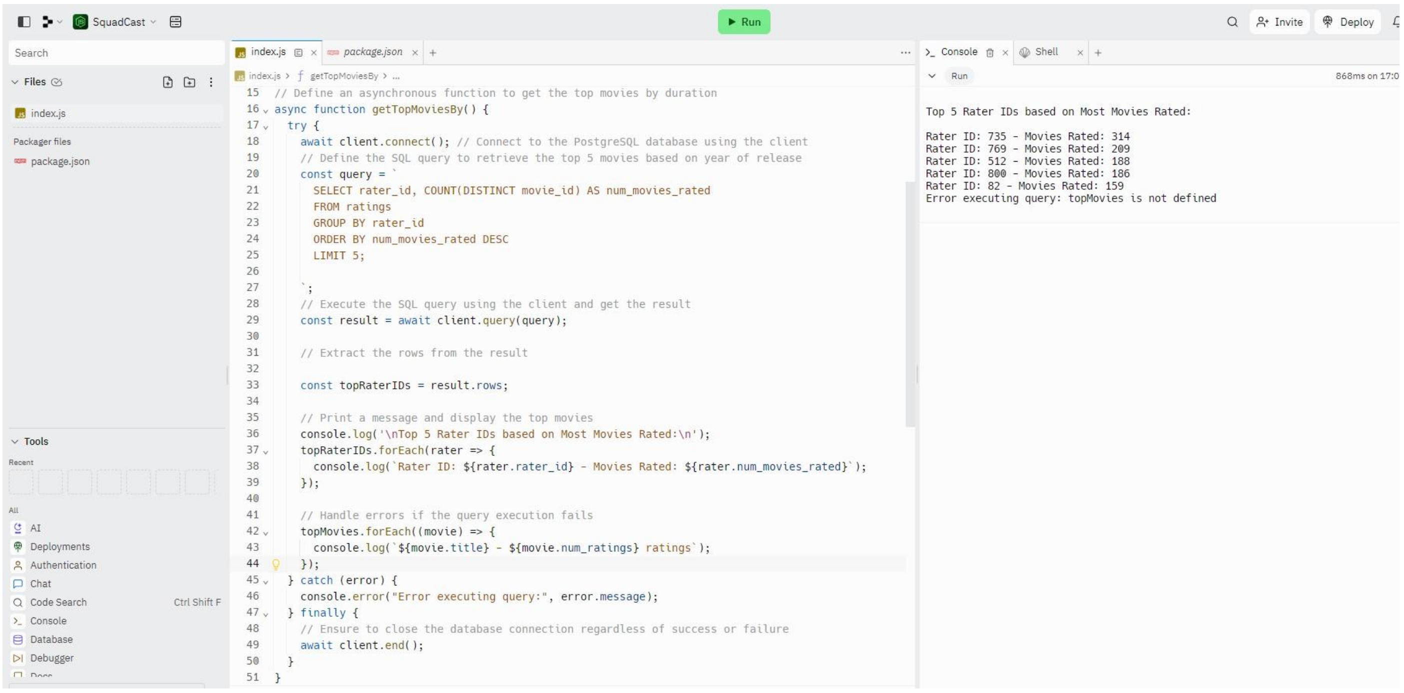
GROUP BY rater_id

ORDER BY num_movies_rated DESC

LIMIT 5;

Explanation :

- SELECT rater_id, COUNT(DISTINCT movie_id) AS num_movies_rated: This command selects the rater ID and counts the number of distinct movie IDs rated by each rater.
- FROM ratings: Specifies that the data is coming from the ratings table.
- GROUP BY rater_id: Groups the results by rater ID.
- ORDER BY num_movies_rated DESC: Orders the results in descending order based on the count of distinct movie IDs rated.
- LIMIT 5: Limits the output to the top 5 rater IDs.
- This query provides the top 5 rater IDs based on the most movies rated.



```javascript
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
      SELECT rater_id, COUNT(DISTINCT movie_id) AS num_movies_rated
      FROM ratings
      GROUP BY rater_id
      ORDER BY num_movies_rated DESC
      LIMIT 5;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const topRaterIDs = result.rows;

    // Print a message and display the top movies
    console.log('\nTop 5 Rater IDs based on Most Movies Rated:\n');
    topRaterIDs.forEach(rater => {
      console.log(`Rater ID: ${rater.rater_id} - Movies Rated: ${rater.num_movies_rated}`);
    });

    // Handle errors if the query execution fails
    topMovies.forEach((movie) => {
      console.log(`${movie.title} - ${movie.num_ratings} ratings`);
    });
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
```

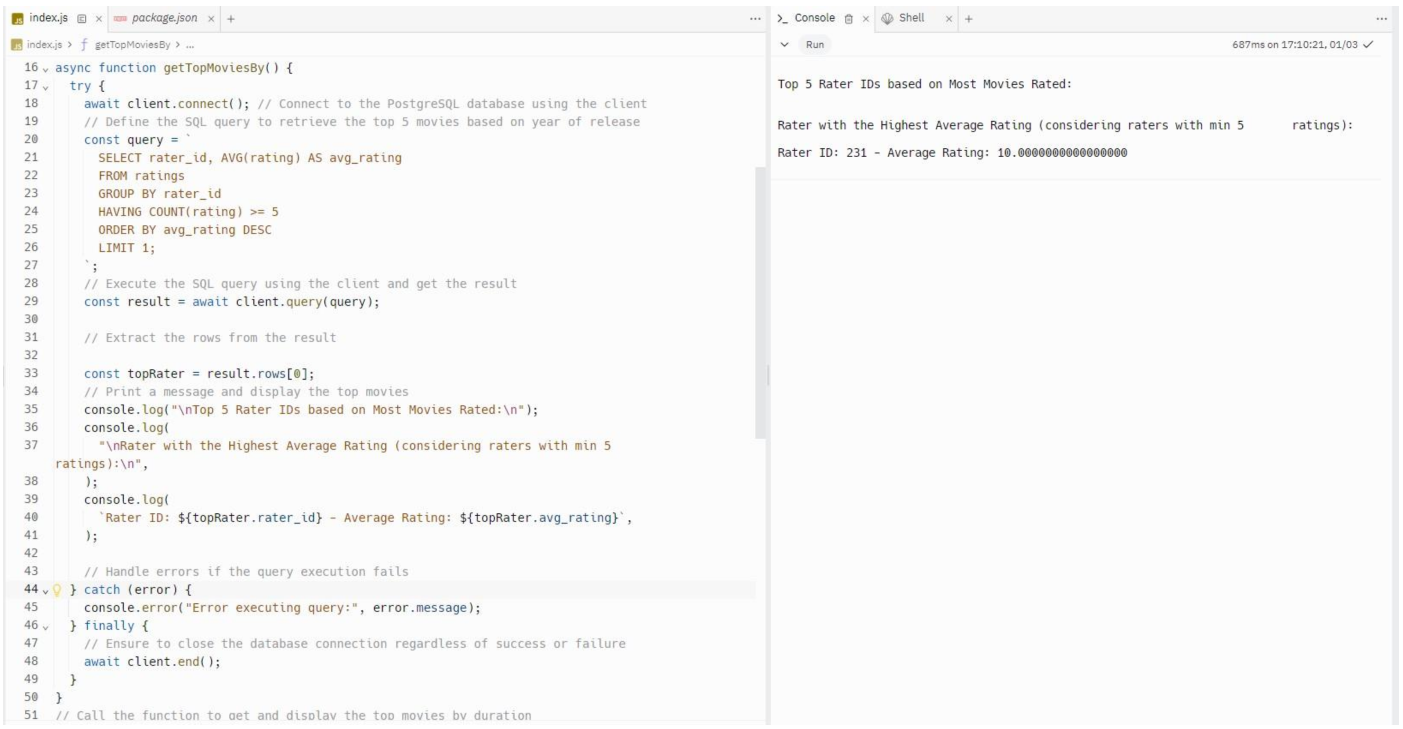
```
Top 5 Rater IDs based on Most Movies Rated:

Rater ID: 735 - Movies Rated: 314
Rater ID: 769 - Movies Rated: 209
Rater ID: 512 - Movies Rated: 188
Rater ID: 800 - Movies Rated: 186
Rater ID: 82 - Movies Rated: 159
Error executing query: topMovies is not defined
```

