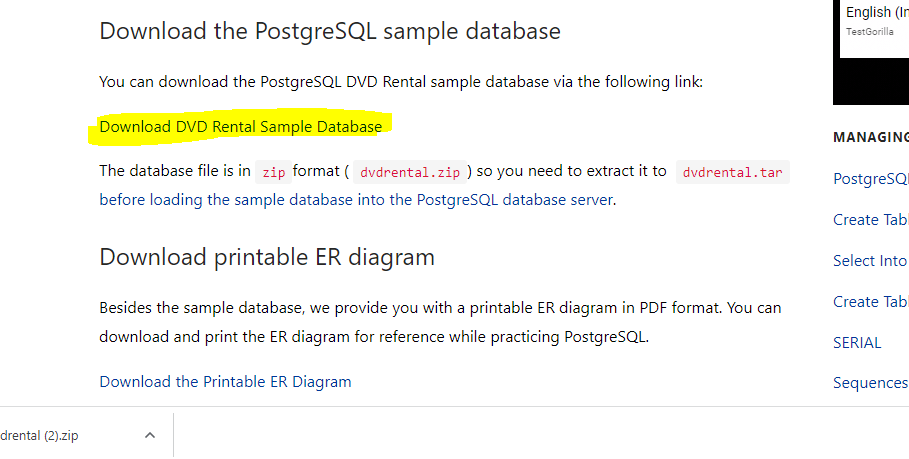
**Title of the project**

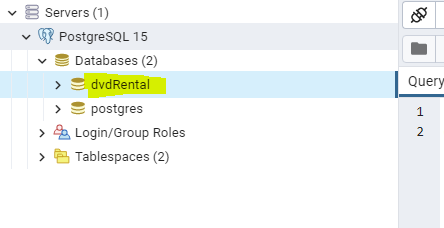
**Building Data Model and Writing ETL Job**

* **Extract the data from any preferred location – for this project we have extracted data from the below link:**

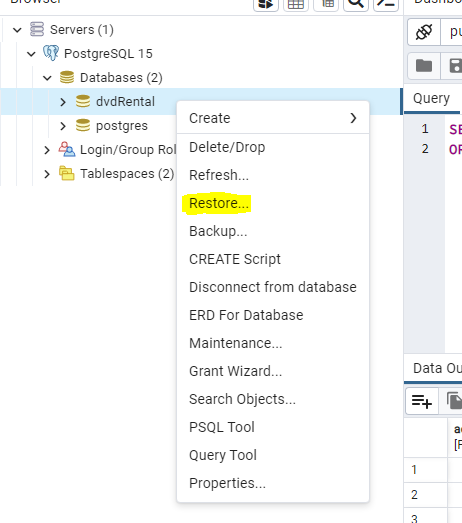
<https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/>



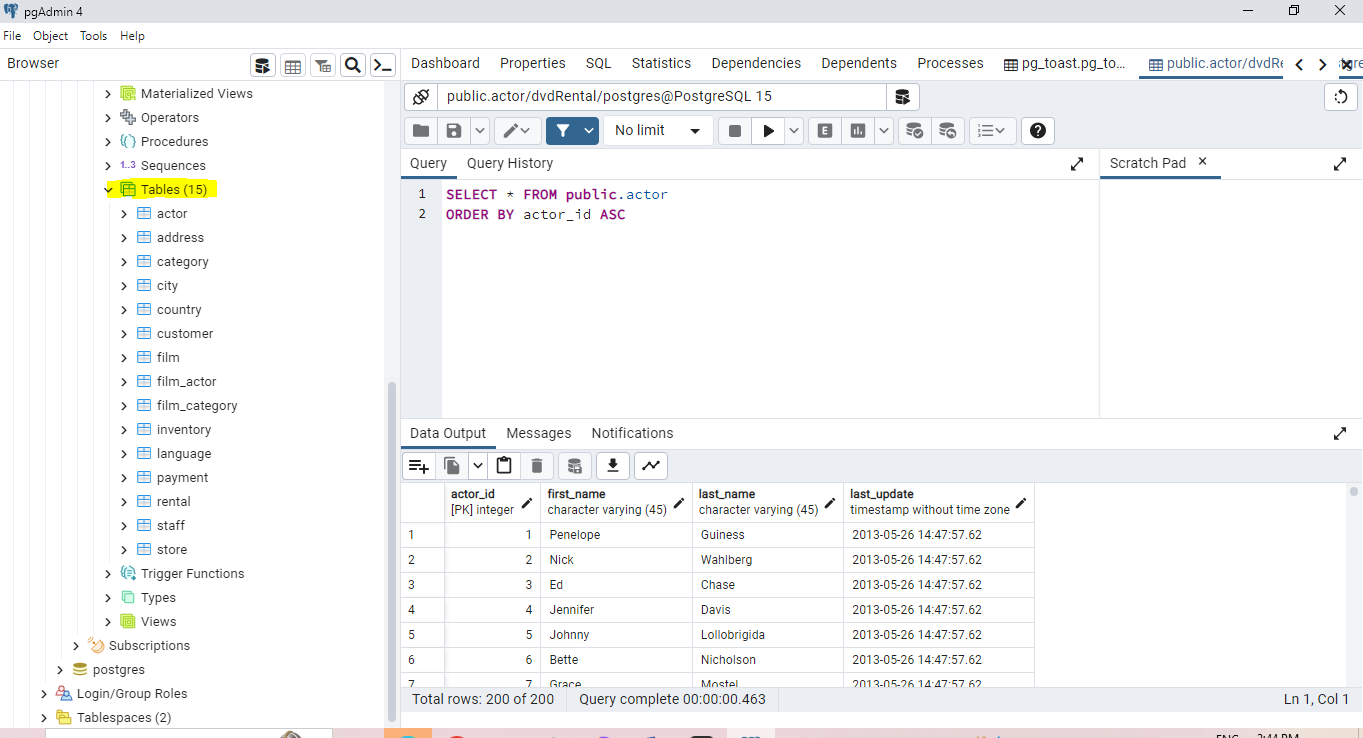
* Click on download DVD Rental Sample Database.
* Open pgAdmin and click on database – Create (to create a new database)
* Now create a new database with the name “dvdRental”.



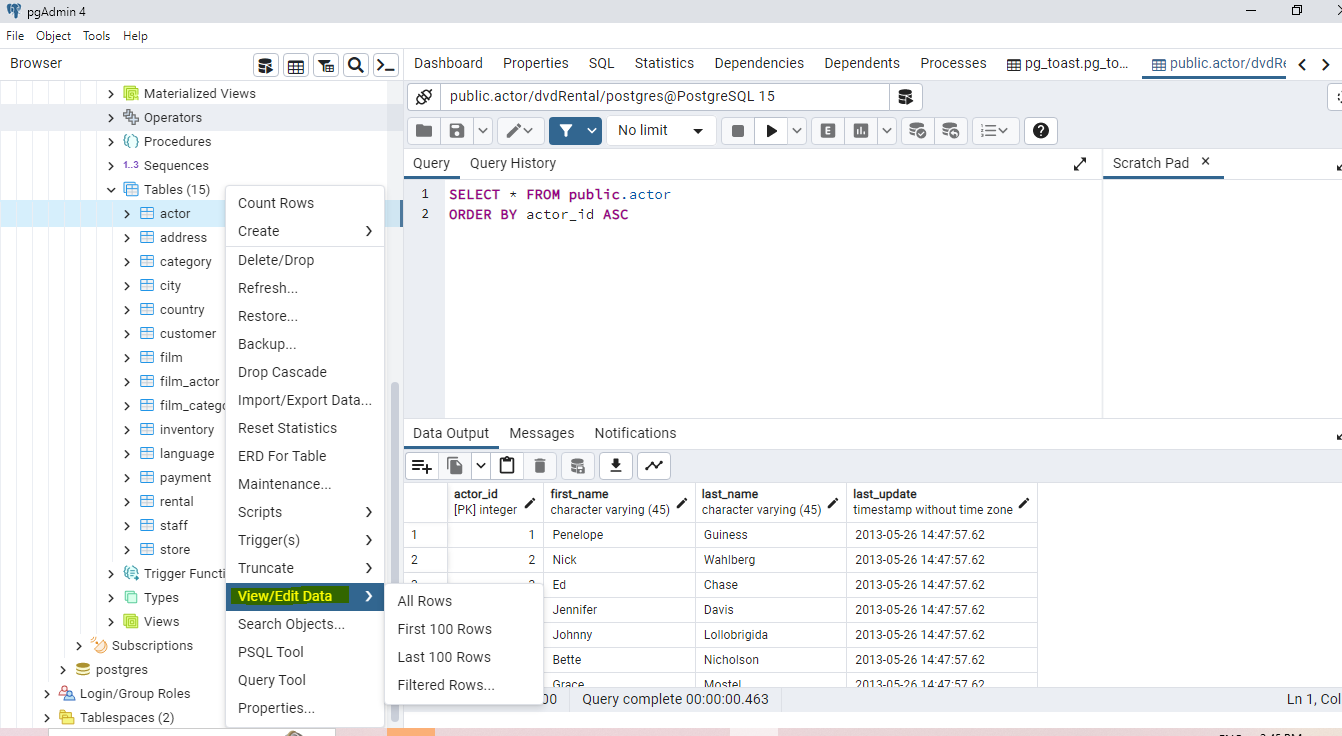
* To load the data into the database -> right click on dvdRental -> and select the option restore.



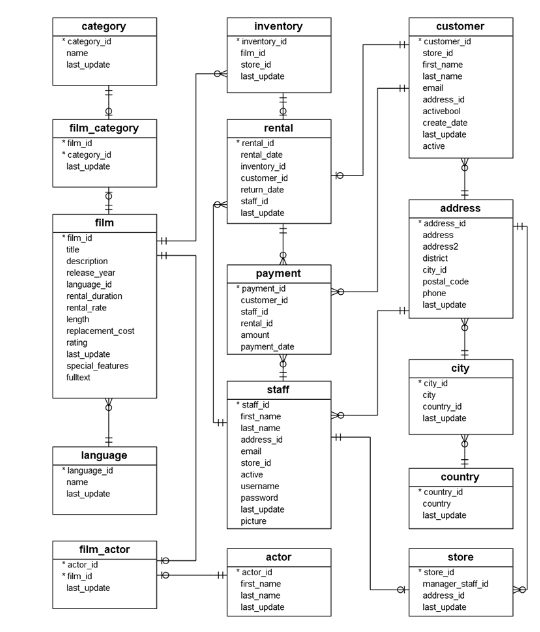
* Click on Restore and provide the file path.
* Click on refresh.
* Now go to schemas and click on tables.



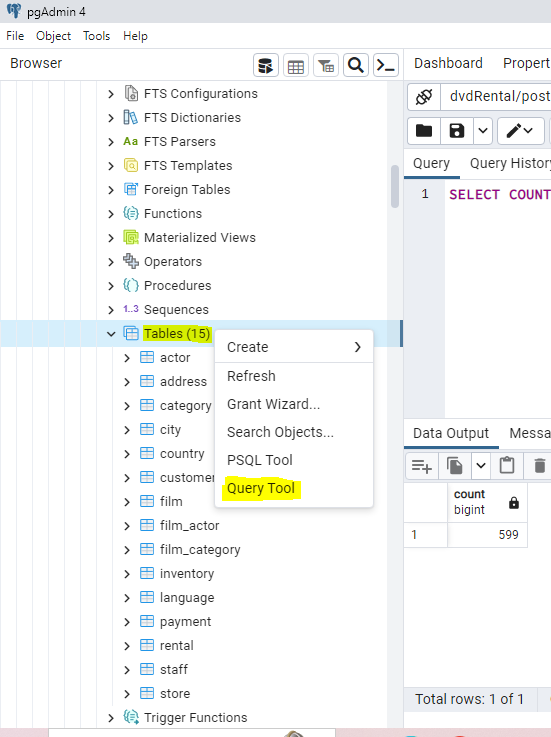
* To view the data in the table, click on -> view/edit data.



* The schema design of dvdRental is as shown below:

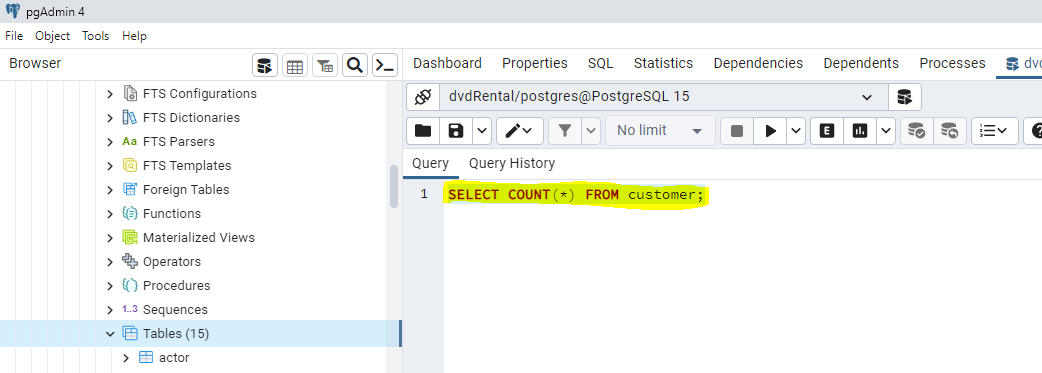


* In the above ER diagram, we have different-different tables connected based on primary key and foreign key.
* To view the tables go to pgAdmin -> click on tables -> Query Tool.