● Highest Average rating given (consider raters with min 5 ratings)

SELECT rater_id, AVG(rating) AS avg_rating
FROM ratings
GROUP BY rater_id
HAVING COUNT(rating) >= 5
ORDER BY avg_rating DESC
LIMIT 1;

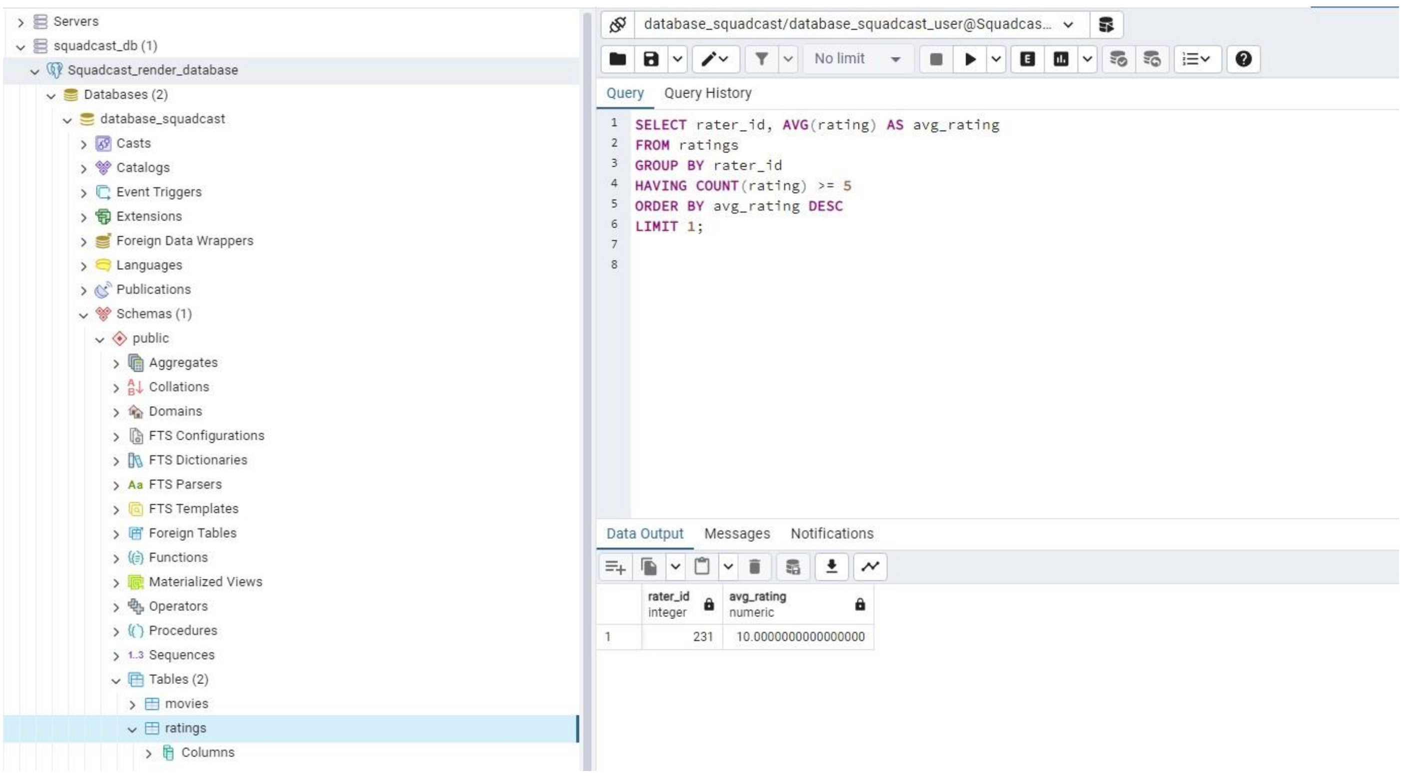
Explanation :

- SELECT rater_id, AVG(rating) AS avg_rating: This command selects the rater ID and calculates the average rating given by each rater.
- FROM ratings: Specifies that the data is coming from the ratings table.
- GROUP BY rater_id: Groups the results by rater ID.
- HAVING COUNT(rating) >= 5: Filters out raters with less than 5 ratings.
- ORDER BY avg_rating DESC: Orders the results in descending order based on the average rating.
- LIMIT 1: Limits the output to the rater with the highest average rating.



```javascript
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
      SELECT rater_id, AVG(rating) AS avg_rating
      FROM ratings
      GROUP BY rater_id
      HAVING COUNT(rating) >= 5
      ORDER BY avg_rating DESC
      LIMIT 1;
    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const topRater = result.rows[0];
    // Print a message and display the top movies
    console.log("\nTop 5 Rater IDs based on Most Movies Rated:\n");
    console.log(
      "\nRater with the Highest Average Rating (considering raters with min 5
ratings):\n",
    );
    console.log(
      `Rater ID: ${topRater.rater_id} - Average Rating: ${topRater.avg_rating}`,
    );

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
// Call the function to get and display the top movies by duration
```

```
Top 5 Rater IDs based on Most Movies Rated:

Rater with the Highest Average Rating (considering raters with min 5       ratings):

Rater ID: 231 - Average Rating: 10.0000000000000000
```



```sql
SELECT rater_id, AVG(rating) AS avg_rating
FROM ratings
GROUP BY rater_id
HAVING COUNT(rating) >= 5
ORDER BY avg_rating DESC
LIMIT 1;
```

	rater_id integer	avg_rating numeric
1	231	10.0000000000000000

d. Top Rated Movie:

- Find and print the top-rated movies by:

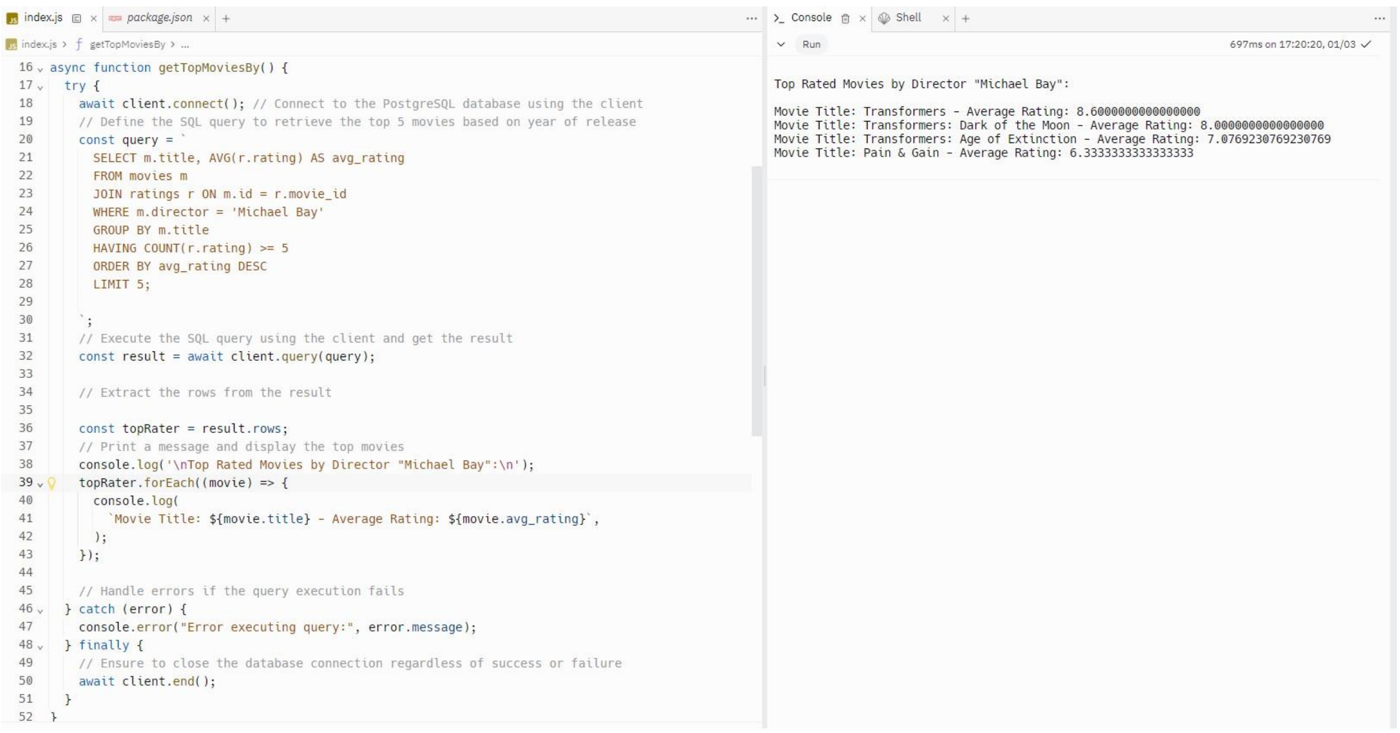
● Director 'Michael Bay',

Query :

```
SELECT m.title, AVG(r.rating) AS avg_rating
FROM movies m
JOIN ratings r ON m.id = r.movie_id
WHERE m.director = 'Michael Bay'
GROUP BY m.title
HAVING COUNT(r.rating) >= 5
ORDER BY avg_rating DESC
LIMIT 5;
;
```

Explanation :

- SELECT m.title, AVG(r.rating) AS avg_rating: This command selects the movie title and calculates the average rating for each movie.
- FROM movies m JOIN ratings r ON m.id = r.movie_id: Specifies that the data is coming from the movies and ratings tables, and it joins them based on the movie ID.
- WHERE m.director = 'Michael Bay': Filters the movies based on the director being 'Michael Bay.'
- GROUP BY m.title: Groups the results by movie title.
- HAVING COUNT(r.rating) >= 5: Filters out movies with less than 5 ratings.
- ORDER BY avg_rating DESC: Orders the results in descending order based on the average rating.
- LIMIT 5: Limits the output to the top 5 movies.



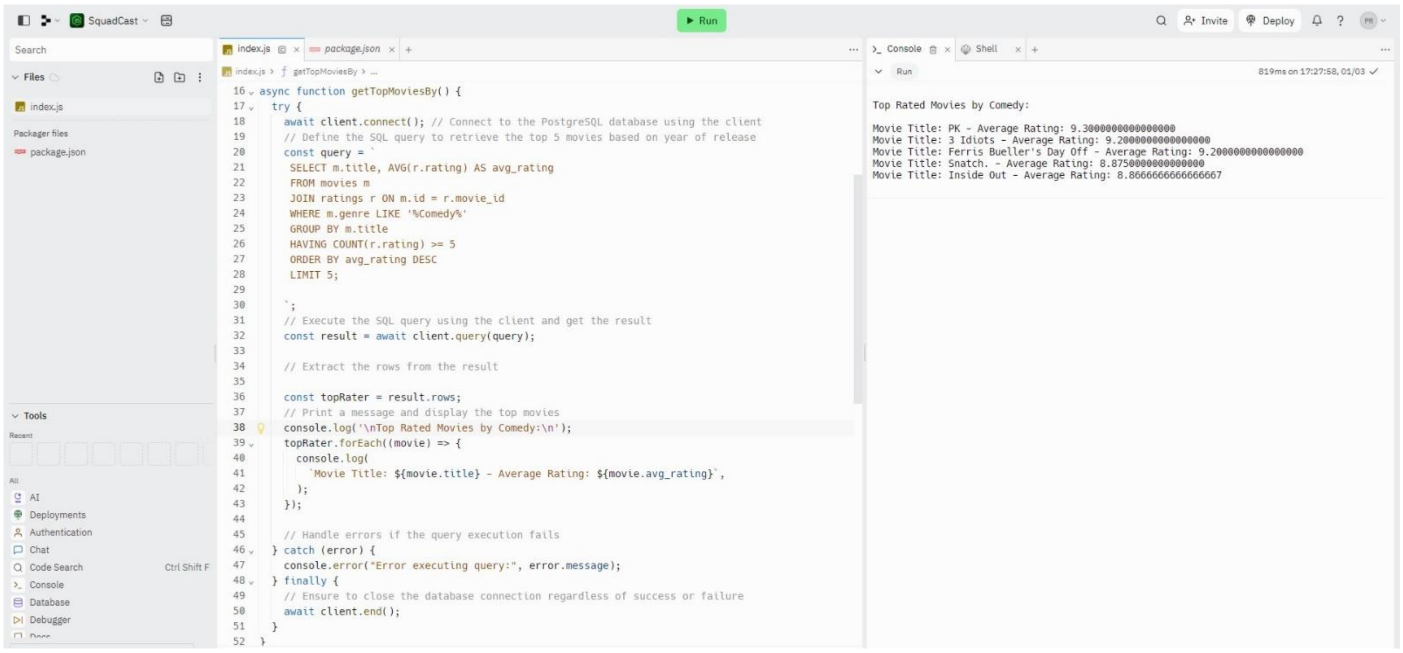
```javascript
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
      SELECT m.title, AVG(r.rating) AS avg_rating
      FROM movies m
      JOIN ratings r ON m.id = r.movie_id
      WHERE m.director = 'Michael Bay'
      GROUP BY m.title
      HAVING COUNT(r.rating) >= 5
      ORDER BY avg_rating DESC
      LIMIT 5;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const topRater = result.rows;
    // Print a message and display the top movies
    console.log('\nTop Rated Movies by Director "Michael Bay":\n');
    topRater.forEach((movie) => {
      console.log(
        `Movie Title: ${movie.title} - Average Rating: ${movie.avg_rating}`,
      );
    });

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
```

```
Top Rated Movies by Director "Michael Bay":

Movie Title: Transformers - Average Rating: 8.6000000000000000
Movie Title: Transformers: Dark of the Moon - Average Rating: 8.0000000000000000
Movie Title: Transformers: Age of Extinction - Average Rating: 7.0769230769230769
Movie Title: Pain & Gain - Average Rating: 6.3333333333333333
```

- **Comedy:**

Query :

```
SELECT m.title, AVG(r.rating) AS avg_rating
    FROM movies m
    JOIN ratings r ON m.id = r.movie_id
    WHERE m.genre LIKE '%Comedy%'
    GROUP BY m.title
    HAVING COUNT(r.rating) >= 5
    ORDER BY avg_rating DESC
    LIMIT 5;
```



```
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
     SELECT m.title, AVG(r.rating) AS avg_rating
     FROM movies m
     JOIN ratings r ON m.id = r.movie_id
     WHERE m.genre LIKE '%Comedy%'
     GROUP BY m.title
     HAVING COUNT(r.rating) >= 5
     ORDER BY avg_rating DESC
     LIMIT 5;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const topRater = result.rows;
    // Print a message and display the top movies
    console.log('\nTop Rated Movies by Comedy:\n');
    topRater.forEach((movie) => {
      console.log(
        `Movie Title: ${movie.title} - Average Rating: ${movie.avg_rating}`,
      );
    });

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
```

```
Top Rated Movies by Comedy:

Movie Title: PK - Average Rating: 9.3000000000000000
Movie Title: 3 Idiots - Average Rating: 9.2000000000000000
Movie Title: Ferris Bueller's Day Off - Average Rating: 9.2000000000000000
Movie Title: Snatch. - Average Rating: 8.8750000000000000
Movie Title: Inside Out - Average Rating: 8.8666666666666667
```

- Selection: It selects the movie title (m.title) and the average rating (AVG(r.rating)) for each movie.
- Data Source: It retrieves data from the movies and ratings tables, joining them based on the movie ID.
- Filtering: It includes only movies in the "Comedy" genre.
- Grouping: It groups the results by movie title to apply aggregate functions.
- Condition: It filters out movies with fewer than 5 ratings.
- Sorting: It orders the results by average rating in descending order.
- Limiting: It limits the output to the top 5 movies.

- **In the year 2013**

Query :

// Define the SQL query to get the top-rated movies in the year 2013:

SELECT m.title, AVG(r.rating) AS avg_rating

FROM movies m

JOIN ratings r ON m.id = r.movie_id

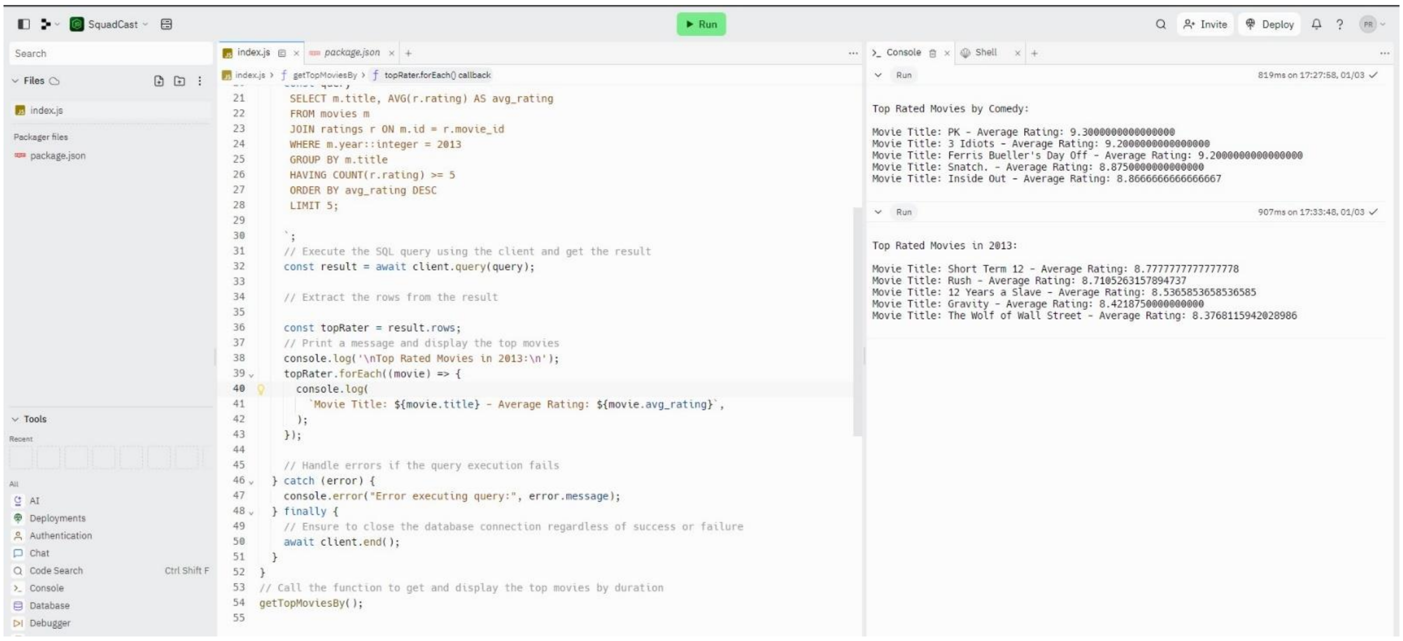
WHERE m.year::integer = 2013

GROUP BY m.title

HAVING COUNT(r.rating) >= 5

ORDER BY avg_rating DESC

LIMIT 5;



```
      SELECT m.title, AVG(r.rating) AS avg_rating
      FROM movies m
      JOIN ratings r ON m.id = r.movie_id
      WHERE m.year::integer = 2013
      GROUP BY m.title
      HAVING COUNT(r.rating) >= 5
      ORDER BY avg_rating DESC
      LIMIT 5;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const topRater = result.rows;
    // Print a message and display the top movies
    console.log('\nTop Rated Movies in 2013:\n');
    topRater.forEach((movie) => {
      console.log(
        `Movie Title: ${movie.title} - Average Rating: ${movie.avg_rating}`,
      );
    });

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
// Call the function to get and display the top movies by duration
getTopMoviesBy();
```

```
Top Rated Movies by Comedy:

Movie Title: PK - Average Rating: 9.3000000000000000
Movie Title: 3 Idiots - Average Rating: 9.2000000000000000
Movie Title: Ferris Bueller's Day Off - Average Rating: 9.2000000000000000
Movie Title: Snatch. - Average Rating: 8.8750000000000000
Movie Title: Inside Out - Average Rating: 8.8666666666666667
```

```
Top Rated Movies in 2013:

Movie Title: Short Term 12 - Average Rating: 8.7777777777777778
Movie Title: Rush - Average Rating: 8.7105263157894737
Movie Title: 12 Years a Slave - Average Rating: 8.5365853658536585
Movie Title: Gravity - Average Rating: 8.4218750000000000
Movie Title: The Wolf of Wall Street - Average Rating: 8.3768115942028986
```

- **In India (consider movies with a minimum of 5 ratings).**

Const Query:

SELECT m.title, AVG(r.rating) AS avg_rating

FROM movies m

JOIN ratings r ON m.id = r.movie_id

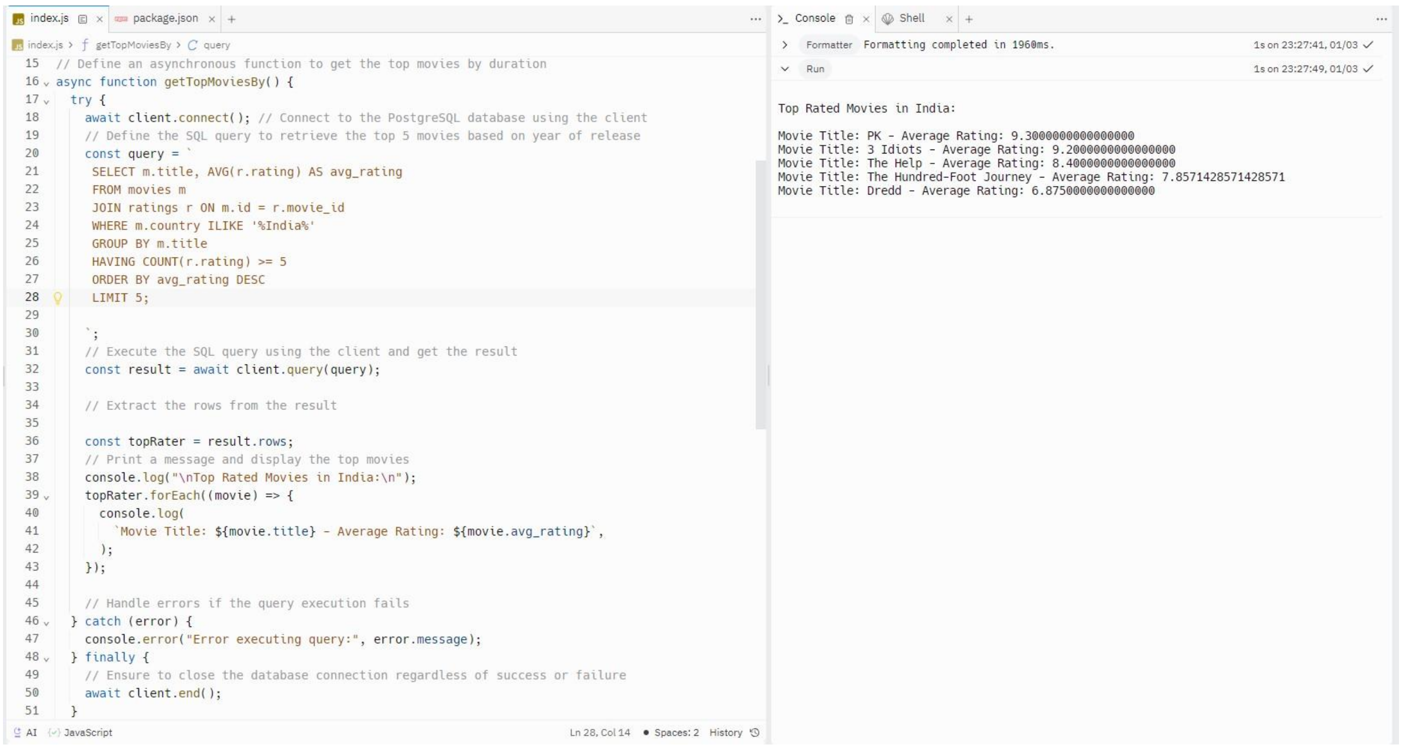
WHERE m.country ILIKE '%India%'

GROUP BY m.title

HAVING COUNT(r.rating) >= 5

ORDER BY avg_rating DESC

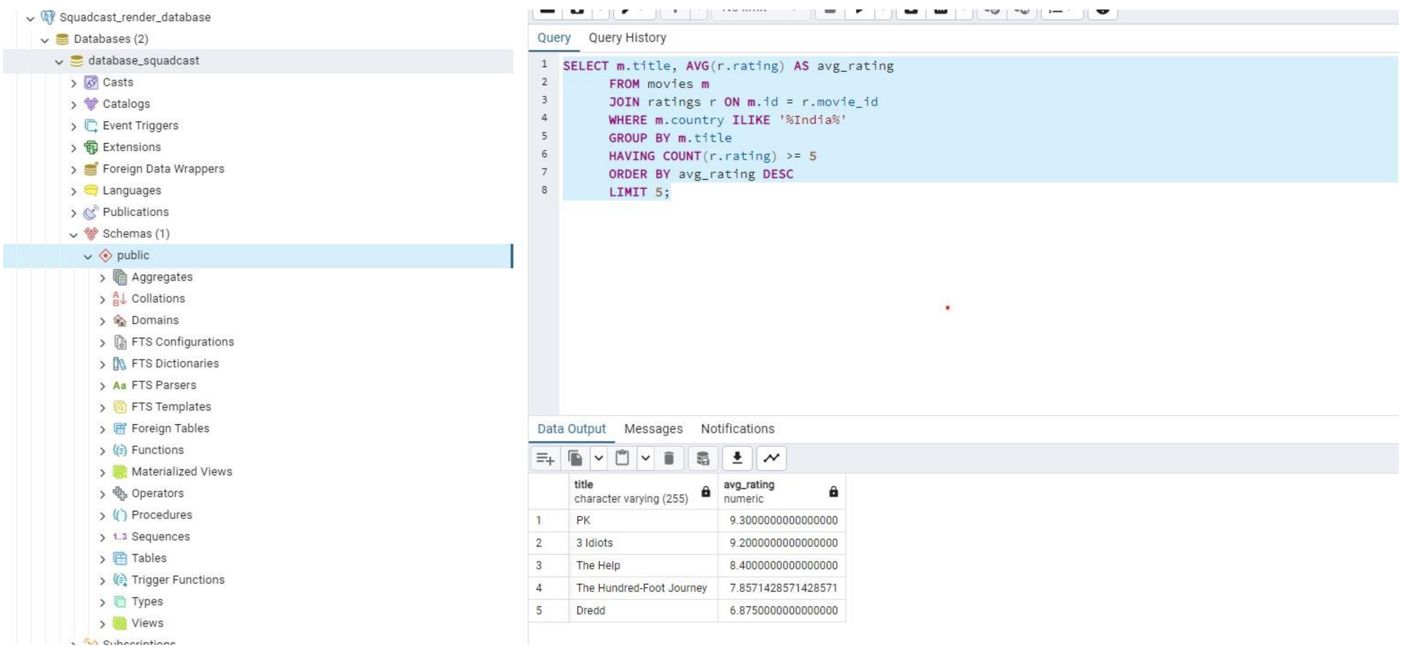
LIMIT 5;



```
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
     SELECT m.title, AVG(r.rating) AS avg_rating
     FROM movies m
     JOIN ratings r ON m.id = r.movie_id
     WHERE m.country ILIKE '%India%'
     GROUP BY m.title
     HAVING COUNT(r.rating) >= 5
     ORDER BY avg_rating DESC
     LIMIT 5;

    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const topRater = result.rows;
    // Print a message and display the top movies
    console.log("\nTop Rated Movies in India:\n");
    topRater.forEach((movie) => {
      console.log(
        `Movie Title: ${movie.title} - Average Rating: ${movie.avg_rating}`,
      );
    });

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
```

```
Formatter  Formatting completed in 1960ms.        1s on 23:27:41, 01/03
Run                                               1s on 23:27:49, 01/03

Top Rated Movies in India:

Movie Title: PK - Average Rating: 9.3000000000000000
Movie Title: 3 Idiots - Average Rating: 9.2000000000000000
Movie Title: The Help - Average Rating: 8.4000000000000000
Movie Title: The Hundred-Foot Journey - Average Rating: 7.8571428571428571
Movie Title: Dredd - Average Rating: 6.8750000000000000
```

Cross Check :



```
SELECT m.title, AVG(r.rating) AS avg_rating
      FROM movies m
      JOIN ratings r ON m.id = r.movie_id
      WHERE m.country ILIKE '%India%'
      GROUP BY m.title
      HAVING COUNT(r.rating) >= 5
      ORDER BY avg_rating DESC
      LIMIT 5;
```

	title character varying (255)	avg_rating numeric
1	PK	9.3000000000000000
2	3 Idiots	9.2000000000000000
3	The Help	8.4000000000000000
4	The Hundred-Foot Journey	7.8571428571428571
5	Dredd	6.8750000000000000

e. Favourite Movie Genre of Rater ID 1040: Determine and print the favorite movie genre for the rater with ID 1040 (defined as the genre of the movie the rater has rated most often).