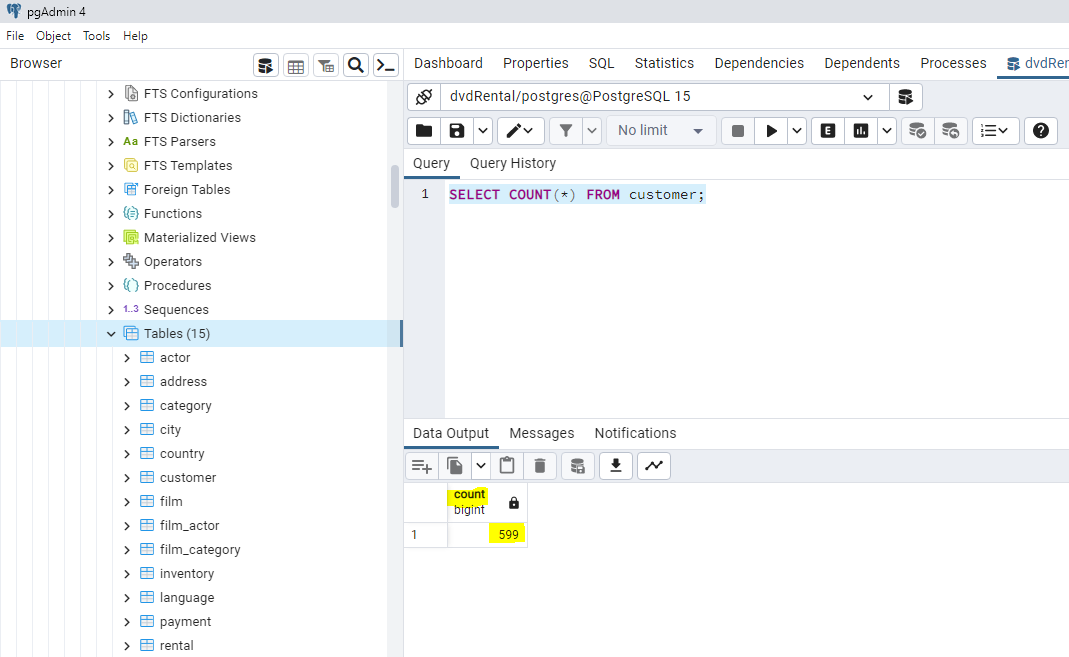


* To get the count of the columns in the customer table we can use the below query.

SELECT COUNT(\*) FROM customer;



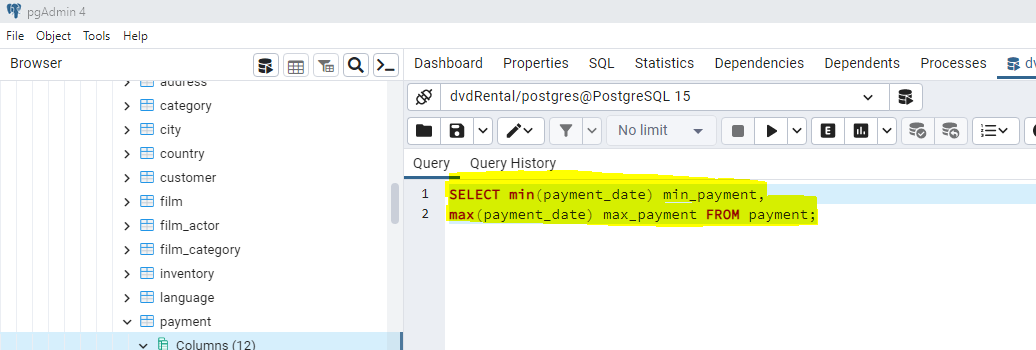
* Click on RUN to see the output (count of columns present in the customer table).



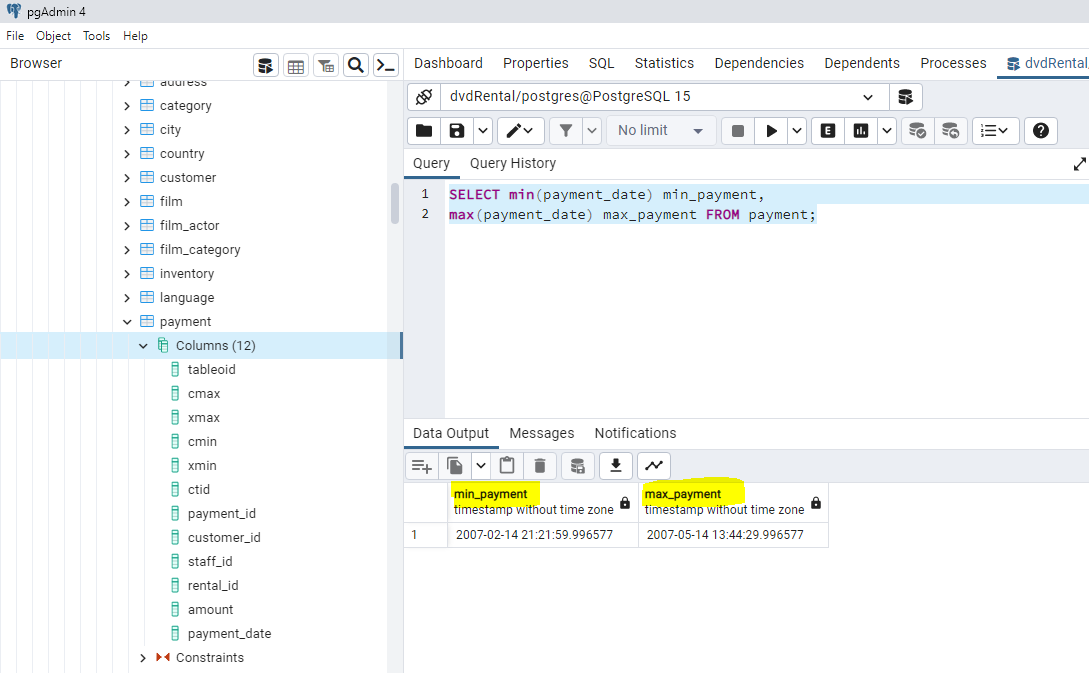
* To select the min and max date for the payment use the below command.

SELECT min(payment\_date) min\_payment,

max(payment\_date) max\_payment FROM payment;



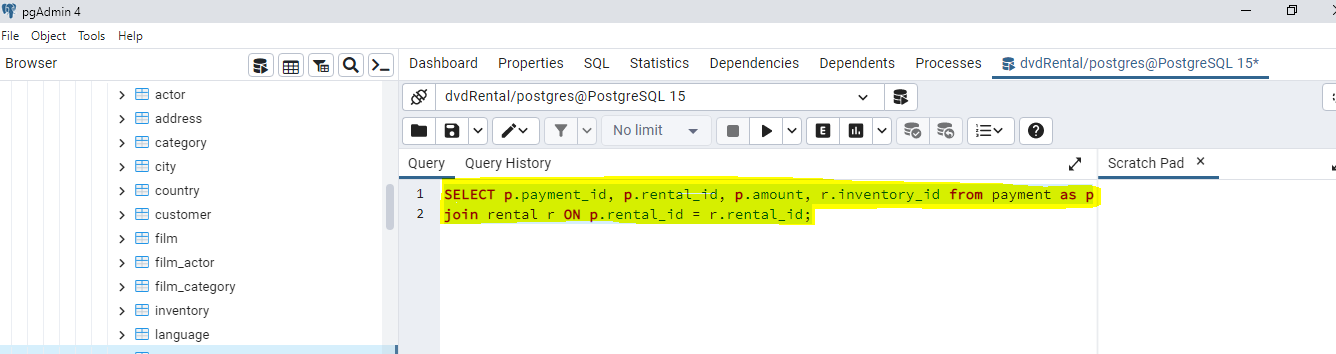
* Click on RUN command to see the below output.



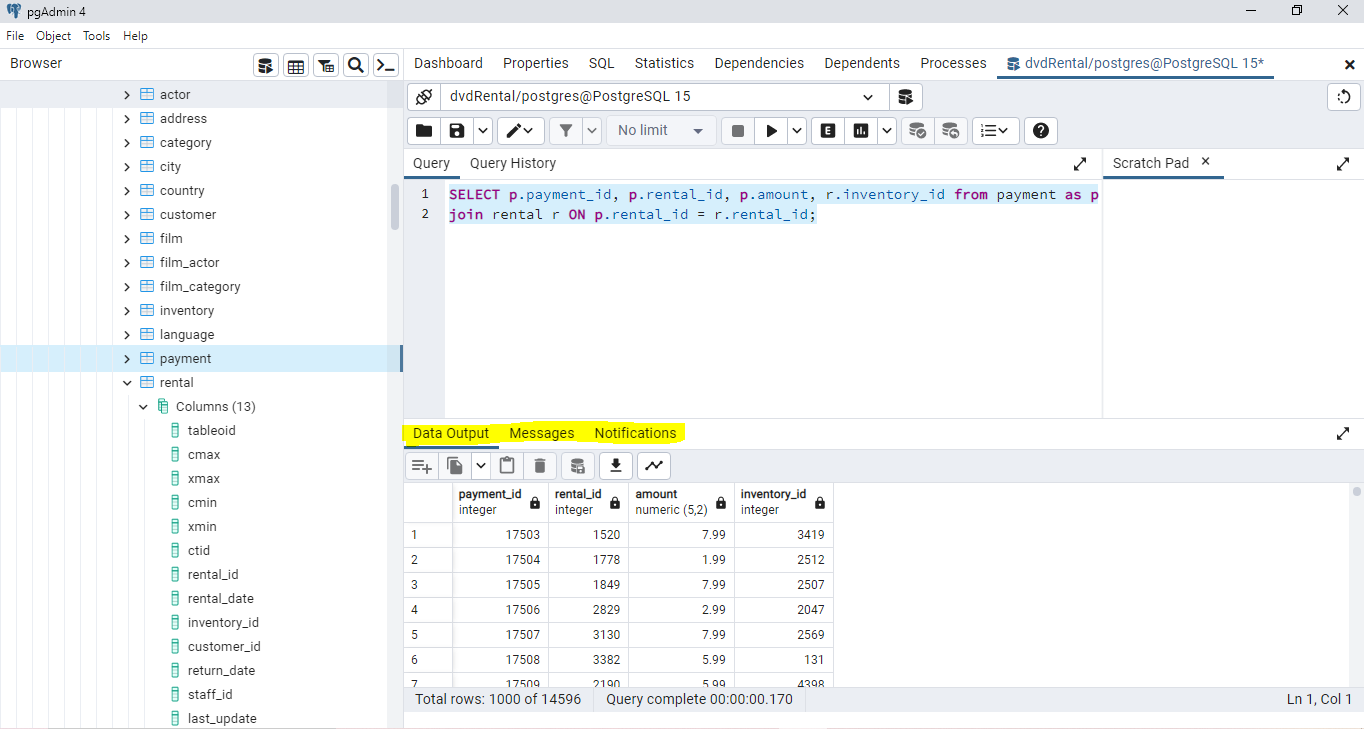
* If we want to get the payment information from the film section then we have to join both the tables however as we see that there is no common connection between the tables we have to connect the tables with the help of multiple joins as shown below.
* First join payment table and rental table on rental\_id using the below code.

SELECT p.payment\_id, p.rental\_id, p.amount, r.inventory\_id from payment as p

join rental r ON p.rental\_id = r.rental\_id;



* Run the command to get the below output.

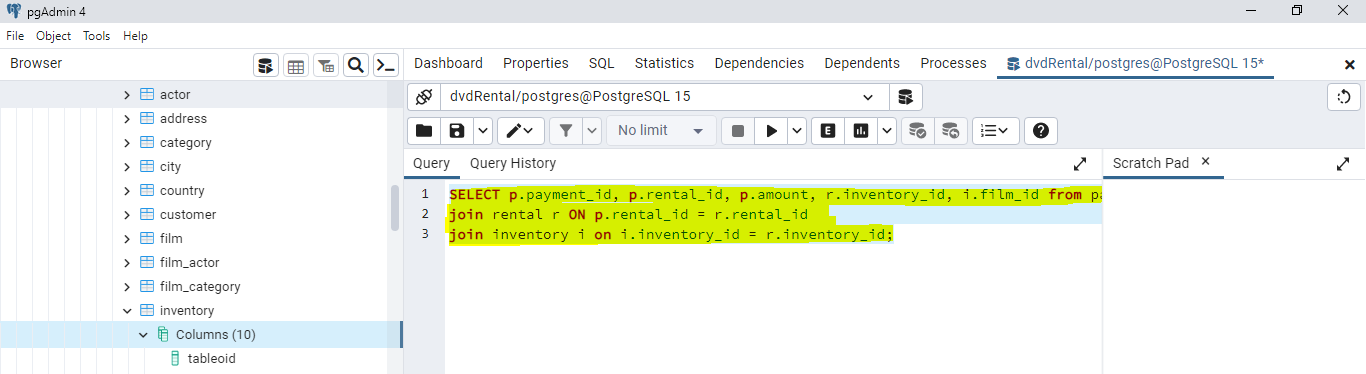


* Next, we need to join rental table with inventory table on inventory\_id using the below code:

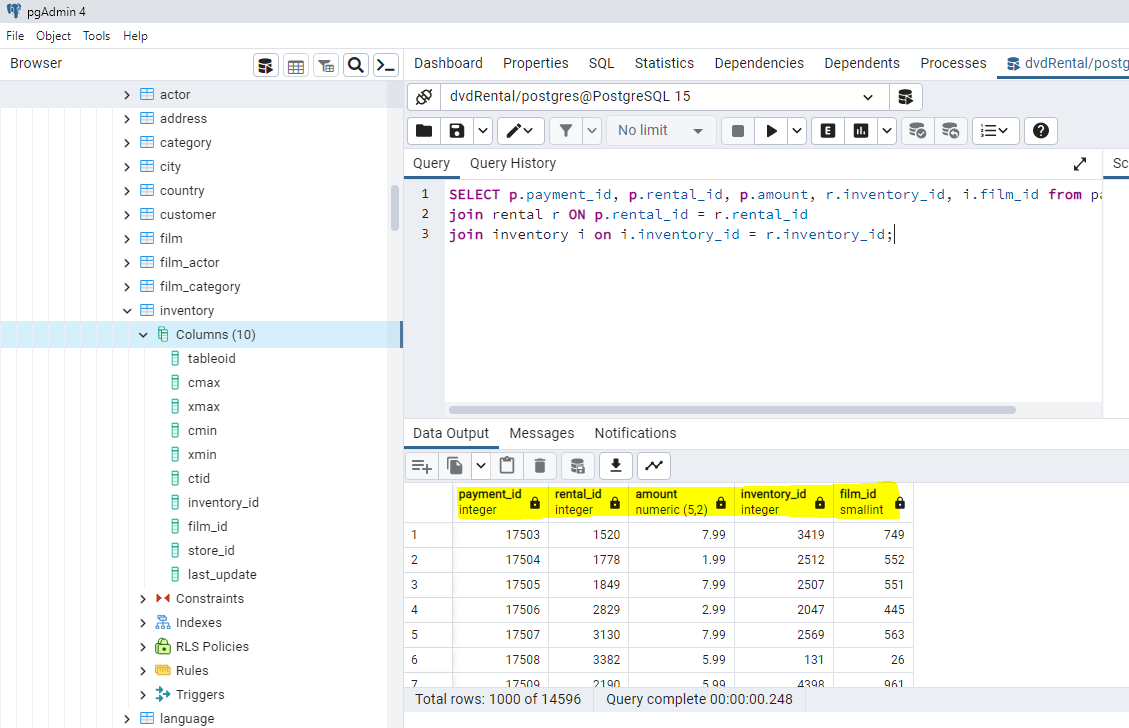
SELECT p.payment\_id, p.rental\_id, p.amount, r.inventory\_id, i.film\_id from payment as p

join rental r ON p.rental\_id = r.rental\_id

join inventory i on i.inventory\_id = r.inventory\_id;



* Run the command to get the below output:



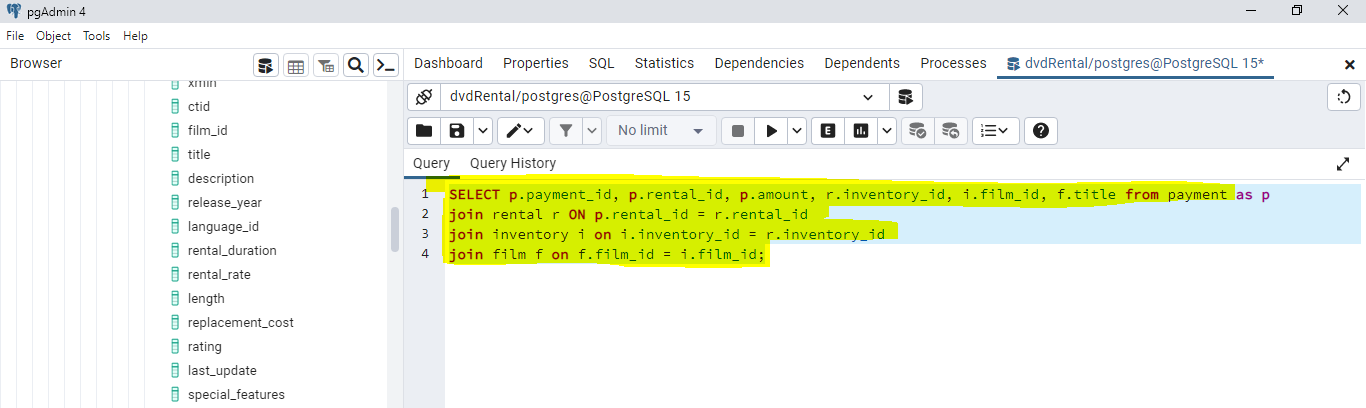
* Next join the inventory table and the film table on film\_id using the below code:

SELECT p.payment\_id, p.rental\_id, p.amount, r.inventory\_id, i.film\_id, f.title from payment as p

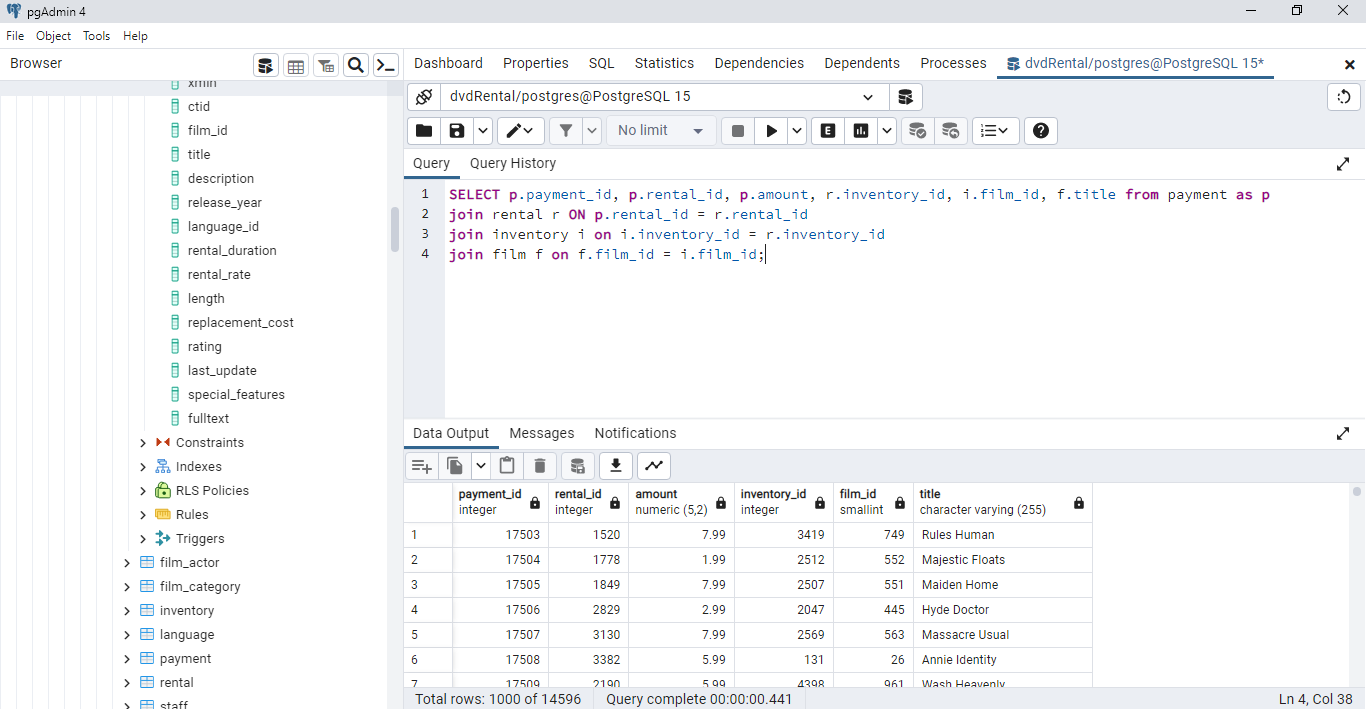
join rental r ON p.rental\_id = r.rental\_id

join inventory i on i.inventory\_id = r.inventory\_id

join film f on f.film\_id = i.film\_id;

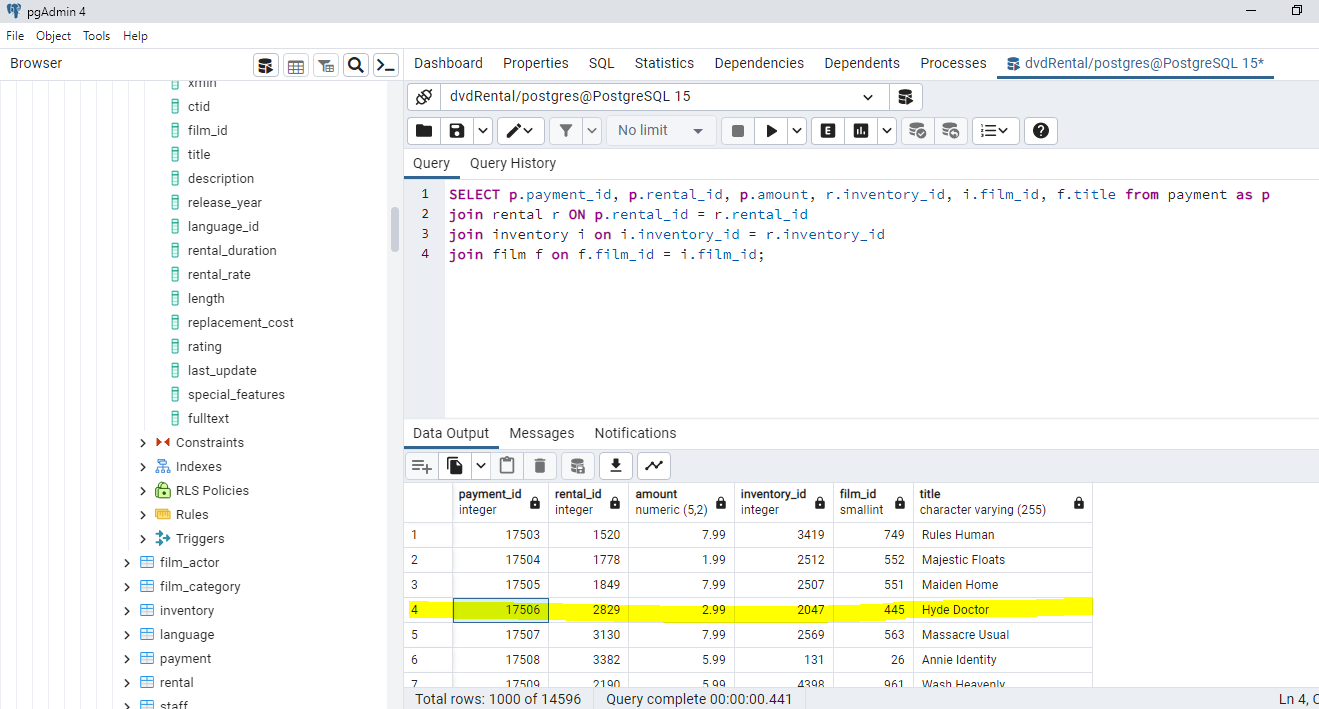


* Click on run command to get the below output.



* Finally, we have the payment information on all the films

Ex: the film with title -> Hyde doctor has made a payment of 17506.



* To know how much amount does the film make use the below code:

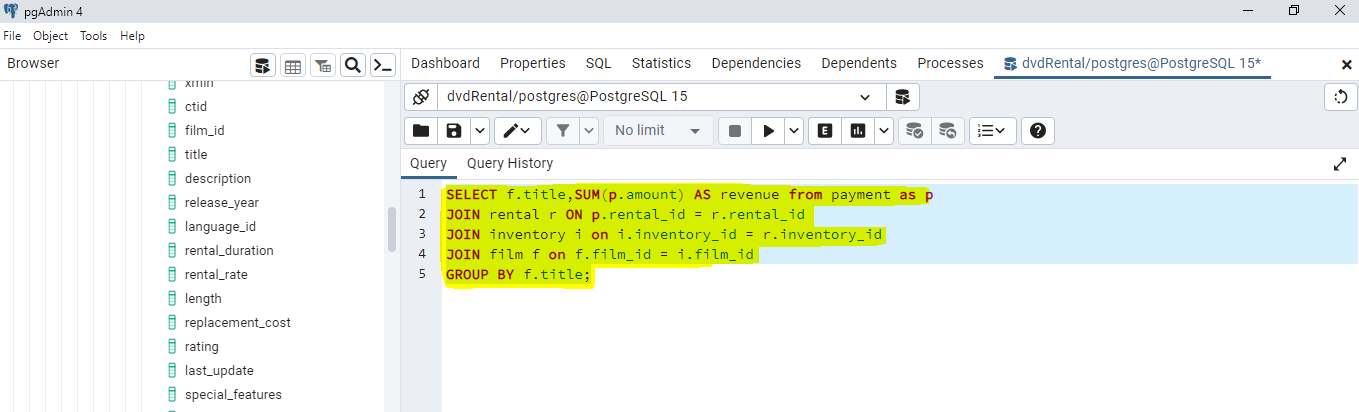
SELECT f.title,SUM(p.amount) AS revenue from payment as p

JOIN rental r ON p.rental\_id = r.rental\_id

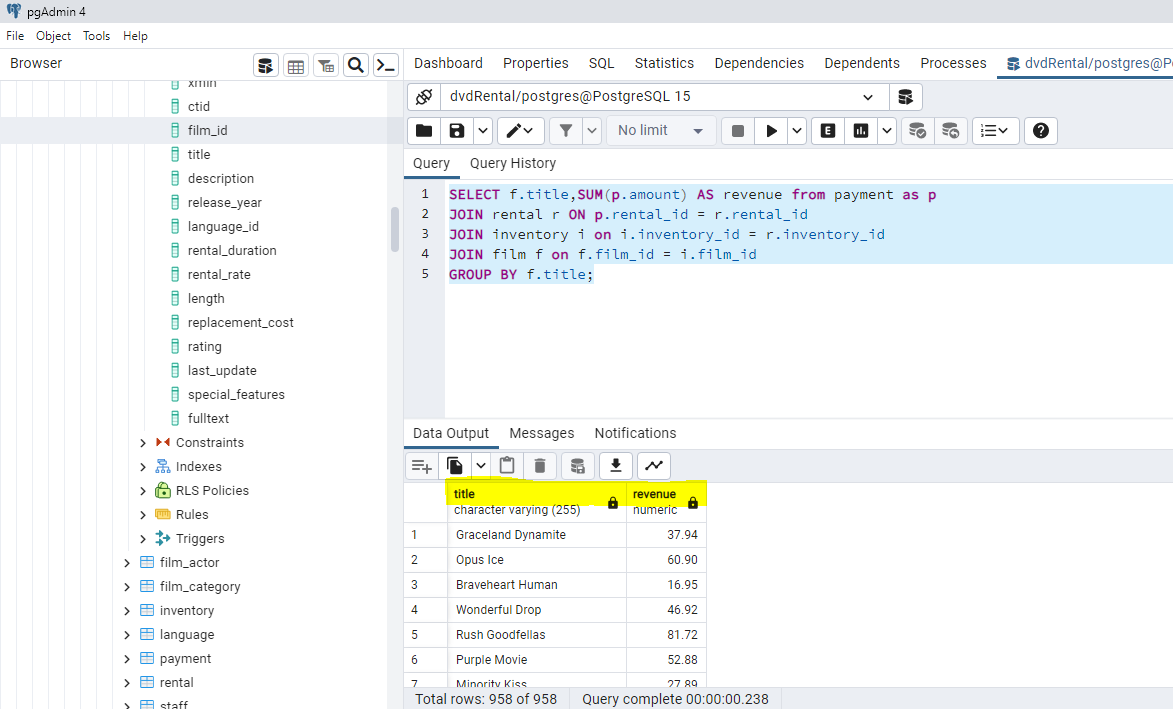
JOIN inventory i on i.inventory\_id = r.inventory\_id

JOIN film f on f.film\_id = i.film\_id

GROUP BY f.title;



* Run the command to get the below output.



* To get the maximum of the revenue use the below code:

SELECT f.title,SUM(p.amount) AS revenue from payment as p

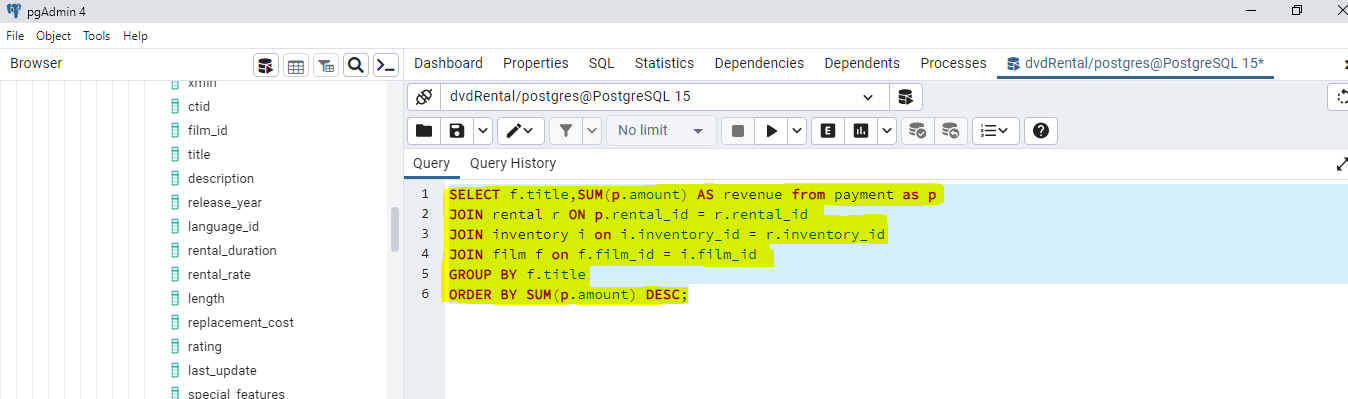
JOIN rental r ON p.rental\_id = r.rental\_id

JOIN inventory i on i.inventory\_id = r.inventory\_id

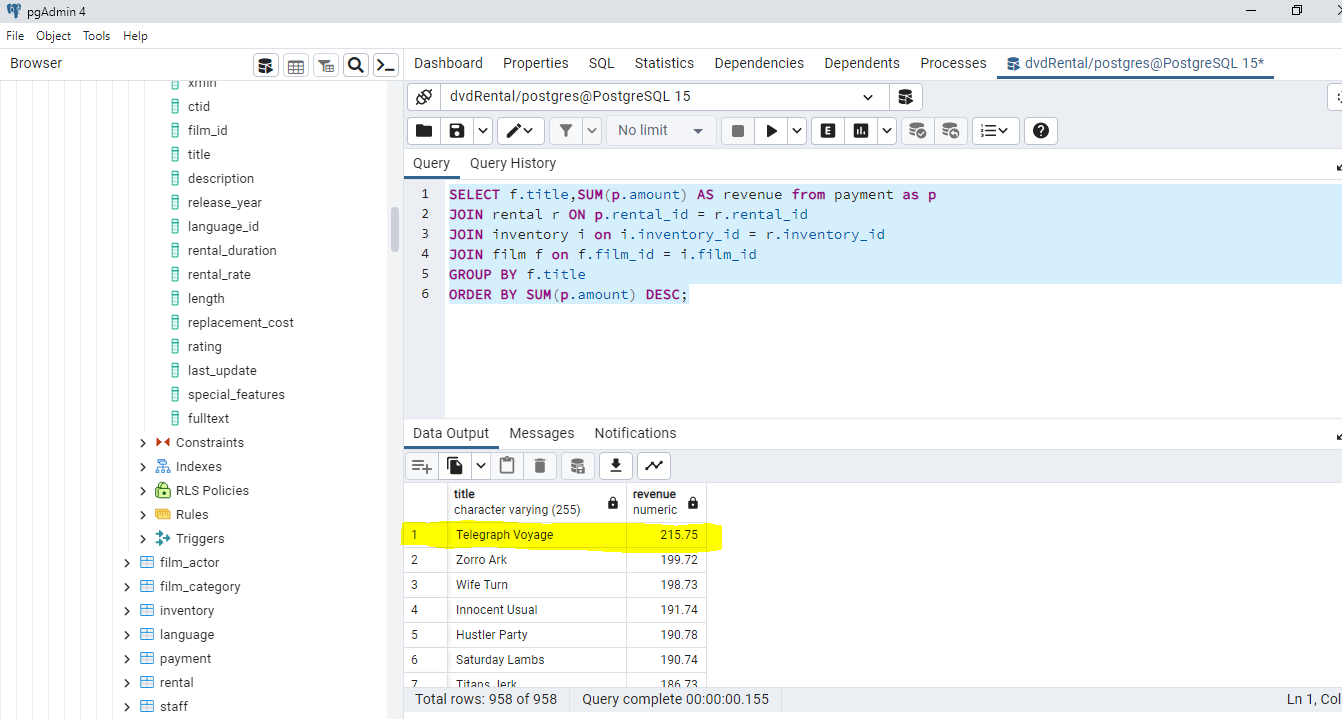
JOIN film f on f.film\_id = i.film\_id

GROUP BY f.title

ORDER BY SUM(p.amount) DESC;



* Run the command to get the below output.



* To get the minimum revenue use the below code.

SELECT f.title,SUM(p.amount) AS revenue from payment as p

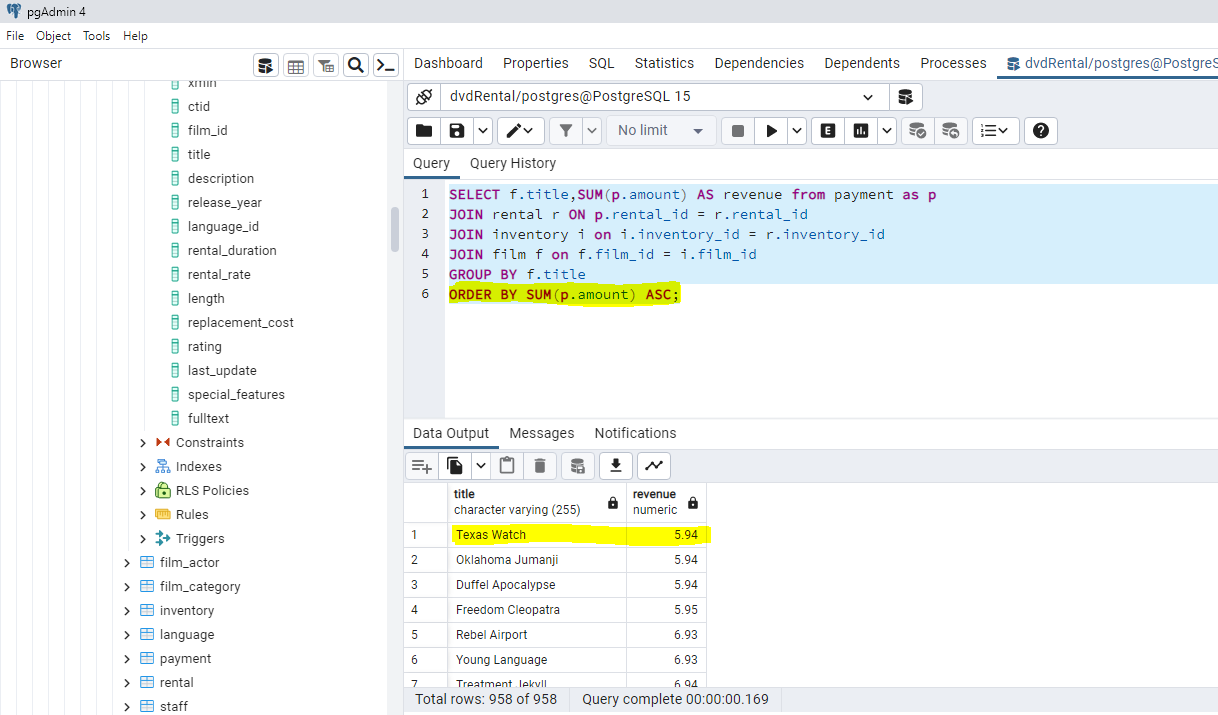
JOIN rental r ON p.rental\_id = r.rental\_id

JOIN inventory i on i.inventory\_id = r.inventory\_id

JOIN film f on f.film\_id = i.film\_id

GROUP BY f.title

ORDER BY SUM(p.amount) ASC;



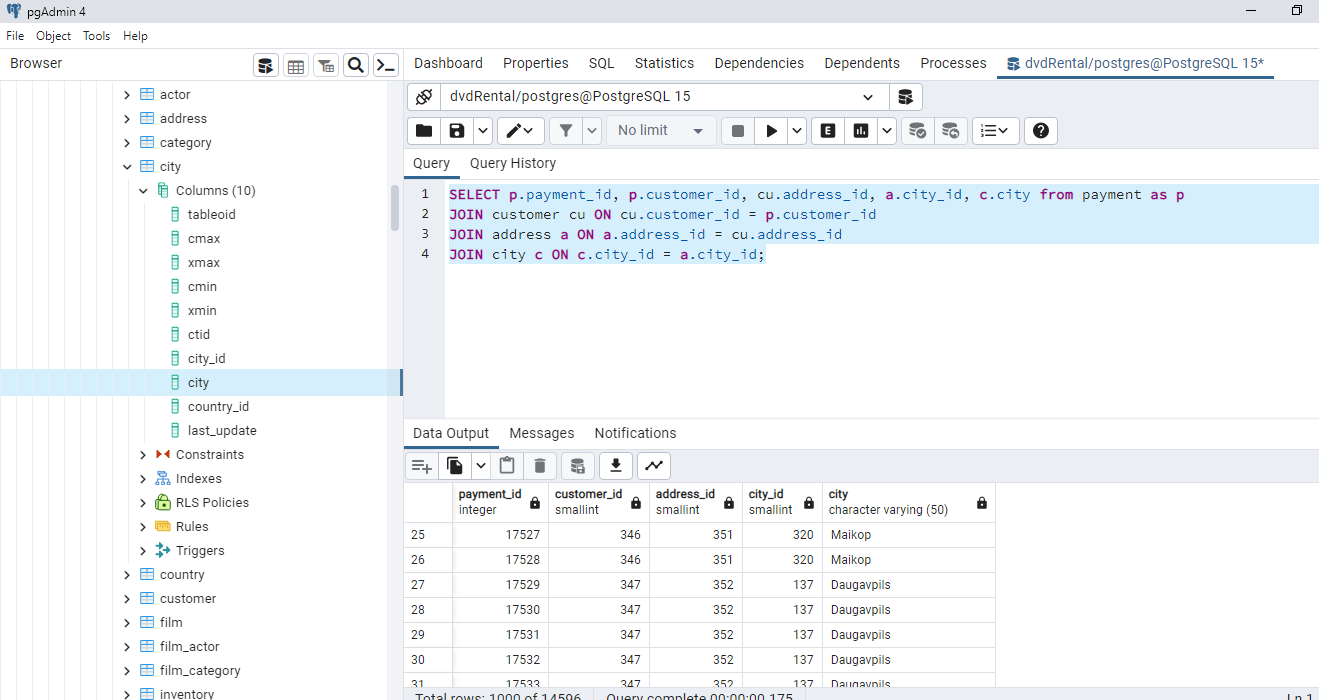
* To find highest revenue generated by city we need to join payment table and city table however there is no direct connection between both the tables so first we have to join payment table and customer table then need to join address and finally join city table to get the desired output.
* Use the below code to get the output.