Query:

SELECT m.genre, COUNT(*) AS genre_count

FROM movies m

JOIN ratings r ON m.id = r.movie_id

WHERE r.rater_id = 1040

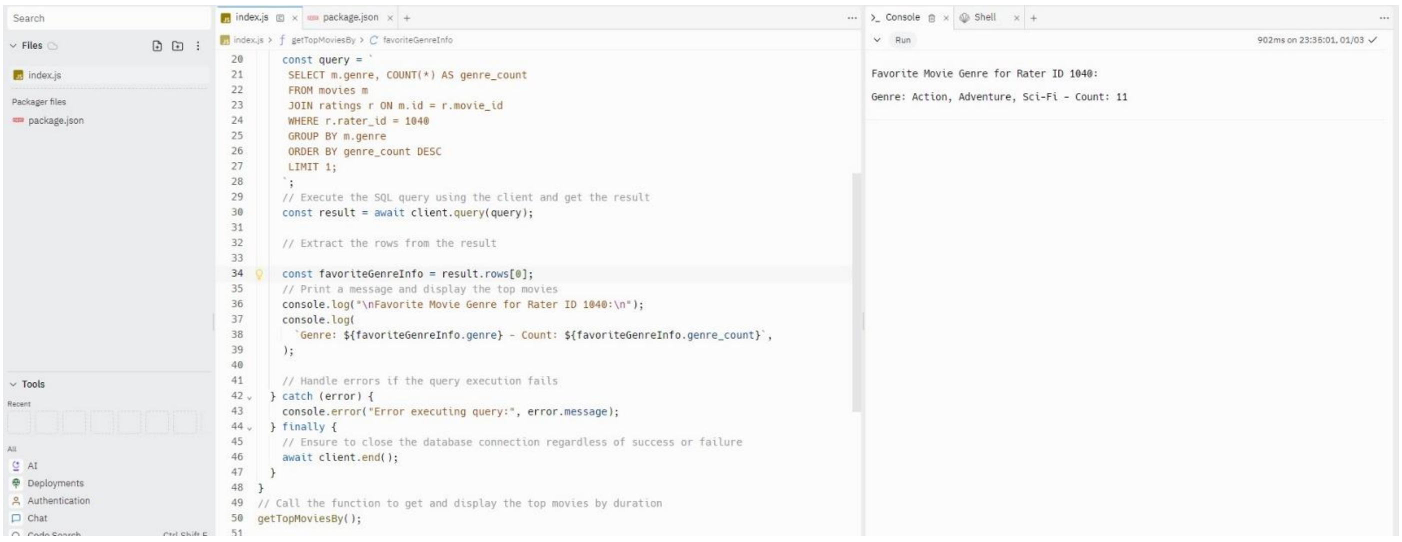
GROUP BY m.genre

ORDER BY genre_count DESC

LIMIT 1;

Explanation :

- Selection: It selects the movie genre (m.genre) and counts the occurrences for each genre (COUNT(*)).
- Data Source: It retrieves data from the movies and ratings tables, joining them based on the movie ID.
- Filtering: It includes only ratings given by the specific rater with ID 1040 (r.rater_id = 1040).
- Grouping: It groups the results by movie genre to count the occurrences for each genre.
- Sorting: It orders the results by the count of genres in descending order.



```
    const query = `
     SELECT m.genre, COUNT(*) AS genre_count
     FROM movies m
     JOIN ratings r ON m.id = r.movie_id
     WHERE r.rater_id = 1040
     GROUP BY m.genre
     ORDER BY genre_count DESC
     LIMIT 1;
    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const favoriteGenreInfo = result.rows[0];
    // Print a message and display the top movies
    console.log("\nFavorite Movie Genre for Rater ID 1040:\n");
    console.log(
      `Genre: ${favoriteGenreInfo.genre} - Count: ${favoriteGenreInfo.genre_count}`,
    );

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
// Call the function to get and display the top movies by duration
getTopMoviesBy();
```

```
Favorite Movie Genre for Rater ID 1040:

Genre: Action, Adventure, Sci-Fi - Count: 11
```

f. Highest Average Rating for a Movie Genre by Rater ID 1040: Find and print the highest average rating for a movie genre given by the rater with ID 1040 (consider genres with a minimum of 5 ratings)

Query:

SELECT m.genre, AVG(r.rating) AS avg_rating

    FROM movies m

    JOIN ratings r ON m.id = r.movie_id

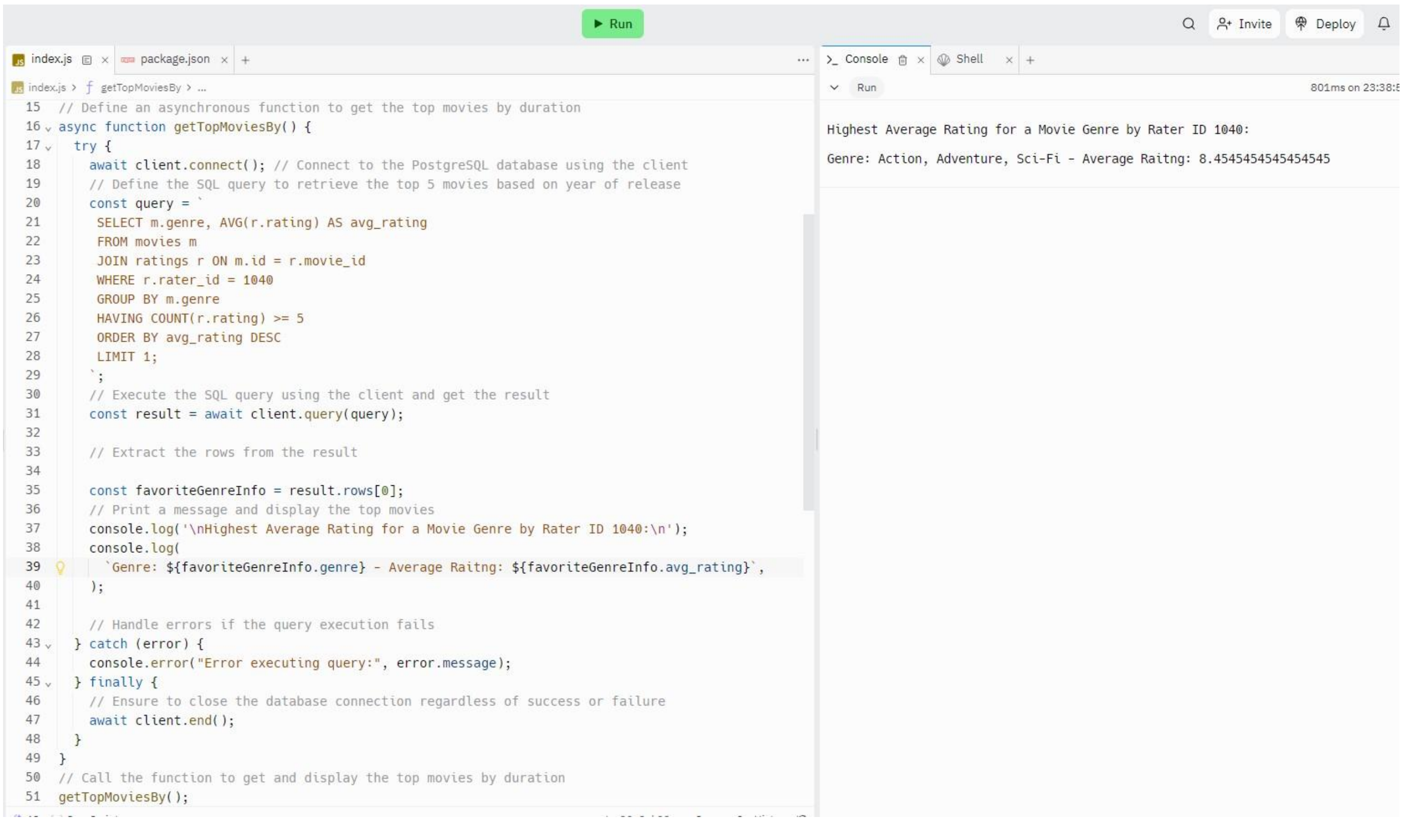
    WHERE r.rater_id = 1040

    GROUP BY m.genre

    HAVING COUNT(r.rating) >= 5

    ORDER BY avg_rating DESC

    LIMIT 1;



```
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
     SELECT m.genre, AVG(r.rating) AS avg_rating
     FROM movies m
     JOIN ratings r ON m.id = r.movie_id
     WHERE r.rater_id = 1040
     GROUP BY m.genre
     HAVING COUNT(r.rating) >= 5
     ORDER BY avg_rating DESC
     LIMIT 1;
    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const favoriteGenreInfo = result.rows[0];
    // Print a message and display the top movies
    console.log('\nHighest Average Rating for a Movie Genre by Rater ID 1040:\n');
    console.log(
      `Genre: ${favoriteGenreInfo.genre} - Average Raitng: ${favoriteGenreInfo.avg_rating}`,
    );

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
// Call the function to get and display the top movies by duration
getTopMoviesBy();
```

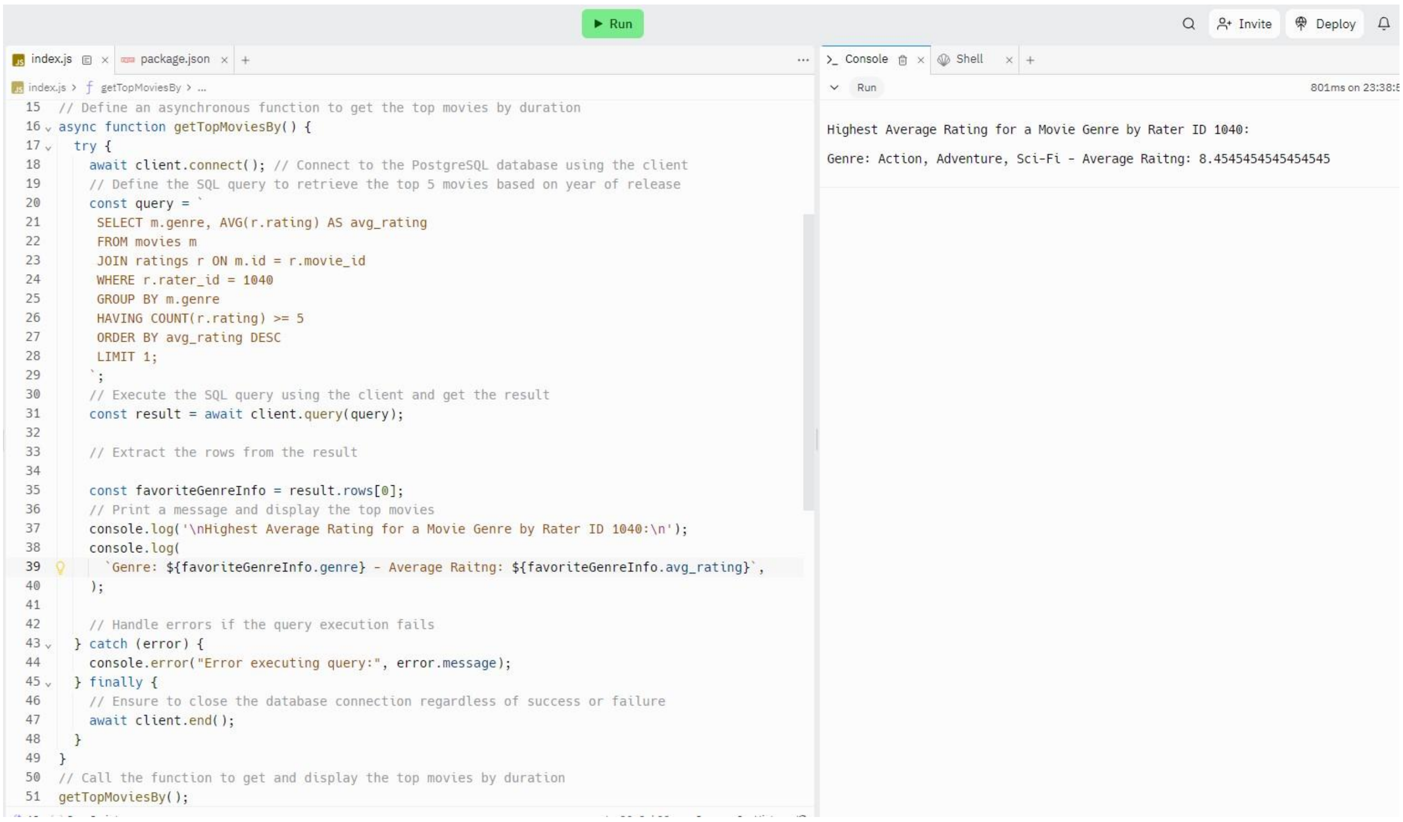
```
Highest Average Rating for a Movie Genre by Rater ID 1040:

Genre: Action, Adventure, Sci-Fi - Average Raitng: 8.4545454545454545
```

g. Year with Second-Highest Number of Action Movies: Identify and print the year with the second-highest number of action movies from the USA that received an average rating of 6.5 or higher and had a runtime of less than 120 minutes.

Query :

```
SELECT m.year, COUNT(*) AS action_movie_count

FROM movies m

WHERE m.country ILIKE '%USA%'

AND m.genre ILIKE '%Action%'

AND m.minutes < 120

GROUP BY m.year

HAVING AVG((SELECT AVG(rating) FROM ratings r WHERE r.movie_id = m.id)) >= 6.5

ORDER BY action_movie_count DESC

OFFSET 1

LIMIT 1;
```

To identify and print the year with the second-highest number of action movies from the USA that received an average rating of 6.5 or higher and had a runtime of less than 120 minutes



```
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
     SELECT m.genre, AVG(r.rating) AS avg_rating
     FROM movies m
     JOIN ratings r ON m.id = r.movie_id
     WHERE r.rater_id = 1040
     GROUP BY m.genre
     HAVING COUNT(r.rating) >= 5
     ORDER BY avg_rating DESC
     LIMIT 1;
    `;
    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Extract the rows from the result

    const favoriteGenreInfo = result.rows[0];
    // Print a message and display the top movies
    console.log('\nHighest Average Rating for a Movie Genre by Rater ID 1040:\n');
    console.log(
      `Genre: ${favoriteGenreInfo.genre} - Average Raitng: ${favoriteGenreInfo.avg_rating}`,
    );

    // Handle errors if the query execution fails
  } catch (error) {
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
  }
}
// Call the function to get and display the top movies by duration
getTopMoviesBy();
```

```
Highest Average Rating for a Movie Genre by Rater ID 1040:

Genre: Action, Adventure, Sci-Fi - Average Raitng: 8.4545454545454545
```

h. Count of Movies with High Ratings: Count and print the number of movies that have received at least five reviews with a rating of 7 or higher.