SELECT p.payment\_id, p.customer\_id, cu.address\_id, a.city\_id, c.city from payment as p

JOIN customer cu ON cu.customer\_id = p.customer\_id

JOIN address a ON a.address\_id = cu.address\_id

JOIN city c ON c.city\_id = a.city\_id;



* Now to see the revenue base on city use the below code:

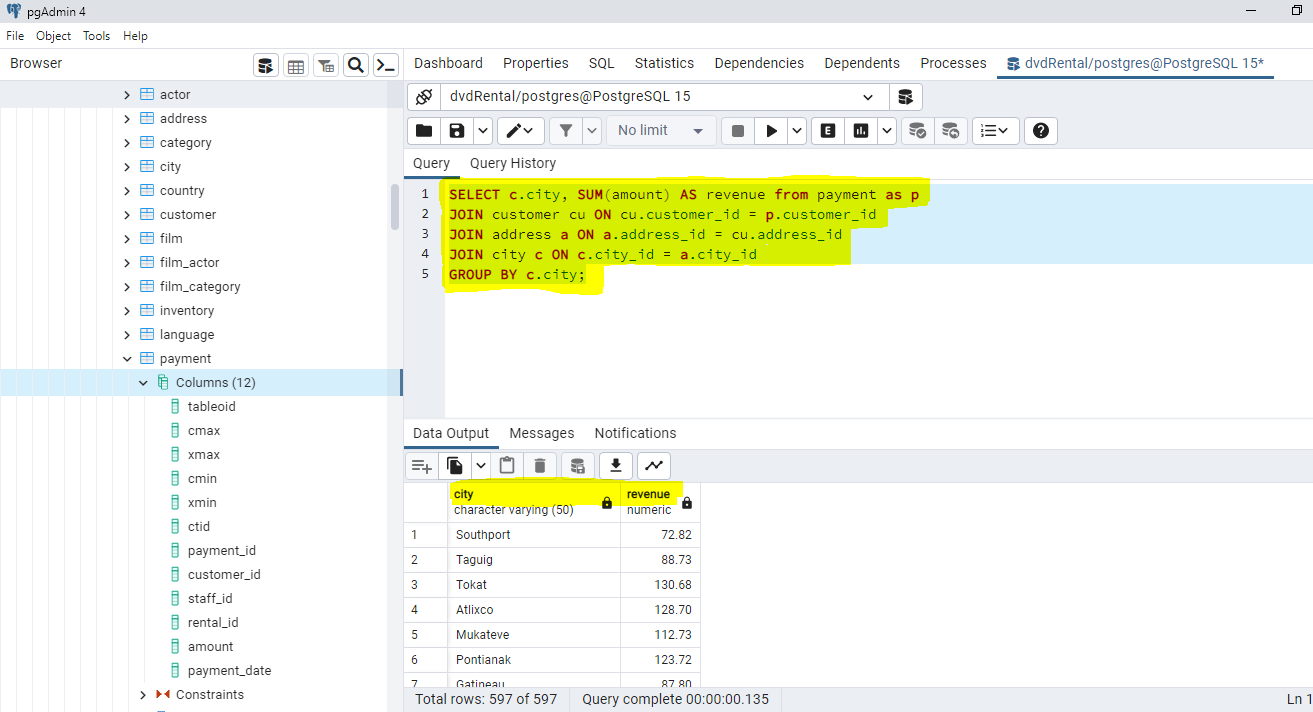
SELECT c.city, SUM(amount) AS revenue from payment as p

JOIN customer cu ON cu.customer\_id = p.customer\_id

JOIN address a ON a.address\_id = cu.address\_id

JOIN city c ON c.city\_id = a.city\_id

GROUP BY c.city;



* To get the highest revenue based on city use the below code.

SELECT c.city, SUM(amount) AS revenue from payment as p

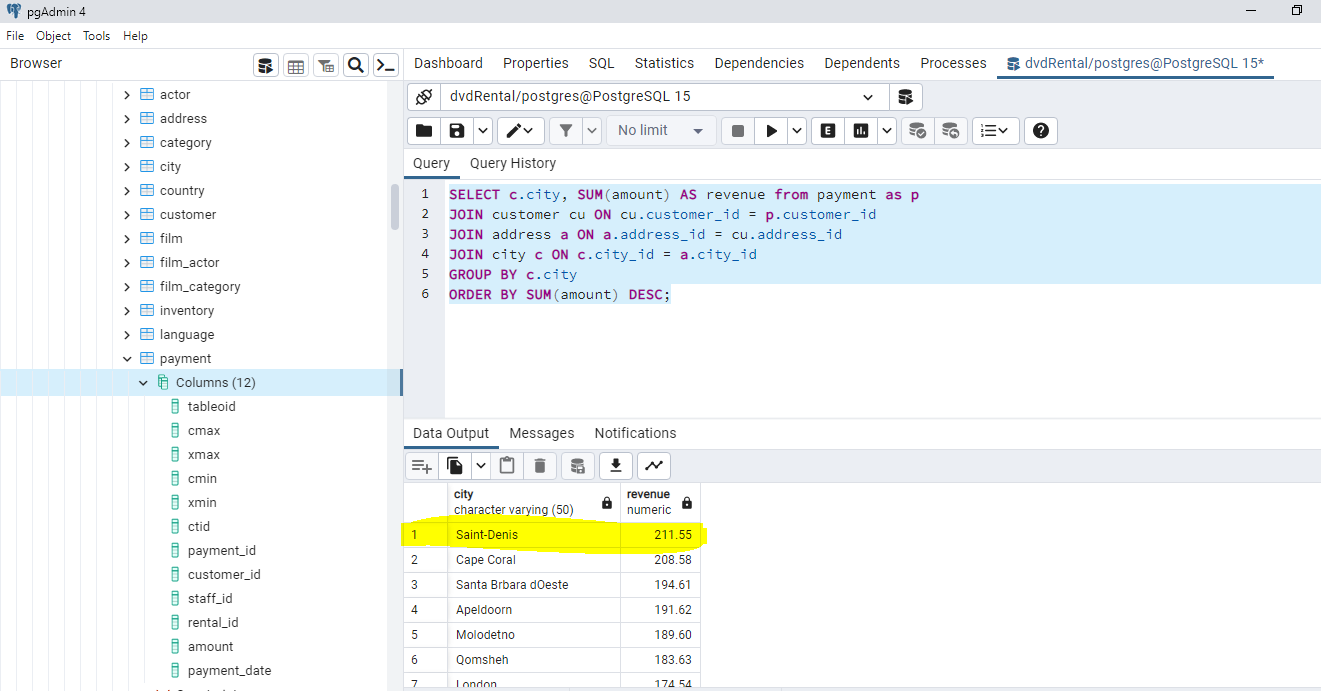
JOIN customer cu ON cu.customer\_id = p.customer\_id

JOIN address a ON a.address\_id = cu.address\_id

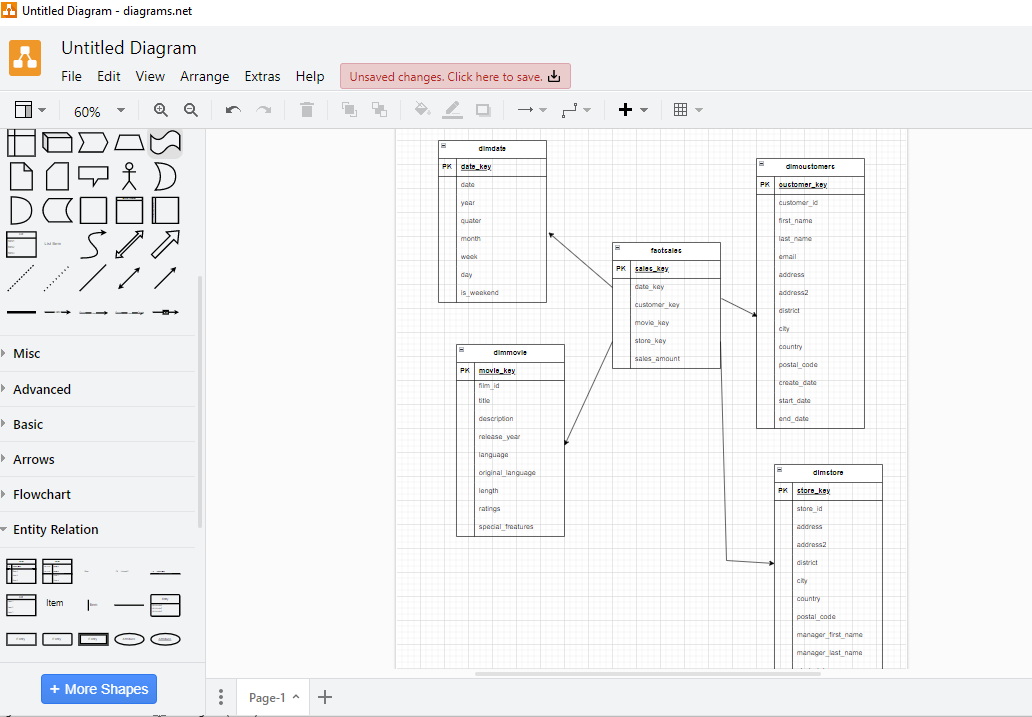
JOIN city c ON c.city\_id = a.city\_id

GROUP BY c.city

ORDER BY SUM(amount) DESC;



* If we see the 3NF schema we can say that data is redundant across different table and in order to avoid redundancy in the data we can convert the 3NF schema to star schema where repetitive data is stored in one dimension.
* The ER diagram for star schema is shown below:



* By using the star schema, we have combined tables containing the same information.
* To create the same table in postgresql go to pgAdmin -> click on Query tool.
* Use the below code to create the dimdate table.

CREATE TABLE dimDate

(

date\_key integer NOT NULL PRIMARY KEY,

date date NOT NULL,

year smallint NOT NULL,

quarter smallint NOT NULL,

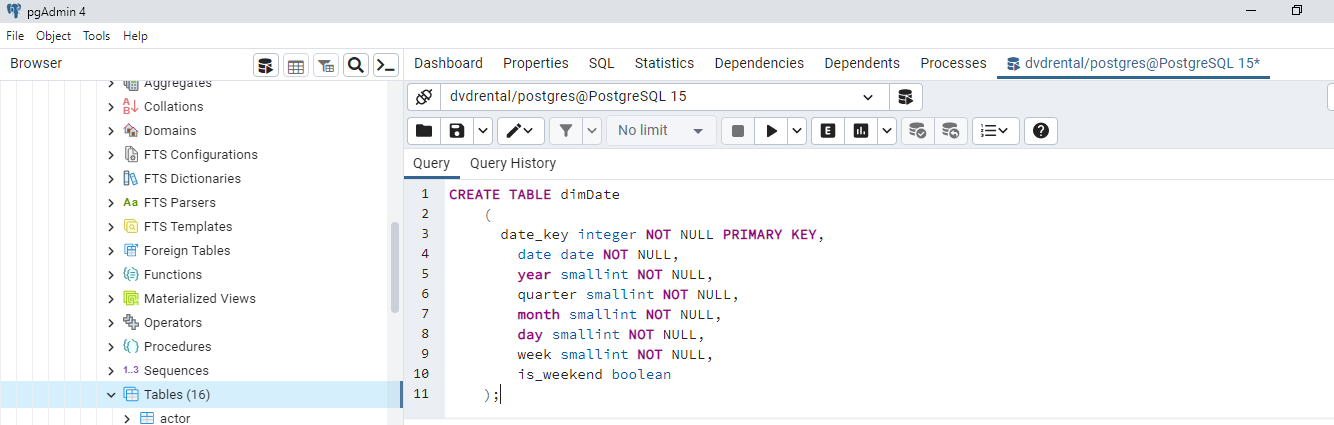
month smallint NOT NULL,

day smallint NOT NULL,

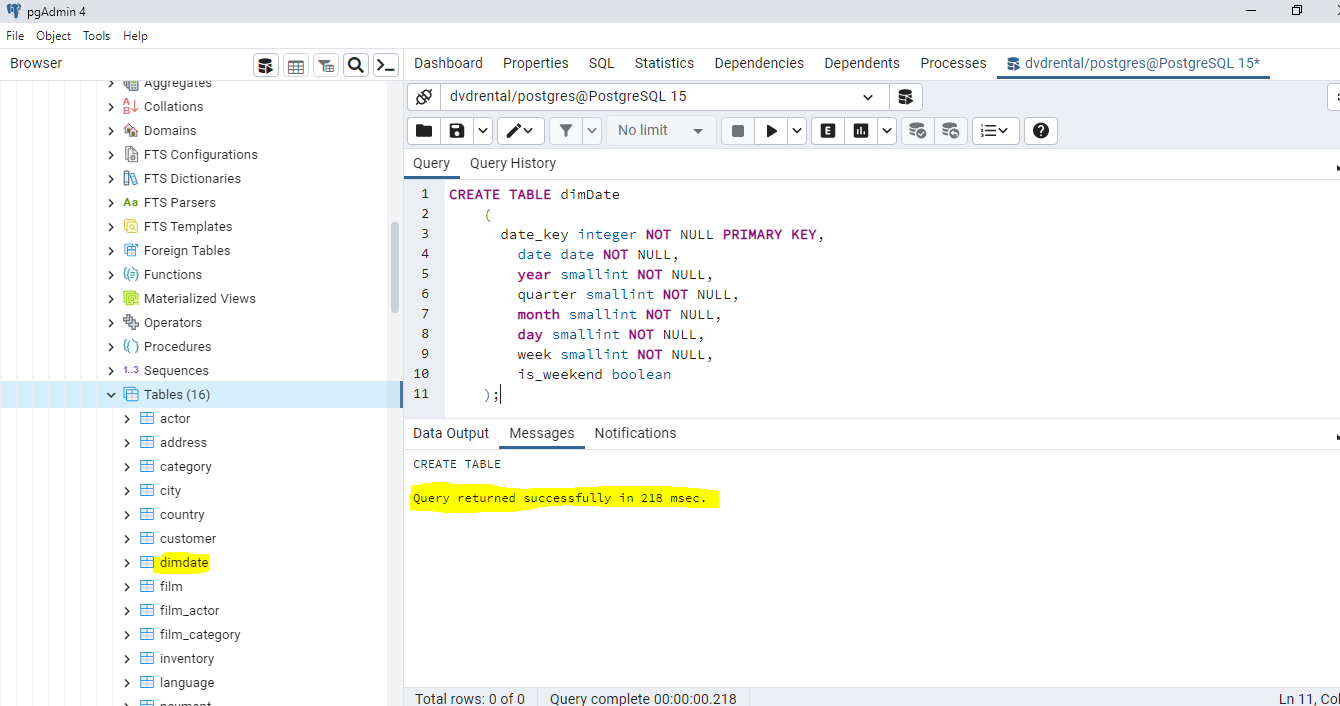
week smallint NOT NULL,

is\_weekend boolean

);



* Run the code to create the table.