SELECT m.title, COUNT(r.movie_id) AS review_count

FROM movies m

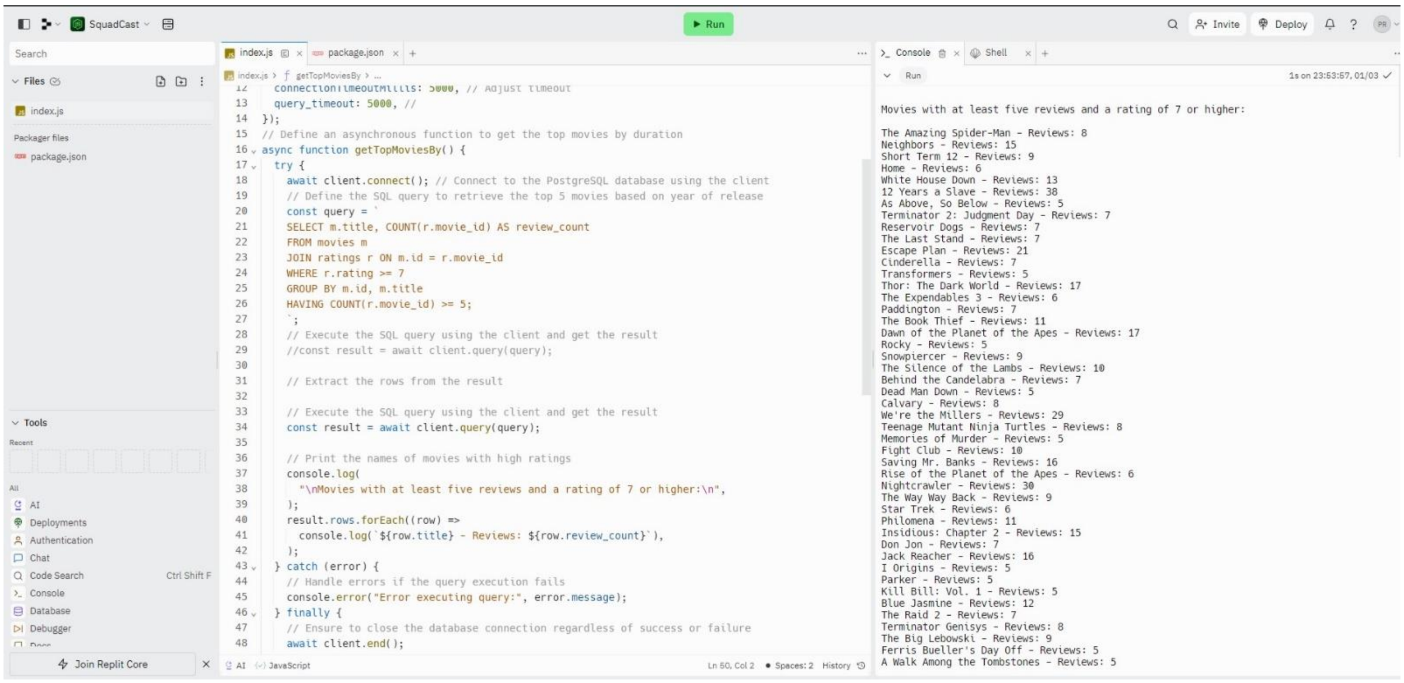
JOIN ratings r ON m.id = r.movie_id

WHERE r.rating >= 7

GROUP BY m.id, m.title

HAVING COUNT(r.movie_id) >= 5;

Explanation: This query selects the movie title (m.title) and counts the number of reviews for each movie where the associated ratings are 7 or higher. It groups the results by movie ID and title, then filters to include only movies that have received at least five reviews with a rating of 7 or higher.



```
    connectionTimeoutMillis: 5000, // Adjust timeout
    query_timeout: 5000, //
});
// Define an asynchronous function to get the top movies by duration
async function getTopMoviesBy() {
  try {
    await client.connect(); // Connect to the PostgreSQL database using the client
    // Define the SQL query to retrieve the top 5 movies based on year of release
    const query = `
    SELECT m.title, COUNT(r.movie_id) AS review_count
    FROM movies m
    JOIN ratings r ON m.id = r.movie_id
    WHERE r.rating >= 7
    GROUP BY m.id, m.title
    HAVING COUNT(r.movie_id) >= 5;
    `;
    // Execute the SQL query using the client and get the result
    //const result = await client.query(query);

    // Extract the rows from the result

    // Execute the SQL query using the client and get the result
    const result = await client.query(query);

    // Print the names of movies with high ratings
    console.log(
      "\nMovies with at least five reviews and a rating of 7 or higher:\n",
    );
    result.rows.forEach((row) =>
      console.log(`${row.title} - Reviews: ${row.review_count}`),
    );
  } catch (error) {
    // Handle errors if the query execution fails
    console.error("Error executing query:", error.message);
  } finally {
    // Ensure to close the database connection regardless of success or failure
    await client.end();
```

```
Movies with at least five reviews and a rating of 7 or higher:

The Amazing Spider-Man - Reviews: 8
Neighbors - Reviews: 15
Short Term 12 - Reviews: 9
Home - Reviews: 6
White House Down - Reviews: 13
12 Years a Slave - Reviews: 38
As Above, So Below - Reviews: 5
Terminator 2: Judgment Day - Reviews: 7
Reservoir Dogs - Reviews: 7
The Last Stand - Reviews: 7
Escape Plan - Reviews: 21
Cinderella - Reviews: 7
Transformers - Reviews: 5
Thor: The Dark World - Reviews: 17
The Expendables 3 - Reviews: 6
Paddington - Reviews: 7
The Book Thief - Reviews: 11
Dawn of the Planet of the Apes - Reviews: 17
Rocky - Reviews: 5
Snowpiercer - Reviews: 9
The Silence of the Lambs - Reviews: 10
Behind the Candelabra - Reviews: 7
Dead Man Down - Reviews: 5
Calvary - Reviews: 8
We're the Millers - Reviews: 29
Teenage Mutant Ninja Turtles - Reviews: 8
Memories of Murder - Reviews: 5
Fight Club - Reviews: 10
Saving Mr. Banks - Reviews: 16
Rise of the Planet of the Apes - Reviews: 6
Nightcrawler - Reviews: 30
The Way Way Back - Reviews: 9
Star Trek - Reviews: 6
Philomena - Reviews: 11
Insidious: Chapter 2 - Reviews: 15
Don Jon - Reviews: 7
Jack Reacher - Reviews: 16
I Origins - Reviews: 5
Parker - Reviews: 5
Kill Bill: Vol. 1 - Reviews: 5
Blue Jasmine - Reviews: 12
The Raid 2 - Reviews: 7
Terminator Genisys - Reviews: 8
The Big Lebowski - Reviews: 9
Ferris Bueller's Day Off - Reviews: 5
A Walk Among the Tombstones - Reviews: 5
```

So that's All the questions and queries were performed by me (Pradeep Sajnani), I actively engaged in tasks related to PostgreSQL database management, data analysis, and communication. For any further questions or clarifications, I can be reached at mailto:pradeepsajnani742@gmail.com. Feel free to connect with me for continued collaboration and assistance.

references : https://replit.com/@PradeepSajnani/SquadCast?v=1#index.js

Note * In this all the queries with output are saved you can check it out for further clarifications

---Thank You ---