* Insert the data into the “dimdate” table and to do so we have to extract the information from the payment table.
* Use the below commands to insert the data:

INSERT INTO dimDate (date\_key, date, year, quarter, month, day, week, is\_weekend)

SELECT DISTINCT(TO\_CHAR(payment\_date :: DATE, 'yyyyMMDD')::integer) AS date\_key,

date(payment\_date) AS date,

EXTRACT(year FROM payment\_date) AS year,

EXTRACT(quarter FROM payment\_date) AS quarter,

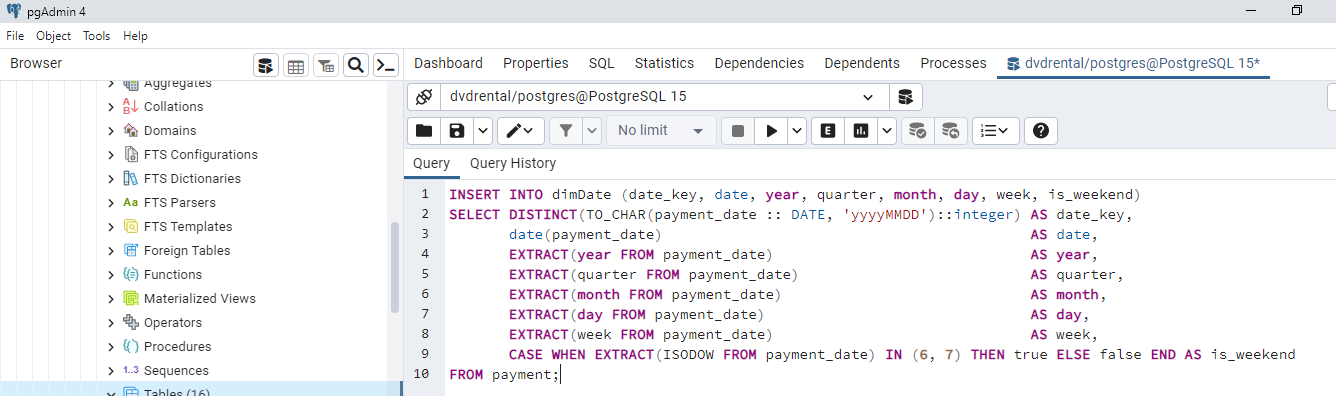
EXTRACT(month FROM payment\_date) AS month,

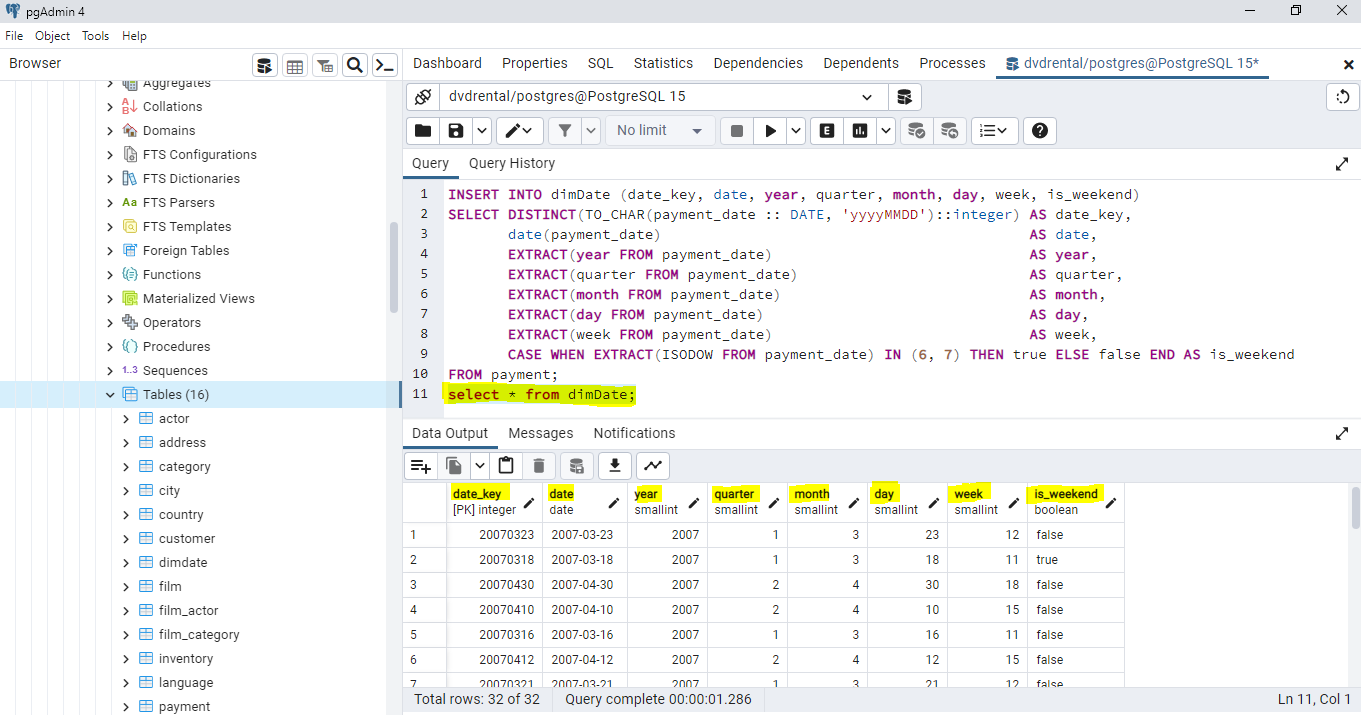
EXTRACT(day FROM payment\_date) AS day,

EXTRACT(week FROM payment\_date) AS week,

CASE WHEN EXTRACT(ISODOW FROM payment\_date) IN (6, 7) THEN true ELSE false END AS is\_weekend

FROM payment;





* Similarly use the below code to create “dimCustomer” table:

DROP TABLE IF EXISTS dimCustomer;

CREATE TABLE dimCustomer

(

customer\_key SERIAL PRIMARY KEY,

customer\_id smallint NOT NULL,

first\_name varchar(45) NOT NULL,

last\_name varchar(45) NOT NULL,

email varchar(50),

address varchar(50) NOT NULL,

address2 varchar(50),

district varchar(20) NOT NULL,

city varchar(50) NOT NULL,

country varchar(50) NOT NULL,

postal\_code varchar(10),

phone varchar(20) NOT NULL,

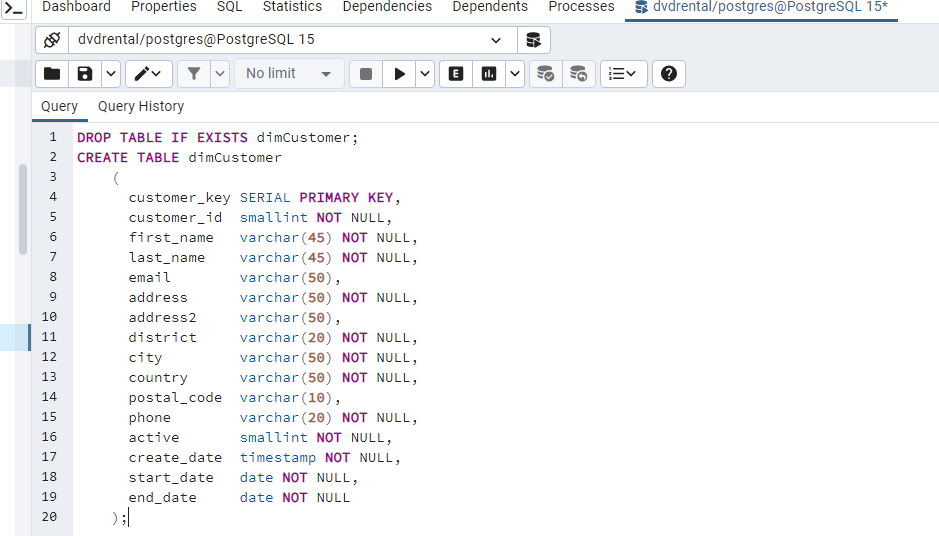
active smallint NOT NULL,

create\_date timestamp NOT NULL,

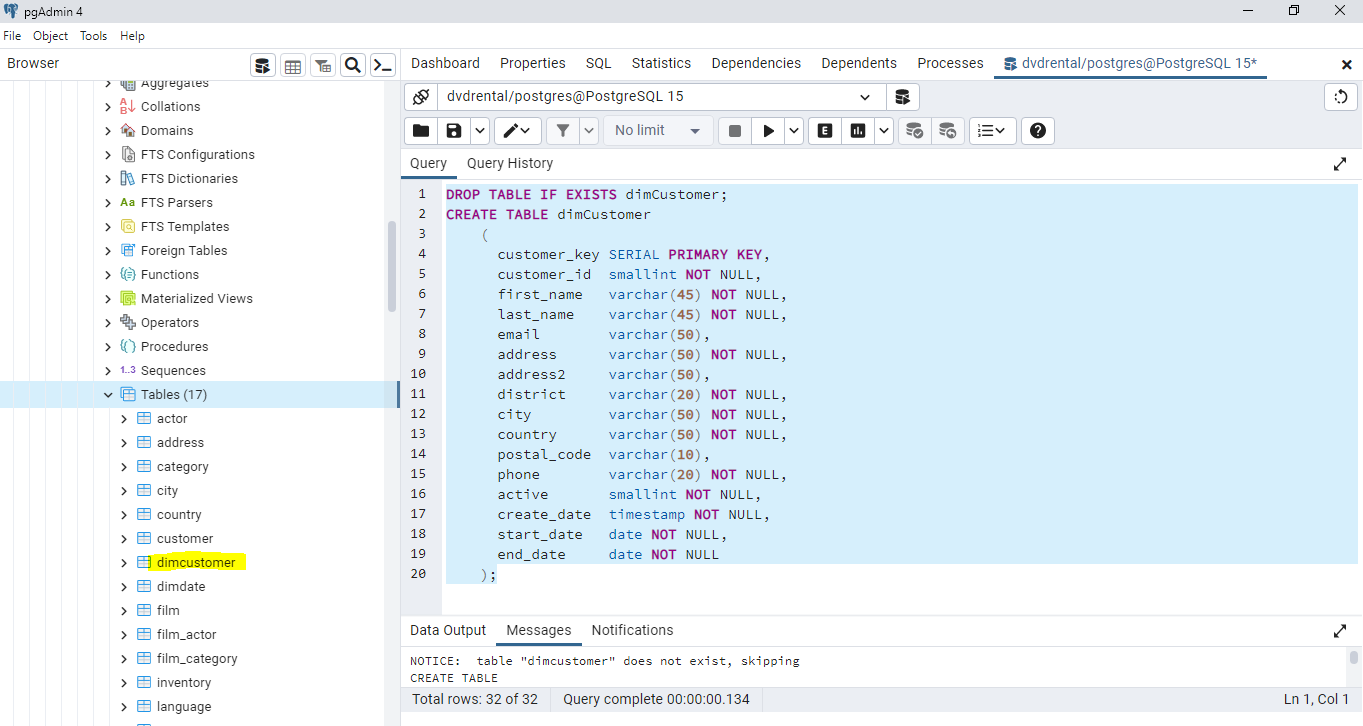
start\_date date NOT NULL,

end\_date date NOT NULL

);



* Run the command to get the table.



* Insert the data into the “dimCustomer” table using the below code:

INSERT INTO dimCustomer (customer\_key, customer\_id, first\_name, last\_name, email, address,

address2, district, city, country, postal\_code, phone, active,

create\_date, start\_date, end\_date)

SELECT c.customer\_id as customer\_key,

c.customer\_id,

c.first\_name,

c.last\_name,

c.email,

a.address,

a.address2,

a.district,

ci.city,

co.country,

postal\_code,

a.phone,

c.active,

c.create\_date,

now() AS start\_date,

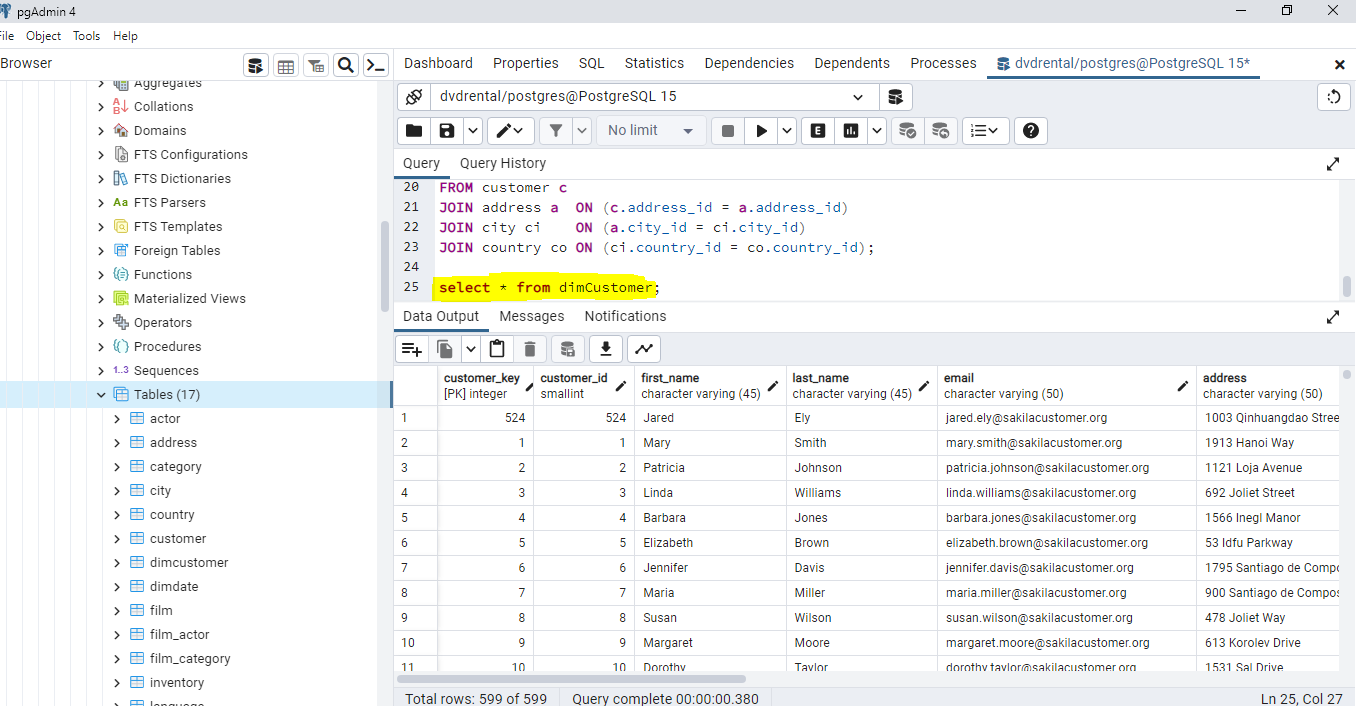
now() AS end\_date

FROM customer c

JOIN address a ON (c.address\_id = a.address\_id)

JOIN city ci ON (a.city\_id = ci.city\_id)

JOIN country co ON (ci.country\_id = co.country\_id);



* Simillarly use the below code to create “dimStore” table:

CREATE TABLE dimStore

(

store\_key SERIAL PRIMARY KEY,

store\_id smallint NOT NULL,

address varchar(50) NOT NULL,

address2 varchar(50),

district varchar(20) NOT NULL,

city varchar(50) NOT NULL,

country varchar(50) NOT NULL,

postal\_code varchar(10),

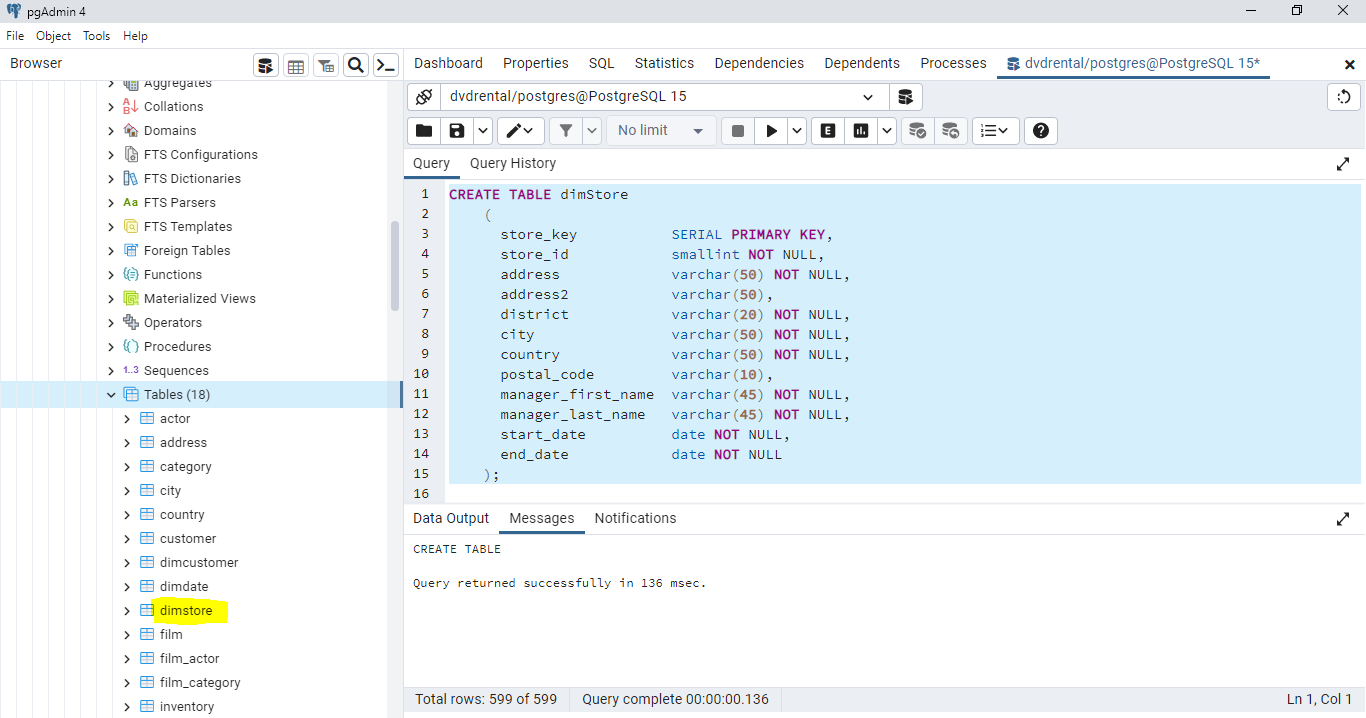
manager\_first\_name varchar(45) NOT NULL,

manager\_last\_name varchar(45) NOT NULL,

start\_date date NOT NULL,

end\_date date NOT NULL

);



* Use the below code to insert the data:

INSERT INTO dimStore (store\_key, store\_id, address, address2, district, city, country, postal\_code, manager\_first\_name, manager\_last\_name, start\_date, end\_date)

SELECT

s.store\_id as store\_key,

s.store\_id,

a.address,

a.address2,

a.district,

c.city,

co.country,

a.postal\_code,

st.first\_name as manager\_first\_name,

st.last\_name as manager\_last\_name,

now() as start\_date,

now() as end\_date

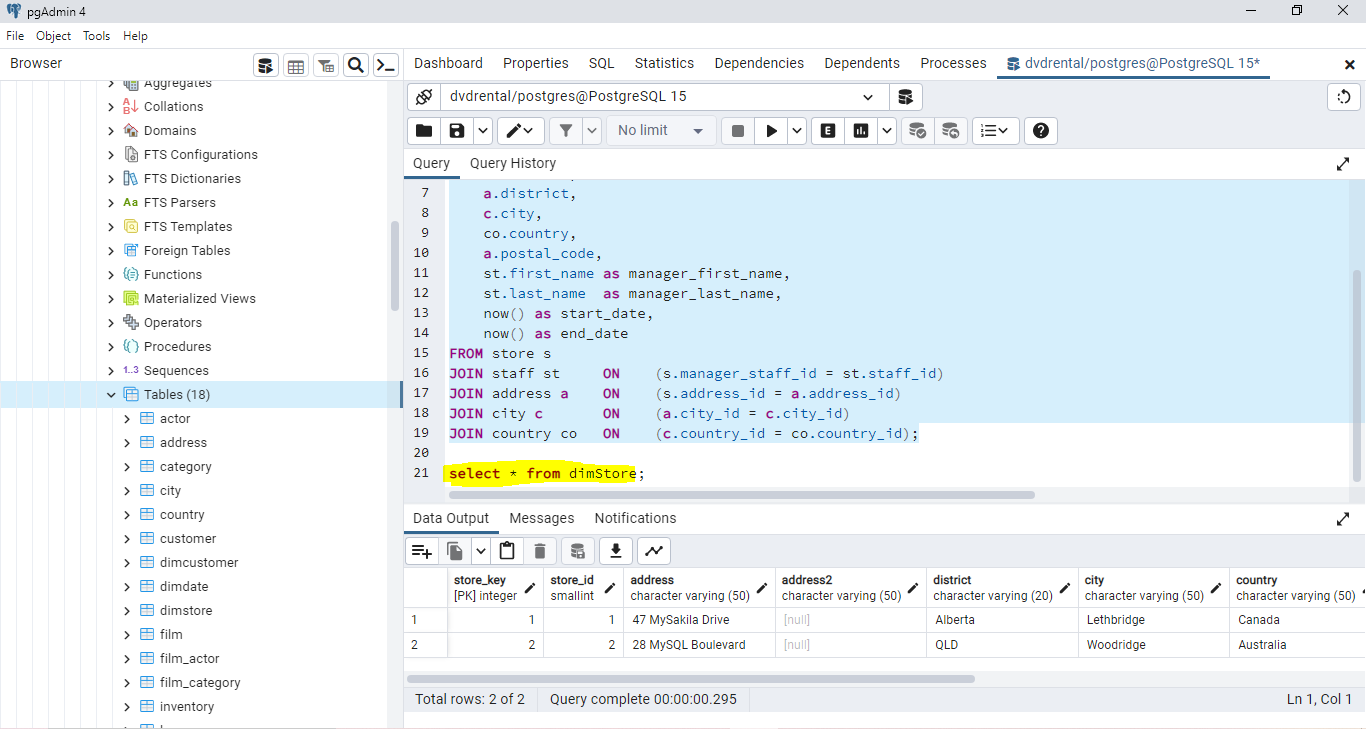
FROM store s

JOIN staff st ON (s.manager\_staff\_id = st.staff\_id)

JOIN address a ON (s.address\_id = a.address\_id)

JOIN city c ON (a.city\_id = c.city\_id)

JOIN country co ON (c.country\_id = co.country\_id);



* Use the below code to create “dimMovie” table:

CREATE TABLE dimMovie

(

movie\_key SERIAL PRIMARY KEY,

film\_id smallint NOT NULL,

title varchar(255) NOT NULL,

description text,

release\_year year,

language varchar(20) NOT NULL,

original\_language varchar(20),

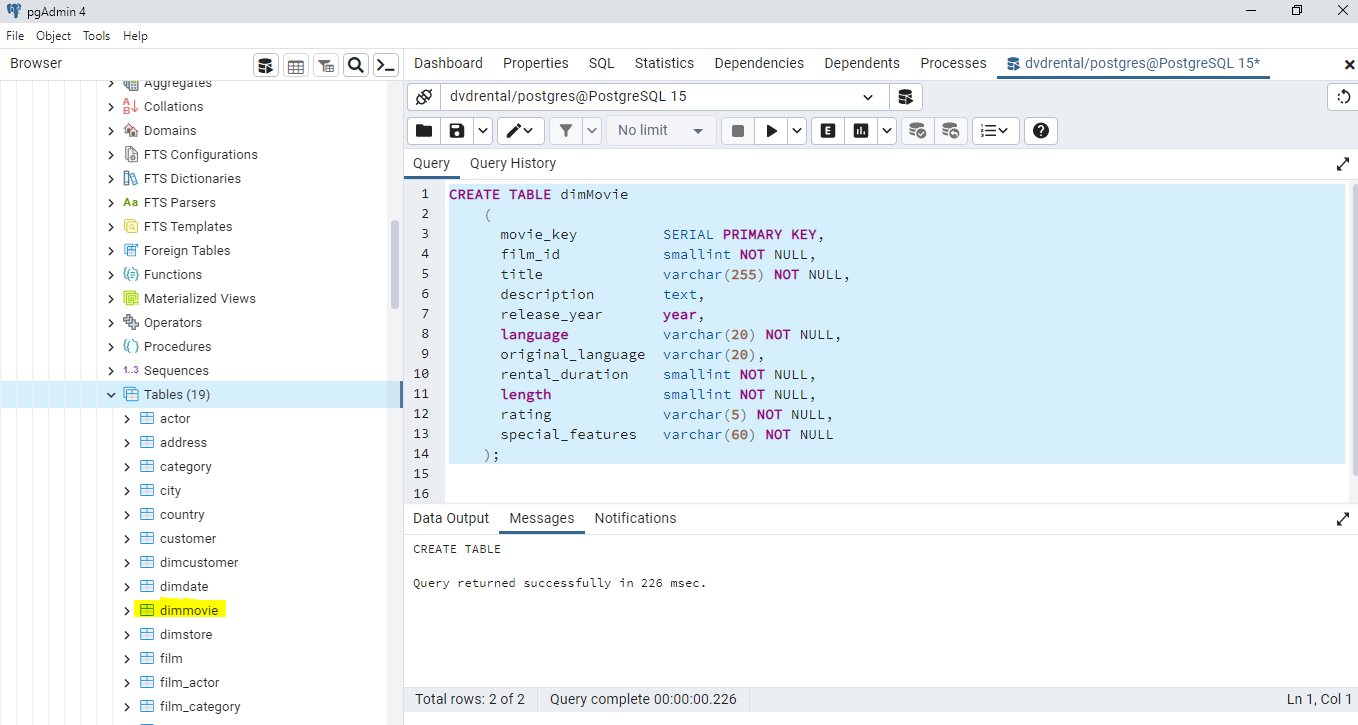
rental\_duration smallint NOT NULL,

length smallint NOT NULL,

rating varchar(5) NOT NULL,

special\_features varchar(60) NOT NULL

);



* Use the below code to insert the data:

INSERT INTO dimMovie (movie\_key, film\_id, title, description, release\_year, language, original\_language, rental\_duration, length, rating, special\_features)

SELECT

f.film\_id as movie\_key,

f.film\_id,

f.title,

f.description,

f.release\_year,

l.name as language,

orig\_lang.name AS original\_language,

f.rental\_duration,

f.length,

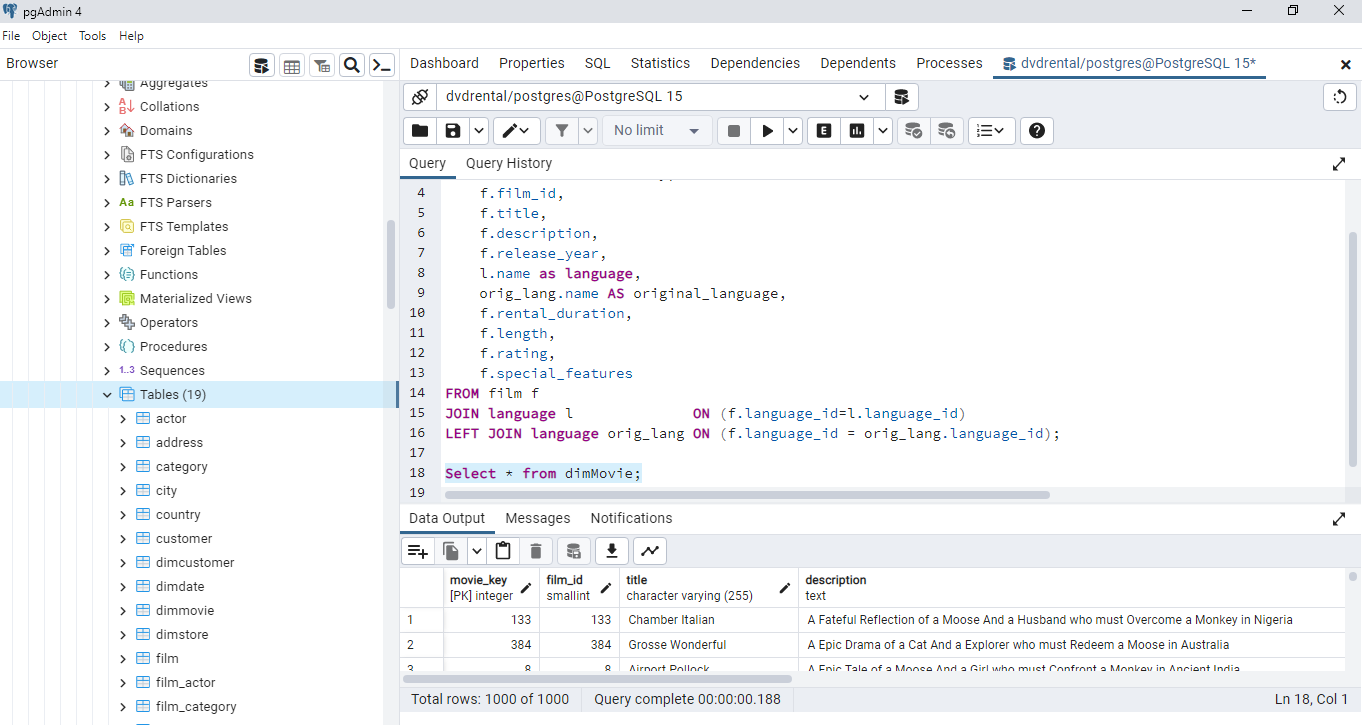
f.rating,

f.special\_features

FROM film f

JOIN language l ON (f.language\_id=l.language\_id)

LEFT JOIN language orig\_lang ON (f.language\_id = orig\_lang.language\_id);



* Finally use the below code to create fact table:

DROP TABLE IF EXISTS factSales;

CREATE TABLE factSales

(

sales\_key SERIAL PRIMARY KEY,

date\_key integer REFERENCES dimDate (date\_key),

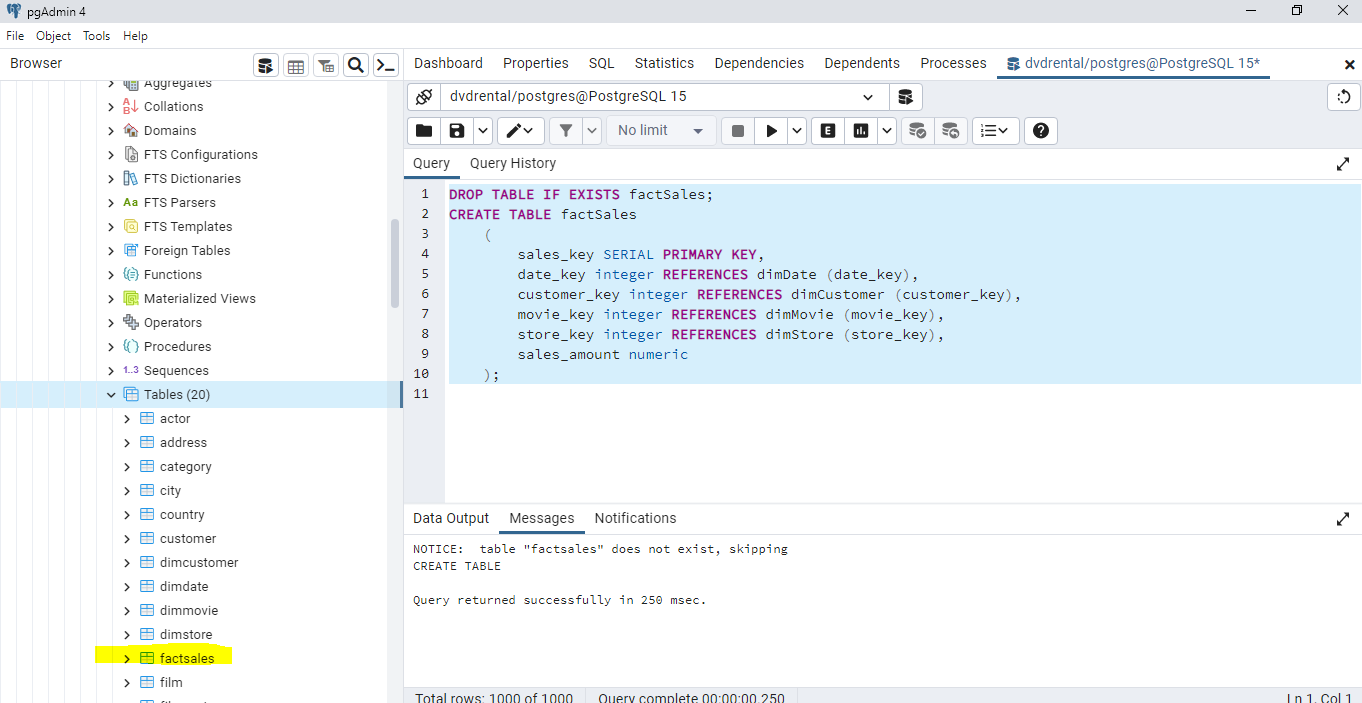
customer\_key integer REFERENCES dimCustomer (customer\_key),

movie\_key integer REFERENCES dimMovie (movie\_key),

store\_key integer REFERENCES dimStore (store\_key),

sales\_amount numeric

);



* Use the below code to insert the data in fact table:

INSERT INTO factSales (date\_key, customer\_key, movie\_key, store\_key, sales\_amount)

SELECT

TO\_CHAR(payment\_date :: DATE, 'yyyyMMDD')::integer AS date\_key,

p.customer\_id as customer\_key,

i.film\_id as movie\_key,

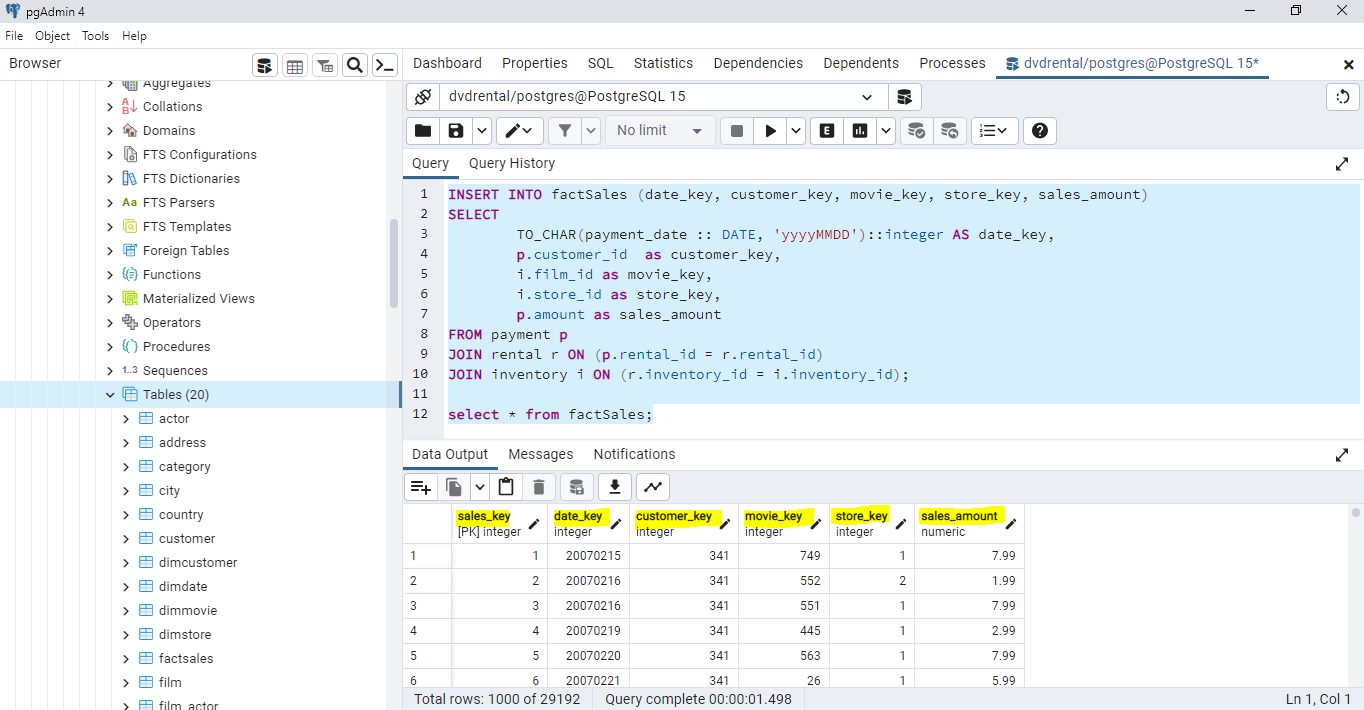
i.store\_id as store\_key,

p.amount as sales\_amount

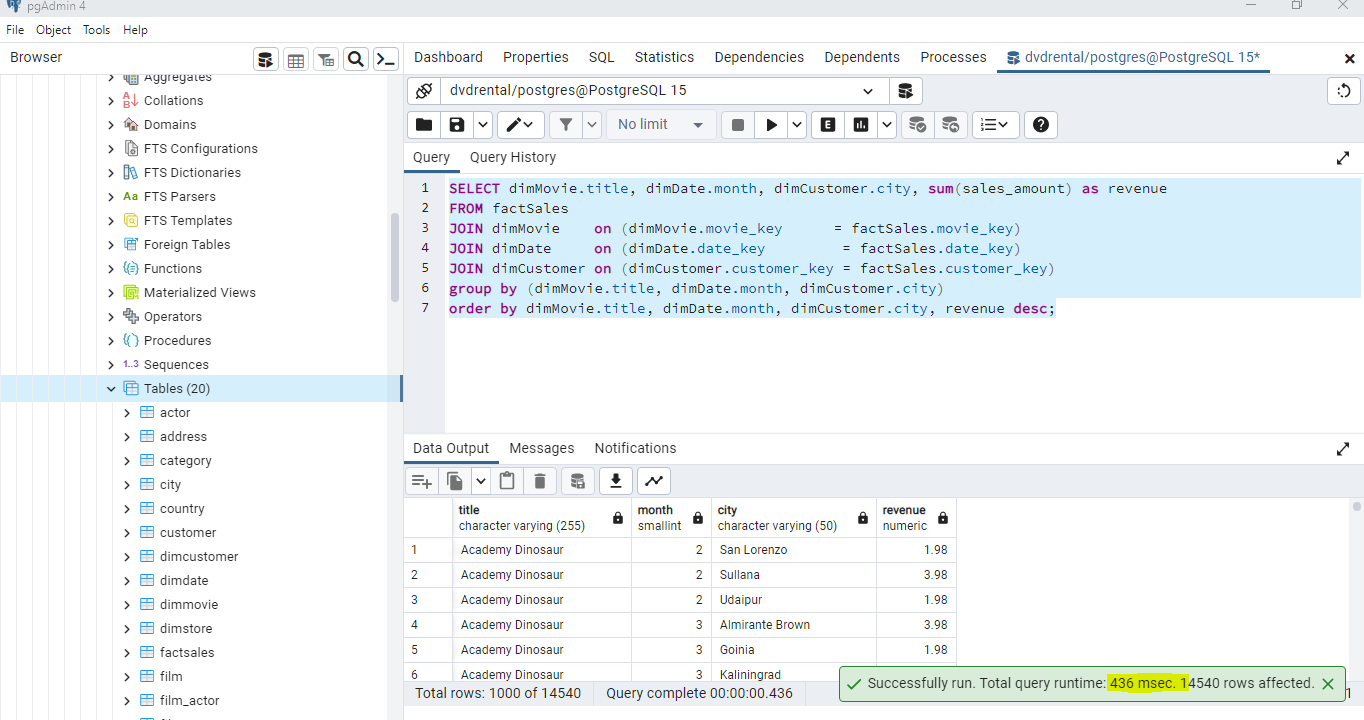
FROM payment p

JOIN rental r ON (p.rental\_id = r.rental\_id)

JOIN inventory i ON (r.inventory\_id = i.inventory\_id);



* To do analysis on the star schema it takes less time compared to 3nf schema as to achieve the desired analysis we have to join multiple table in 3nf schema which makes it complex where as in star schema it can be achieved with less joins which ultimately results in less complexity and less execution time.
* Star schema - It took 436 milliseconds.



* 3nf Schema – It took 484 milliseconds:

