**Portfolio Project**

**Data Loading – Data Warehouse**

Joshua Humphreys

Colorado State University Global

Course Code:MIS541

Dr. Osama Morad

October 8, 2023

**Portfolio Project**

I selected the pagila database for my portfolio project. In my first milestone paper, I identified the business process, grain, dimensions, and fact table for my data warehouse star schema (Humphreys, 2023a). Based on the instructor feedback I received from my first milestone and a review of my previous work, I made minor changes to the original attributes, but did not make significant changes to the business process, grain, dimensions, or fact tables in my second milestone. In my second milestone I created the fact and dimension tables for my star schema (Humphreys, 2023b). For this step of my project, I did not make any changes to the business process, grain, dimensions, fact tables, or attributes.

In the following pages I will provide a description of the process and SQL commands that I used to extract data from the OLTP source database. I will then provide a description of the process and SQL commands that I used to insert the data into my destination database. Finally, I will provide a brief description of my lessons learned in this project. I have included screenshots of each table’s row count and the first ten rows of each table (Appendix A). I have also included a complete listing of all SQL commands for reference (Appendix B).

**SQL Commands**

The first step in the ETL process was to extract the data from the source database. I used the *COPY* command table joins to do this. The code used to extract data for the film dimension table includes all the commands I used for all the data extraction procedures. Those procedures are as follows:

--copy data except for original language for the film dimension to csv

COPY (SELECT film\_id, title, release\_year, language.name, length, replacement\_cost, rating, description, special\_features

FROM film INNER JOIN language USING (language\_id) )

TO '/users/public/dim\_film\_no\_orig\_lang.csv' DELIMITER ',' CSV HEADER;

--copy original language data for film dimension to csv

COPY (SELECT film\_id, language.name

FROM film INNER JOIN language ON (film.original\_language\_id=language.language\_id) )

TO '/users/public/dim\_film\_orig\_lang.csv'

DELIMITER ','

CSV HEADER;

I used the *COPY* command to copy a given selection to an external file. In this case I copied all the data to CSV files. The *DELIMITER* command defines what type of delimiter to use in the CSV file. The *CSV HEADER* command creates a header row in the CSV file.

I used the *SELECT* command to define the individual columns to be copied in the *COPY* command. The *FROM* command defines where those columns come from. Since the source database was a normalized transactional database, in some cases, the data for a dimension in the target database came from multiple tables in the source database. For efficiency’s sake, I used table joins to extract data from multiple tables at a time.

As you can see, I had to extract the data for the film dimension in two different steps. This is because there are two different languages associated with a film dimension. That is, the *film* table refers to the *language* table in two different columns. As previously mentioned, I used table joins via the *INNER JOIN* function in each step. The *INNER JOIN* function will allow for nested joins (as can be seen in the previous code). However, you cannot join two different tables more than once. Mathematically speaking, the *INNER JOIN* function produces the intersection between two sets where the intersection is defined by the condition given in the command line. In some cases, I used the *ON* command to define the join condition. The *ON* command combines rows where the given logical test returns a *true* value. In some cases, I used the *USING* command. The *USING* command identifies the given column name in both joined tables and selects rows where the values in the given columns are equal.

The load procedures used to populate the *film* and *rental\_fact* tables required a combination of all the procedures used to populate all the tables in the target databases. The procedures for the *film* table are as follows:

--copy data except original language from csv to film dimension table

COPY film(film\_id, title, release\_year, language\_name, length, replacement\_cost, rating, description, special\_features)

FROM '/users/public/dim\_film\_no\_orig\_lang.csv' DELIMITER ',' CSV HEADER;

--create temporary table for original language data

CREATE TEMP TABLE temp\_table1(

film\_id integer,

original\_language\_name character(20)

);

--copy original language data from csv to temporary table

COPY temp\_table1

FROM '/users/public/dim\_film\_orig\_lang.csv' DELIMITER ',' CSV HEADER;

--insert original language data from temporary table in to film dimension

UPDATE film

SET original\_language=(SELECT original\_language\_name FROM temp\_table1 where film.film\_id=temp\_table1.film\_id);

--drop temporary table

DROP TABLE temp\_table1;

I used the *COPY* function once again. In the first instance, I used the *COPY* function to copy the data from the given source directory given in the *FROM* command to the table and columns given in the *COPY* command. *DELIMITER* in this case, defines what delimiter to look for in the data. *CSV HEADER* ensures the first row (header row) in the source file is ignored.

In the next step, I created a temporary table using the *CREATE TEMP TABLE* commands. The table attributes are defined to accept the data from the source file containing the original language data. I used the *COPY* command once again to populate the temporary table. Then I used the *UPDATE* command to update the already populated *film* table. The *SET* command defines what changes are made to the *film* table. In this case, the *original\_language* attribute is set to be equal to the attribute values defined by the *SELECT* command.

I used the *SELECT* and *FROM* commands in the same way as before except for the addition of the *WHERE* command. The *WHERE* command defines the condition that must be true in the defined rows for the specified row values to be copied. In plain English, this set of commands identifies rows in both tables where their respective *film\_id* values are equal. When this condition is met, the *original\_language* attribute in the *film* table is updated with the *original\_language­* value from the temporary table.

I used a similar procedure to populate the *rental\_fact* table. The only difference was that instead of the data needed for the *rental\_fact* table being in two different source files, the data was in a source file and the date dimension. That is, the dates themselves were in the source file. However, those dates needed to be converted to key values that referenced rows in the date dimension. The load procedures for the *rental\_fact* table are as follows:

--create temporary table for rental fact table

--temporary table includes columns for date and date keys

--date values of date type were extracted from source database

CREATE TEMP TABLE temp\_fact\_table(

date\_key integer,

rental\_date date,

film\_key integer,

actor\_key integer,

customer\_key integer,

staff\_key integer,

category\_Key integer,

store\_key integer,

rental\_return\_date\_key integer,

rental\_return\_date date,

payment\_date\_key integer,

payment\_date date,

rental\_id integer,

payment\_amount numeric(5,2)

);

--copy data from csv to rental fact table

COPY temp\_fact\_table(rental\_date, film\_key, actor\_key, customer\_key, staff\_key, category\_key,

store\_key, rental\_return\_date, payment\_date, rental\_id, payment\_amount)

FROM '/users/public/rental\_fact.csv'

DELIMITER ','

CSV HEADER;

--copy date key from date dimension for corresponding rental dates to temporary table

UPDATE temp\_fact\_table

SET date\_key=(select date\_idfrom date where temp\_fact\_table.rental\_date=date.date\_name);

--copy date key from date dimension for corresponding rental return date to temporary table

UPDATE temp\_fact\_table

SET rental\_return\_date\_key=(select date\_id from date where temp\_fact\_table.rental\_return\_date=date.date\_name);

--alter rental\_fact\_id to automatically populate as surrogate key

ALTER table rental\_fact

ALTER column rental\_fact\_id ADD GENERATED ALWAYS AS identity;

--delete rows with missing foregn key data due to not null constraint

DELETE FROM temp\_fact\_table

WHERE date\_key IS NULL OR film\_key IS NULL OR actor\_key IS NULL OR

customer\_key IS NULL OR staff\_ IS NULL OR category\_key IS NULL OR

store\_key IS NULL OR rental\_return\_date\_key IS NULL OR payment\_date\_key IS NULL;

--insert data except dates from temporary table into rental fact table

INSERT INTO rental\_fact(date\_key, film\_key, actor\_key, customer\_key, staff\_key, category\_key,

store\_key, rental\_return\_date, payment\_date, rental\_id, payment\_amount)

SELECT date\_key, film\_key, actor\_key, customer\_key, staff\_key, category\_key,

store\_key, rental\_return\_date\_key, payment\_date\_key, rental\_id, payment\_amount FROM temp\_fact\_table;

I used many of the same commands used to load the *film* table in the load procedure for the *rental\_fact* table. I created a temporary table with the same attributes as the *rental\_fact* table except that I added attributes for date keys. I used the same procedures to populate the temporary table. This time, I used the *UPDATE* command to update the temporary table with the *date\_id* values that correspond to the dates in the temporary table.

I used the *ALTER TABLE* and *ALTER COLUMN* commands to make the *rental\_fact\_id* primary key attribute a surrogate key that is populated whenever a row is added to the table. To do so, I used the *ADD GENERATED ALWAYS AS IDENTITY* commands. The *ADD* command adds a condition to the attribute. The *GENERATED ALWAYS* command means that any time a row is added, the attribute is generated. The *AS IDENTITY* commands means the attribute is populated with a new unique number.

Due to the *NOT NULL* constraints I placed on all the foreign keys, the *rental\_fact\_id* table will not accept rows of data with null values in the foreign key columns. Because I don’t have a way to reconcile missing data, I elected to delete rows with missing data in foreign key columns. To do so, I used the *DELETE* command. The *FROM* and *WHERE* commands were used as before, and the *OR* command was used according to the strict logical definition of the word *OR.*

Finally, I used the *INSERT INTO* command to populate the *rental\_fact* table with the data from the temporary table. Following the *INSERT INTO* command is the table and column declaration that will be populated by this step. I used the *SELECT* and *FROM* commands as before to define the source columns. Importantly, the order the columns are listed in defines the mapping from source to target columns.

**Lessons Learned**

The most important lesson I have learned in the completion of this project (included both milestone projects) is just how involved this entire process is. The database I used as a source is very simple. As simple as it is, creating a data warehouse still required a great deal of attention to detail. The ETL procedures also require a great deal of attention to detail to get right. This project, to me, highlighted the necessity of collaboration in this kind of project.

If I were to offer any advice to any organization embarking on this type of project, it would be to put a lot of effort into selecting a team of experts. That team needs to be to the standards set by Kimball and Ross (2013). Additionally, because of the amount of detail required in this kind of project, the coordination of the different moving pieces needs to be performed by an experienced manager that understands how to break this kind of project into smaller pieces. On that same note, breaking this project down into small, manageable pieces would be vital to ensuring success and eliminating errors.

**References**

Humphreys, J. (2023, September 3). *Portfolio project milestone 1: data warehouse star schema.* Department of Management Information Systems, Colorado State University Global.

Humphreys, J. (2023, September 17). *Portfolio project milestone 2: table creation – data warehouse.* Department of Management Information Systems, Colorado State University Global

Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The definitive guide to dimensional modeling* (3rd ed.). John Wiley & Sons, Inc.

Morenoh149.(2015). *PostgresDBSamples/pagila0.10.1.* Github. <https://github.com/morenoh149/postgresDBSamples/tree/master/pagila-0.10.1>

Oracle. (2023). *Sakila sample Database.* MySQL. <https://dev.mysql.com/doc/sakila/en/>

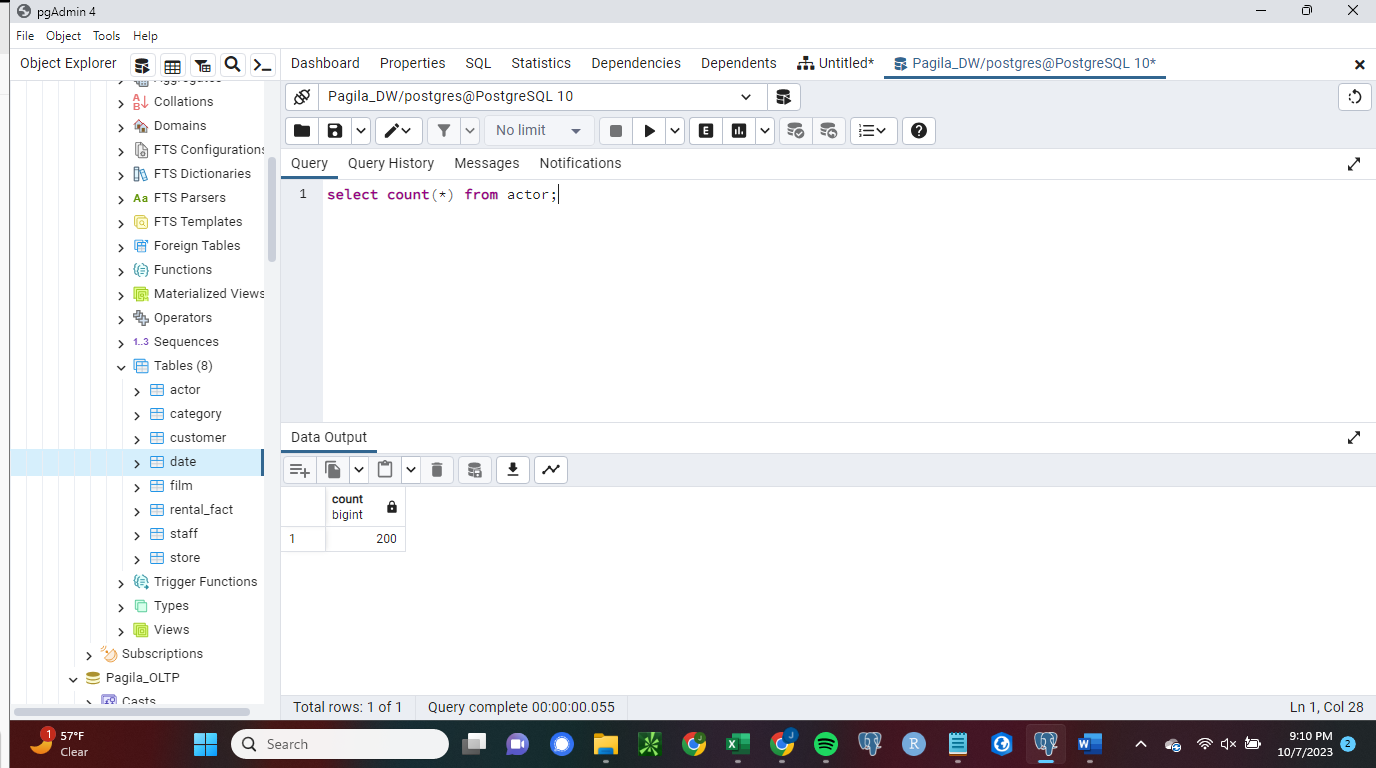
The PostgreSQL Global Development Group. (2023). *PostgreSQL 15.4 Documentation.* PostgreSQL. <https://www.postgresql.org/docs/15/index.html>

**Appendix A**

**Table Screenshots**

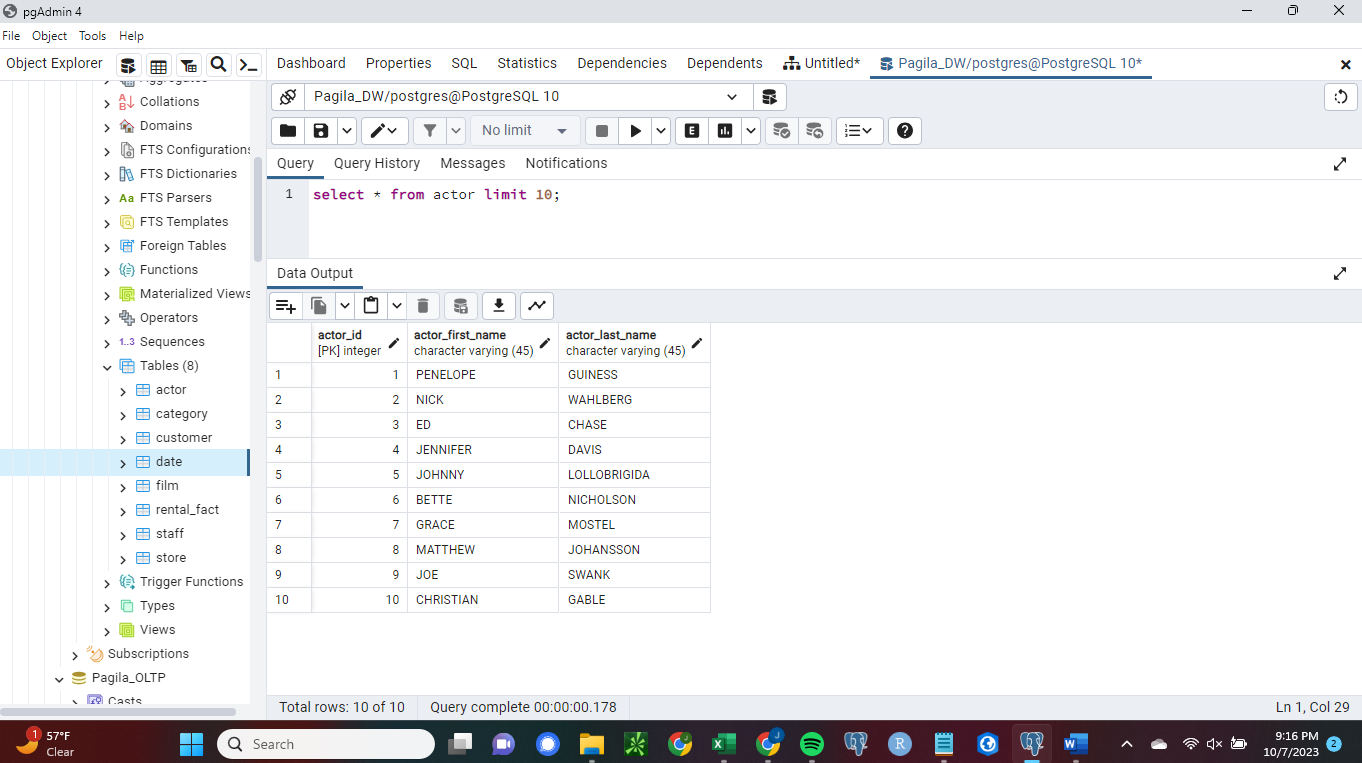
**Figure 1.**

*Screenshot of row count of actor table*

**

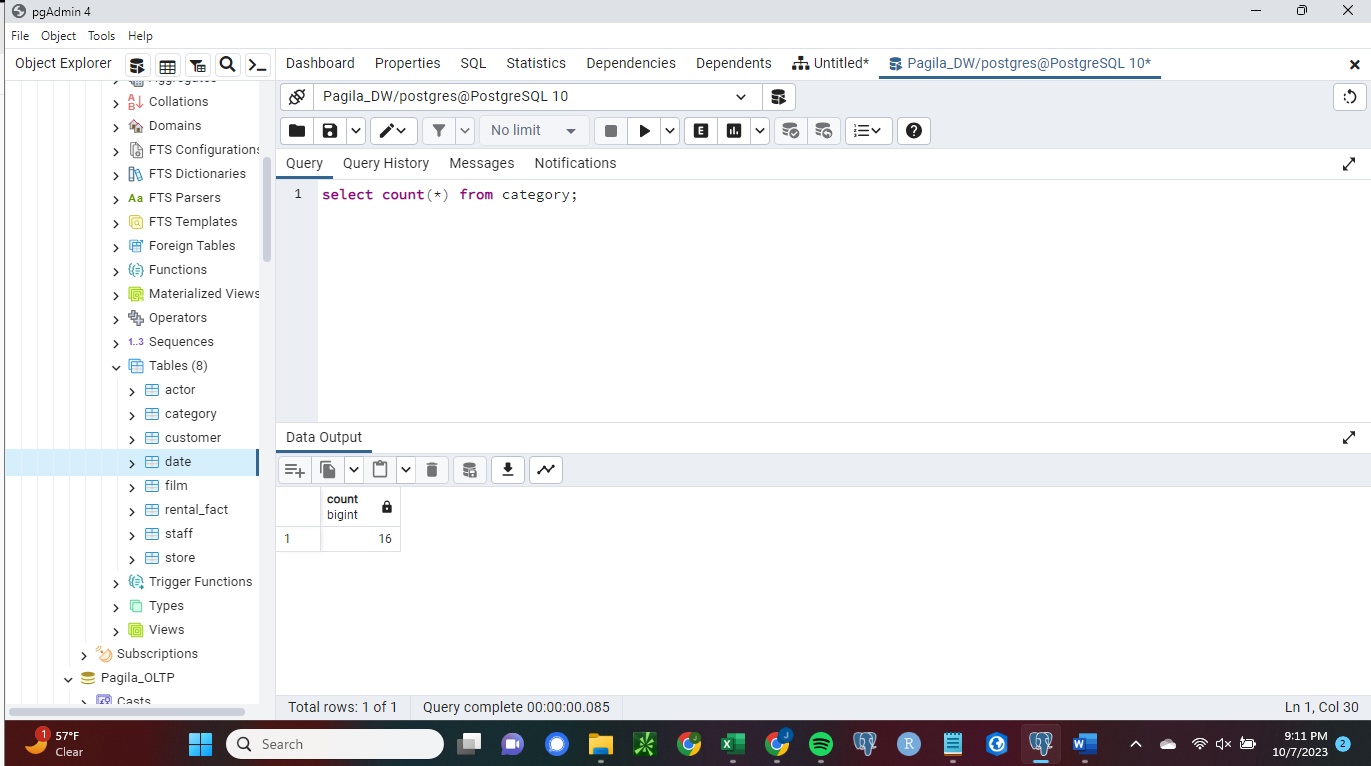
**Figure 2.**

*Screenshot of first ten rows of actor table*

**

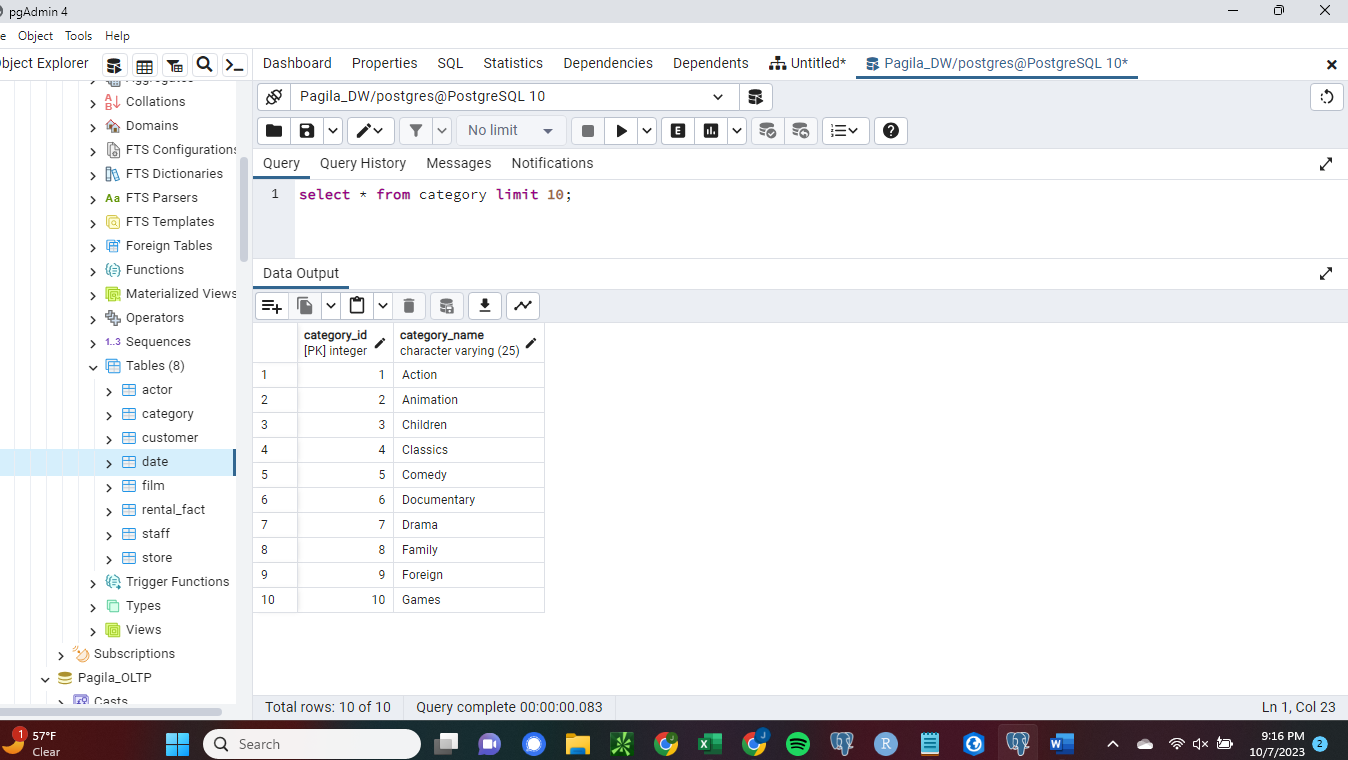
**Figure 3.**

*Screenshot of row count of category table*

****

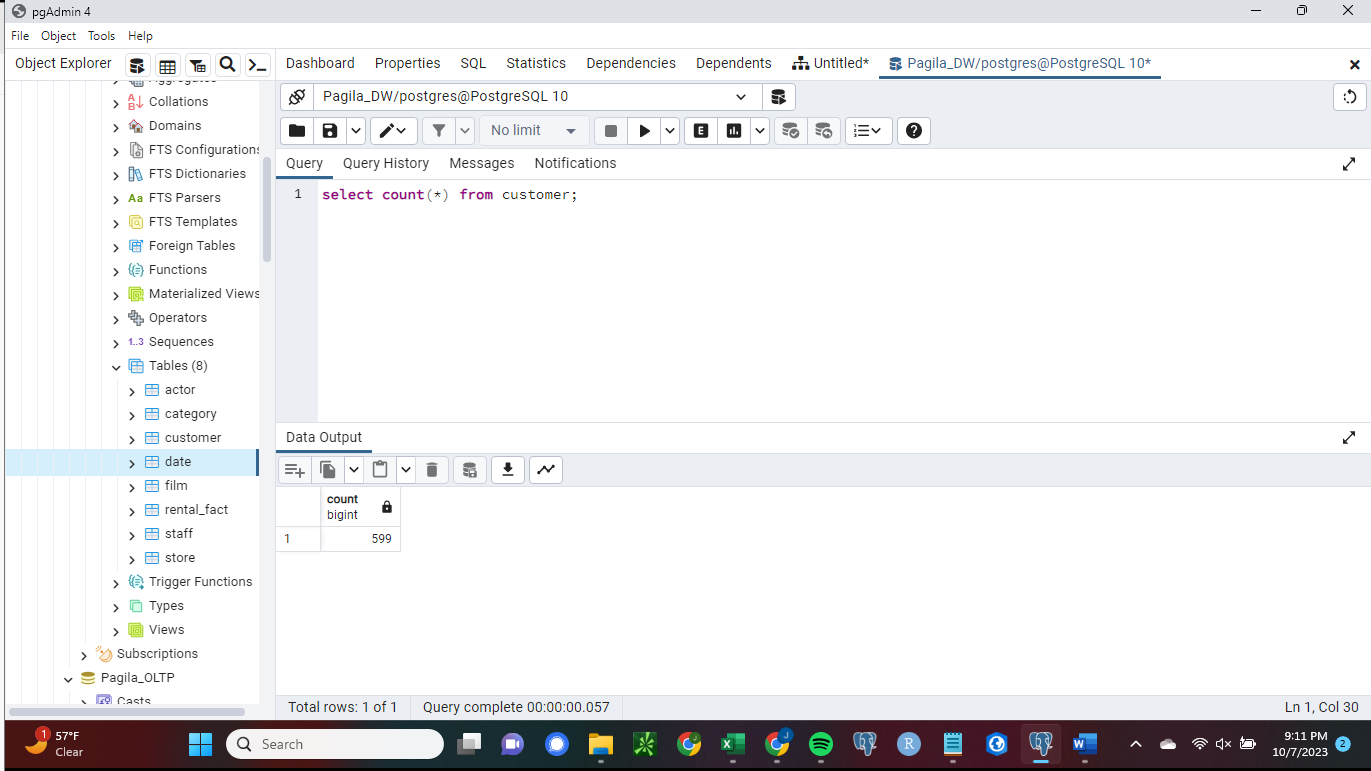
**Figure 4.**

*Screenshot of first ten rows of category table*

****

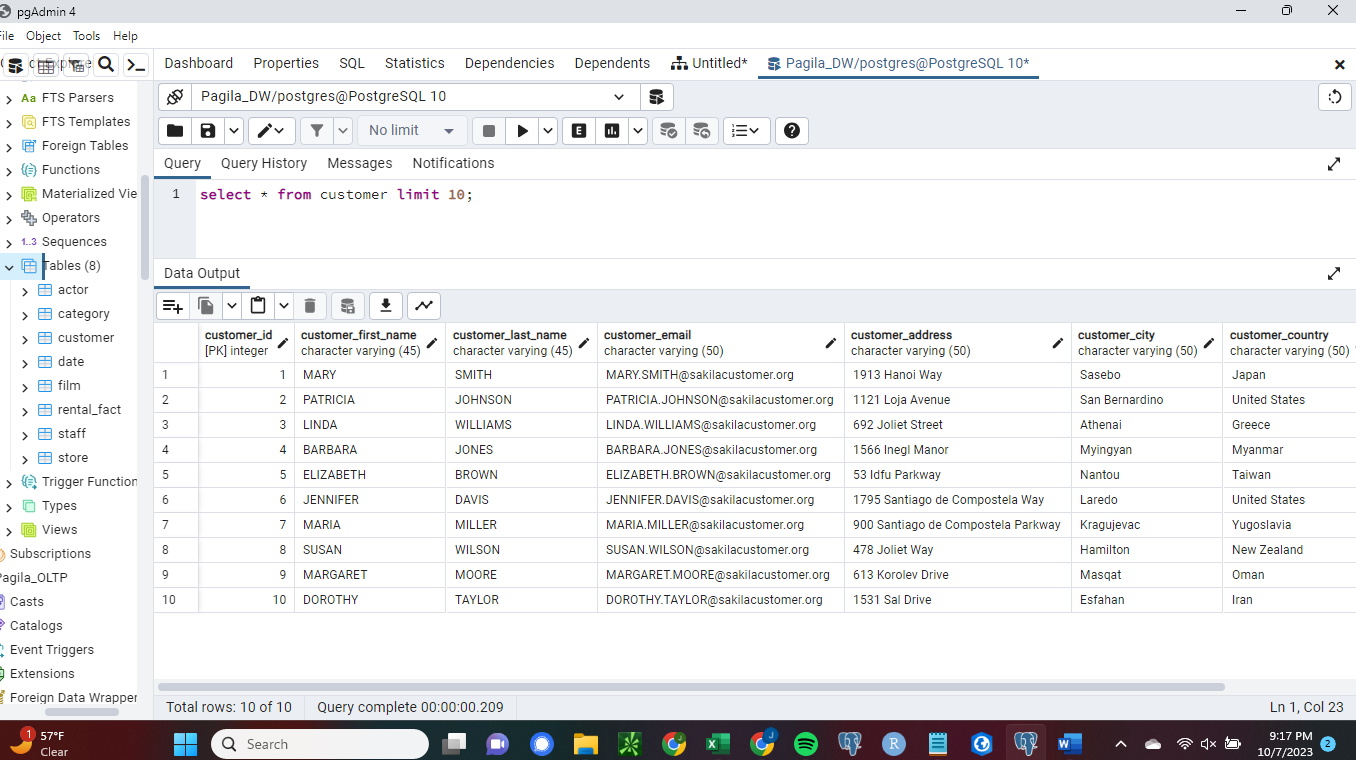
**Figure 5.**

*Screenshot of row count of customer table*

****

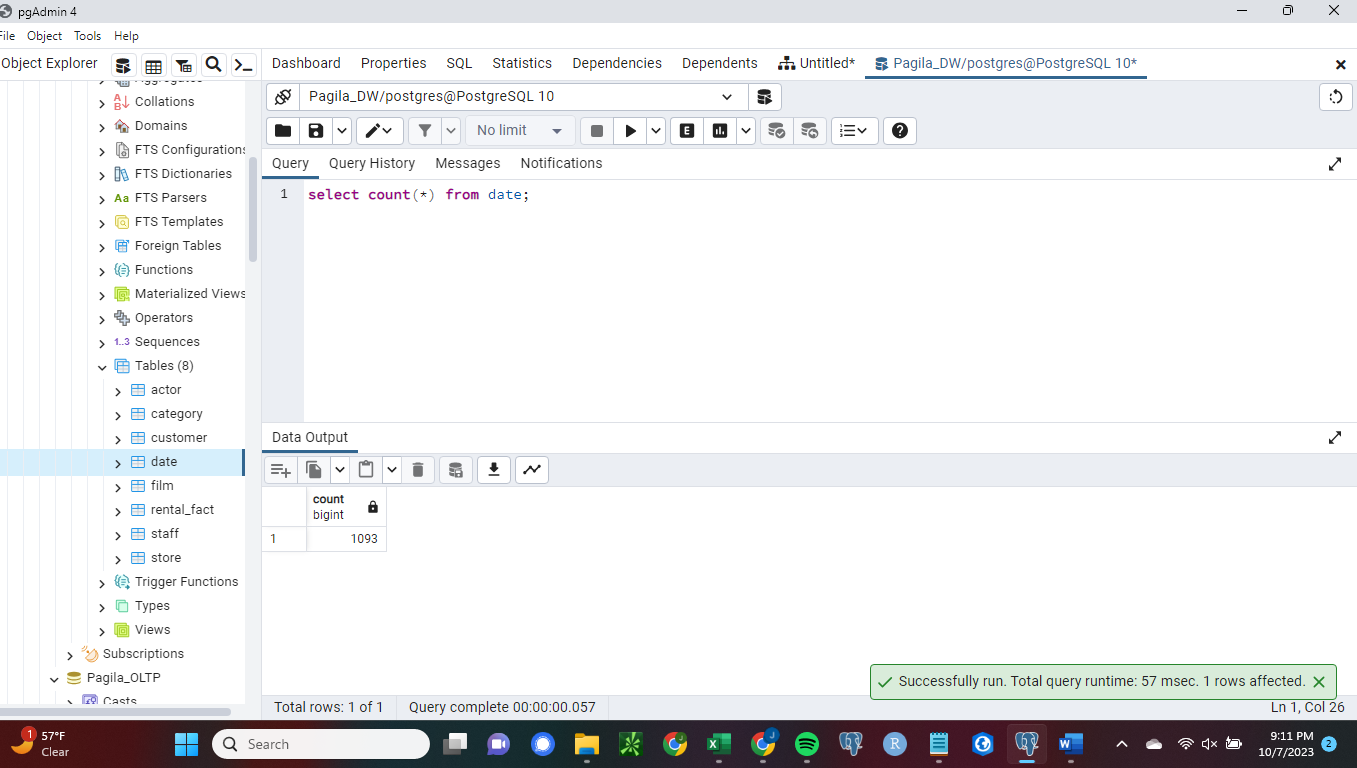
**Figure 6.**

*Screenshot of first ten rows of customer table*

****

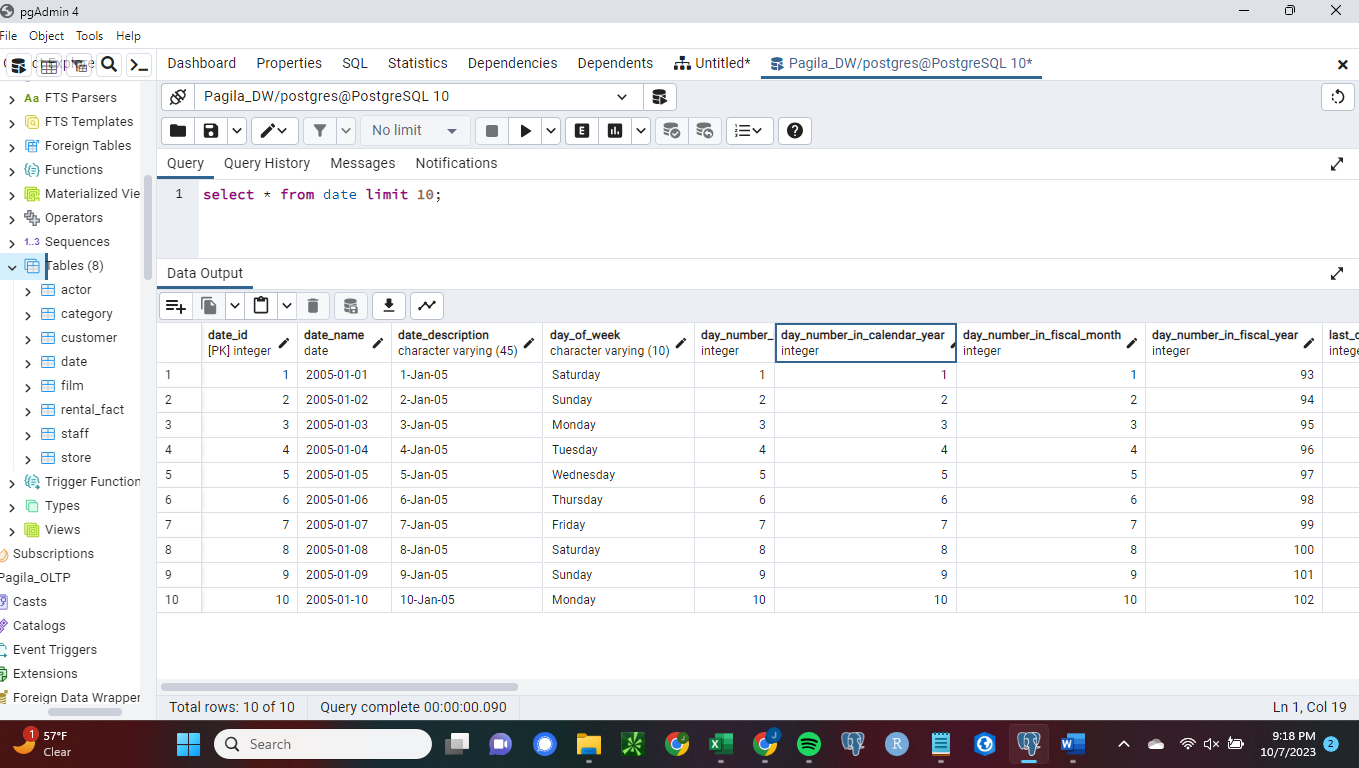
**Figure 7.**

*Screenshot of row count of date table*

****

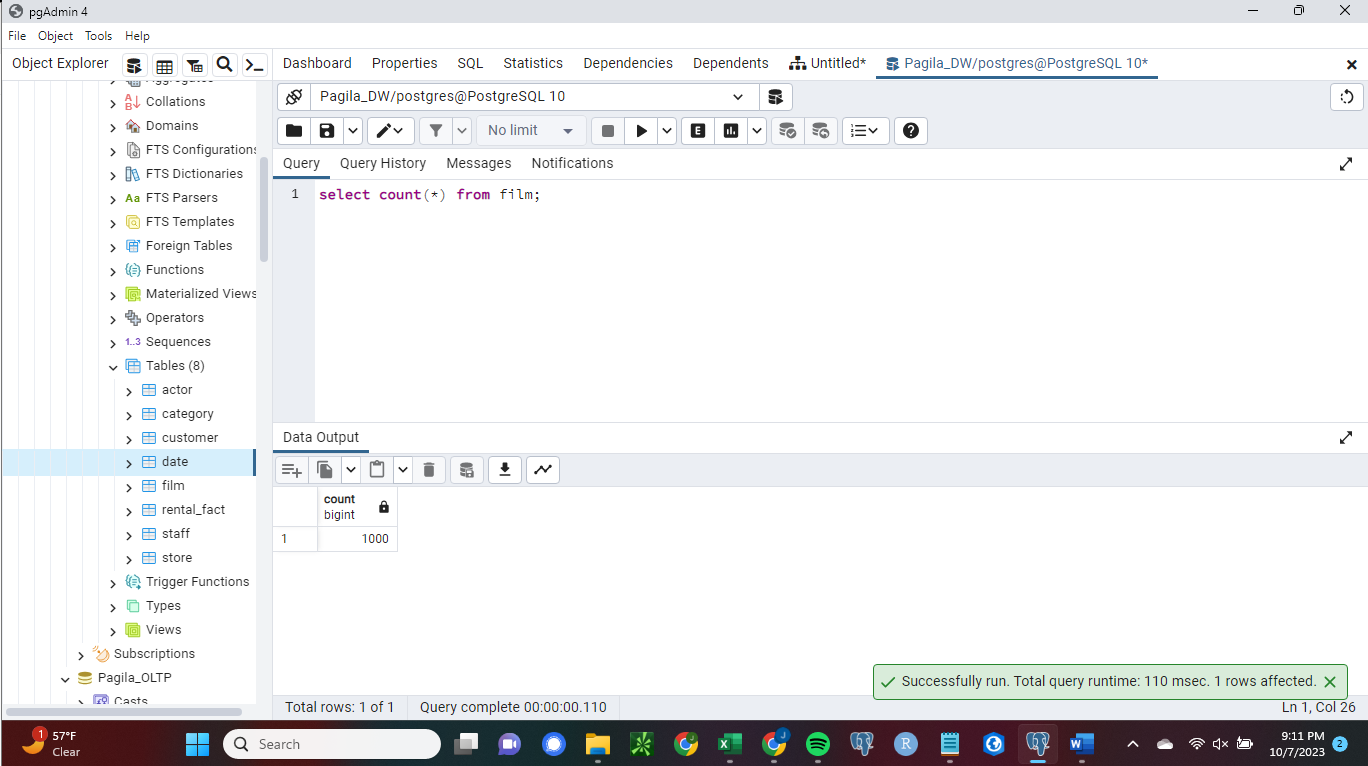
**Figure 8.**

*Screenshot of first ten rows of date table*

****

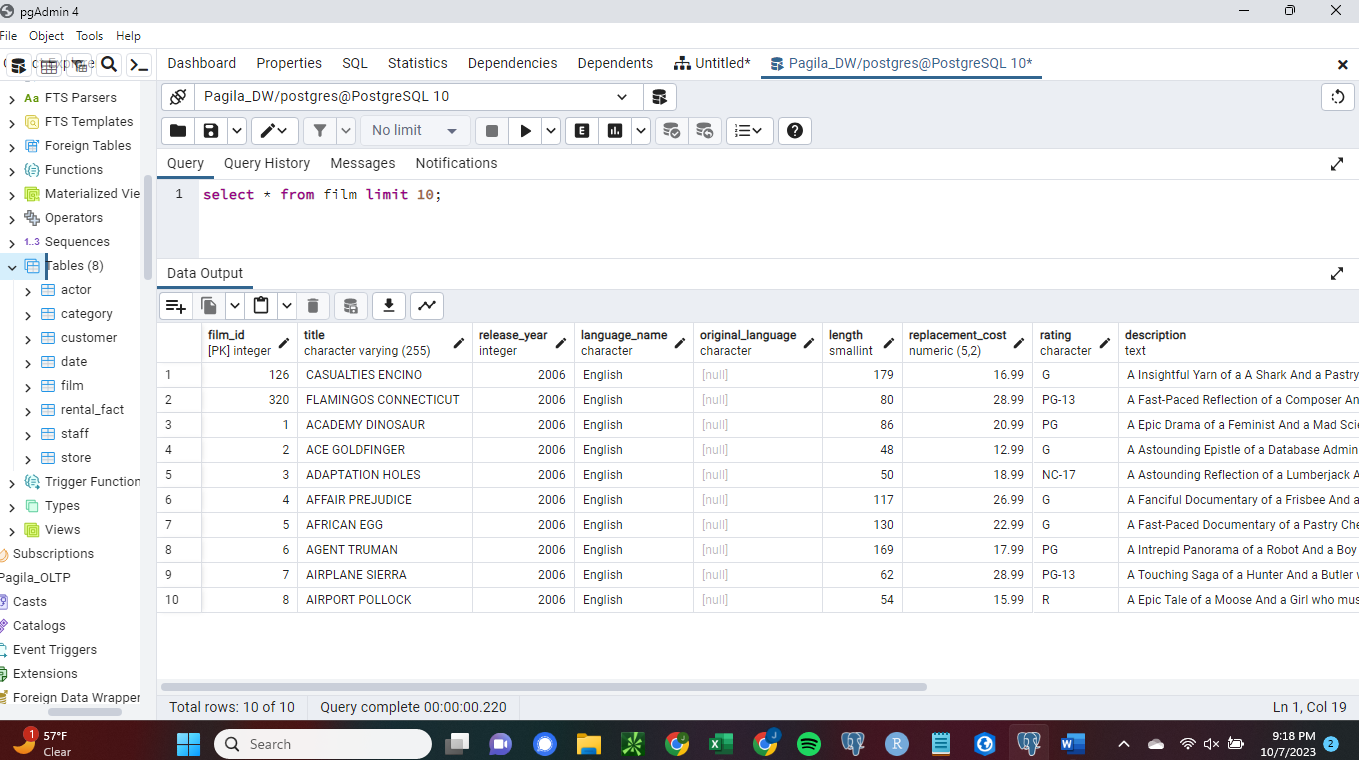
**Figure 9.**

*Screenshot of row count of film table*

****

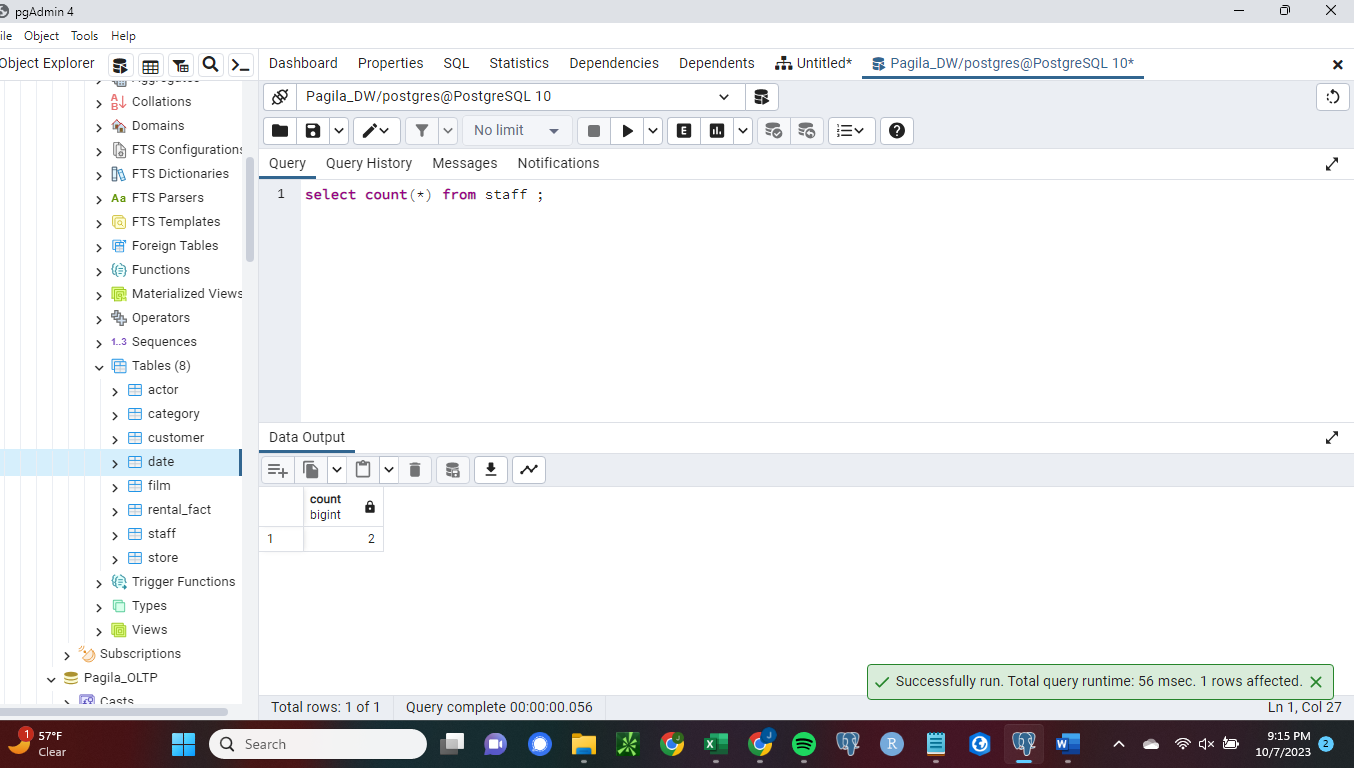
**Figure 10.**

*Screenshot of first ten rows of film table*

**

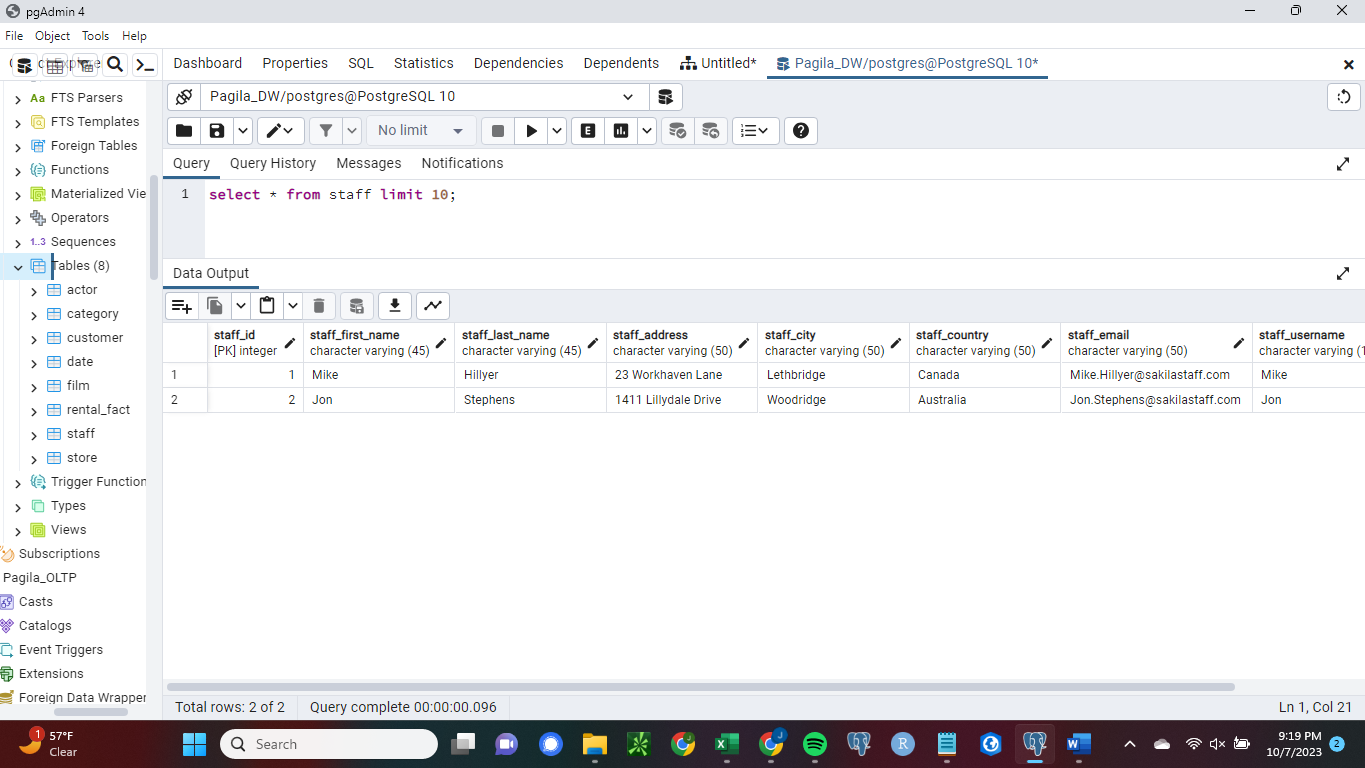
**Figure 11.**

*Screenshot of row count of staff table*

****

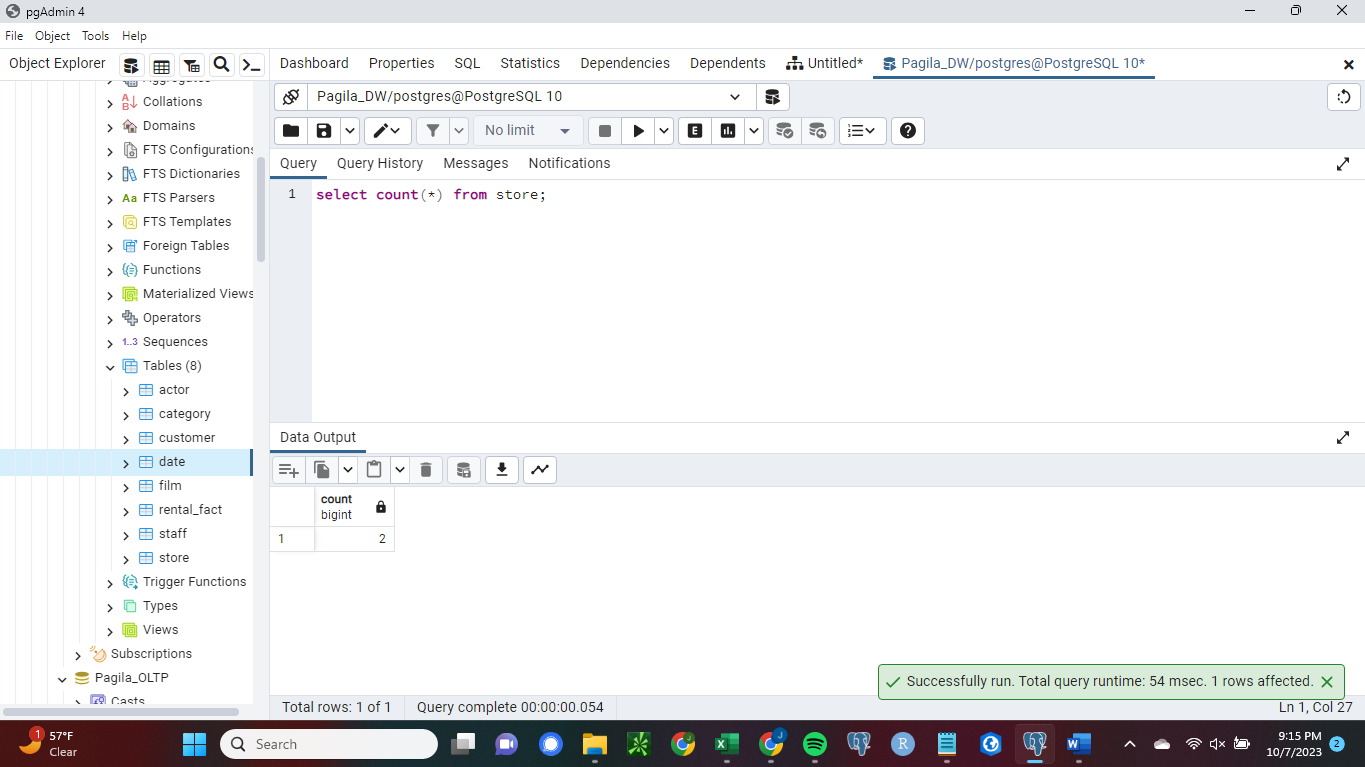
**Figure 12.**

*Screenshot of first ten rows of staff table*

****

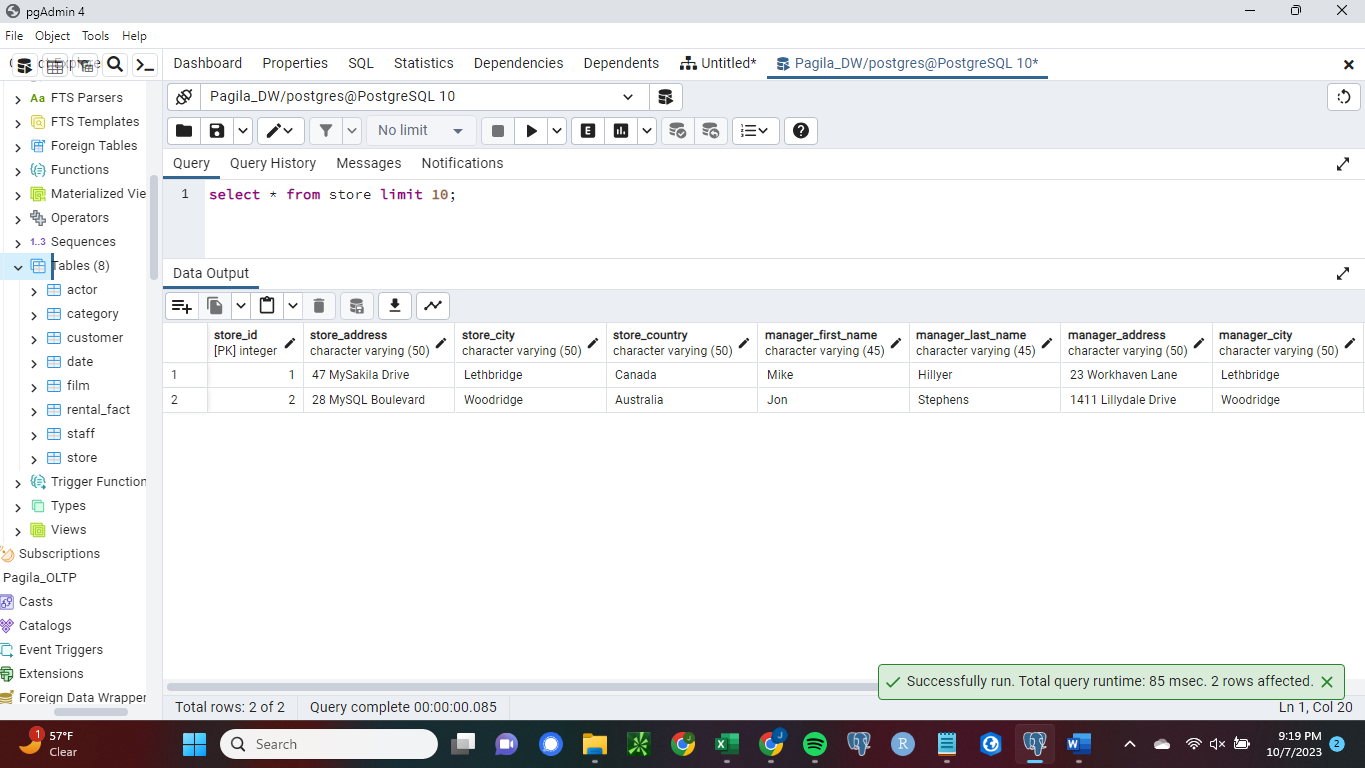
**Figure 13.**

*Screenshot of row count of store table*

****

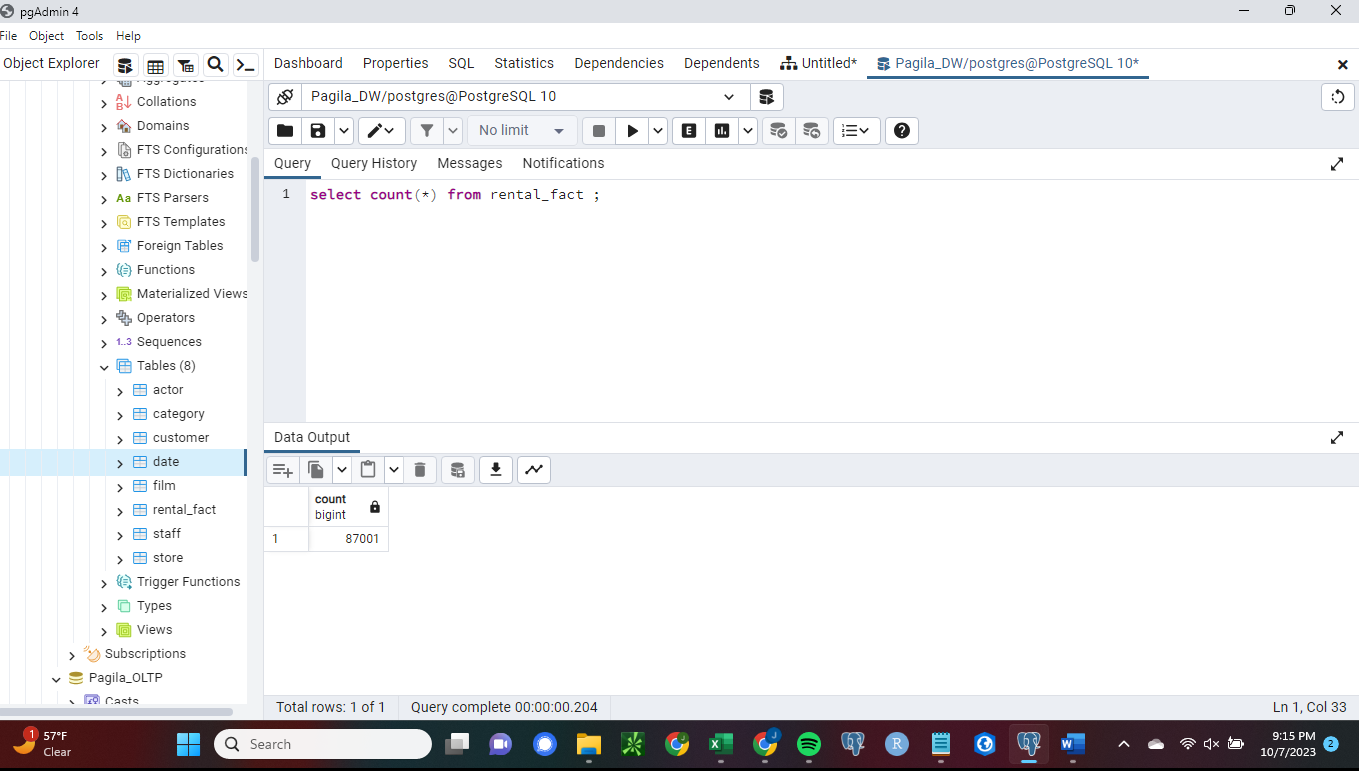
**Figure 14.**

*Screenshot of first ten rows of store table*

****

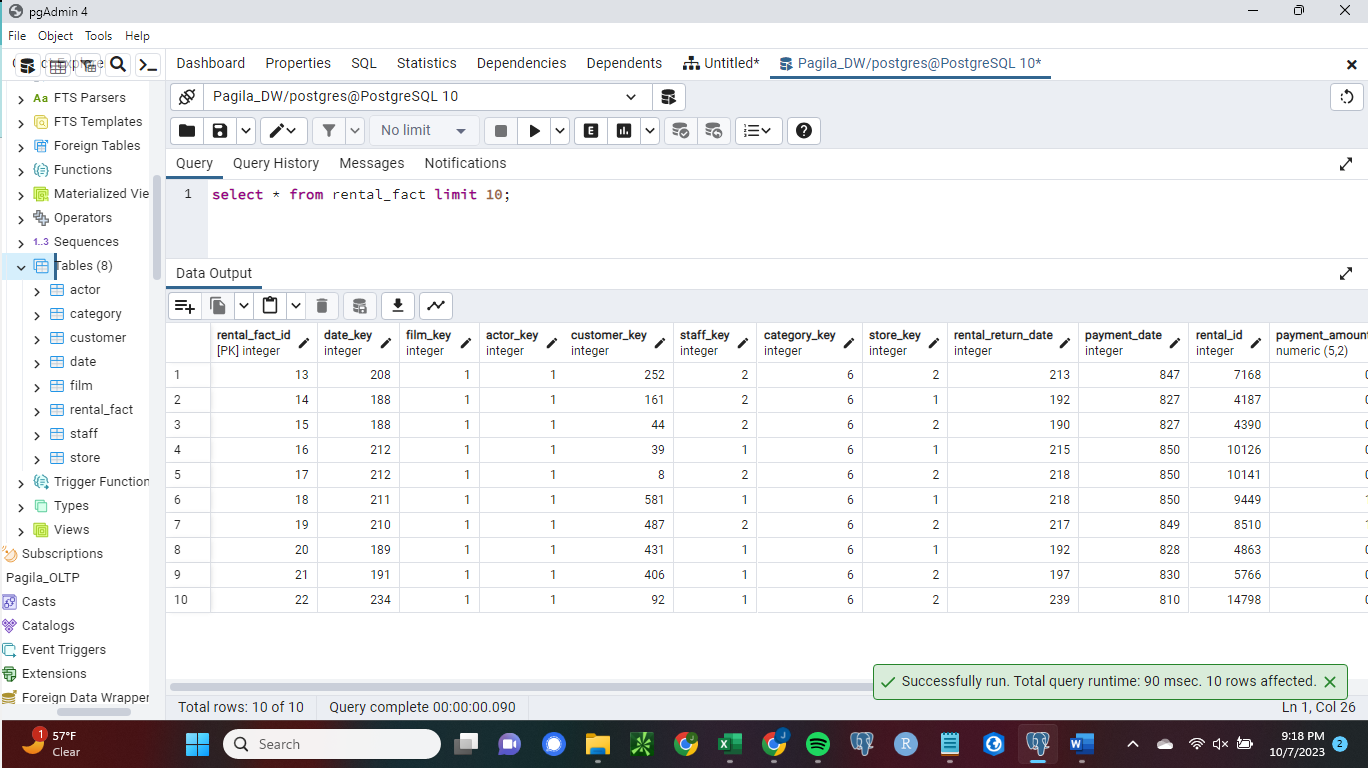
**Figure 15.**

*Screenshot of row count of rental\_fact table*

****

**Figure 16.**

*Screenshot of first ten rows of rental\_fact table*

**

**Appendix A**

**Listing of SQL Commands for ETL Procedures**

--copy the store manager data for the store dimension to csv

copy(select store.store\_id, first\_name, last\_name, address, city, country, email, username, picture from store inner join (staff inner join (address inner join (city inner join country using(country\_id)) using(city\_id))using(address\_id)) on store.manager\_staff\_id=staff.staff\_id)

to'/users/public/dim\_store.manager.csv'

delimiter ','

csv header;

--copy the remaining store dimension data to csv

copy(select store.store\_id, address, city, country from store inner join (address inner join (city inner join country using(country\_id)) using(city\_id))using(address\_id))

to'/users/public/dim\_store.csv'

delimiter ','

csv header;

--copy staff dimension data to csv

copy (select staff\_id, first\_name, last\_name, address, city, country, email, username, picture from staff inner join (address inner join (city inner join country using(country\_id)) using(city\_id))using(address\_id))

to'/users/public/dim\_staff.csv'

delimiter ','

csv header;

--copy actor dimension data to csv

copy (select actor\_id, first\_name, last\_name from actor)

to '/users/public/dim\_actor.csv'

delimiter ','

csv header;

--copy category dimension data to csv

copy (select category\_id, name from category)

to '/users/public/dim\_category.csv'

delimiter ','

csv header;

--copy customer dimension data to csv

copy (select customer\_id, first\_name, last\_name, email, address, city, country, active from customer inner join (address inner join (city inner join country using(country\_id)) using(city\_id))using (address\_id))

to '/users/public/dim\_customer.csv'

delimiter ','

csv header;

--copy data except for original language for the film dimension to csv

copy (select film\_id, title, release\_year, language.name, length, replacement\_cost, rating, description, special\_features

from film inner join language using (language\_id) )

to '/users/public/dim\_film\_no\_orig\_lang.csv'

delimiter ','

csv header;

--copy original language data for film dimension to csv

copy (select film\_id, language.name

from film inner join language on (film.original\_language\_id=language.language\_id) )

to '/users/public/dim\_film\_orig\_lang.csv'

delimiter ','

csv header;

--copy rental fact data to csv

copy (select date(rental\_date), film\_id, actor\_id, payment.customer\_id, payment.staff\_id,

category\_id, store\_id, date(return\_date), date(payment\_date), rental\_id, amount

from payment inner join (rental inner join (inventory inner join(film inner join (film\_actor inner join film\_category

using(film\_id)) using(film\_id))using(film\_id))using(inventory\_id)) using(rental\_id))

to '/users/public/rental\_fact.csv'

delimiter ','

csv header;

--copy store manager data from csv to store dimension table

copy store(store\_id, manager\_first\_name, manager\_last\_name, manager\_address, manager\_city, manager\_country, manager\_email, manager\_username, manager\_picture)

from '/users/public/dim\_store.manager.csv'

delimiter ',' csv header;

--create a temporary table for remaining store dimension table data

create temp table temp\_table1(

store\_id integer,

store\_address character varying(50),

store\_city character varying(50),

store\_country character varying(50)

);

--copy remaining store dimension data to temporary table

copy temp\_table1(store\_id, store\_address, store\_city, store\_country)

from '/users/public/dim\_store.csv'

delimiter ','

csv header;

--update store dimension table with remaining data from temporary table

Update store

set (store\_address, store\_city, store\_country)=

(select store\_address, store\_city, store\_country

From temp\_table1 where store.store\_id=temp\_table1.store\_id);

--drop the temporary table

drop table temp\_table1;

--copy data from csv to staff dimension table

copy staff

from '/users/public/dim\_staff.csv'

delimiter ','

csv header;

--copy data from csv to actor dimension table

copy actor

from '/users/public/dim\_actor.csv'

delimiter ','

csv header;

--copy data from csv to category dimension table

copy category

from '/users/public/dim\_category.csv'

delimiter ','

csv header;

-- copy data from csv to customer dimension table

copy customer

from '/users/public/dim\_customer.csv'

delimiter ','

csv header;

--copy data except original language from csv to film dimension table

copy film(film\_id, title, release\_year, language\_name, length, replacement\_cost, rating, description, special\_features)

from '/users/public/dim\_film\_no\_orig\_lang.csv'

delimiter ','

csv header;

--create temporary table for original language data

create temp table temp\_table1(

film\_id integer,

original\_language\_name character(20)

);

--copy original language data from csv to temporary table

copy temp\_table1

from '/users/public/dim\_film\_orig\_lang.csv'

delimiter ','

csv header;

--insert original language data from temporary table in to film dimension

update film

set original\_language=(select original\_language\_name from temp\_table1 where film.film\_id=temp\_table1.film\_id);

--drop temporary table

drop table temp\_table1;

--copy date dimension data from csv

copy date

from '/users/public/dim\_date1.txt'

delimiter ','

csv header;

--create temporary table for rental fact table

--temporary table includes columns for date and date keys

--date values of date type were extracted from source database

create temp table temp\_fact\_table(

date\_key integer,

rental\_date date,

film\_key integer,

actor\_key integer,

customer\_key integer,

staff\_key integer,

category\_Key integer,

store\_key integer,

rental\_return\_date\_key integer,

rental\_return\_date date,

payment\_date\_key integer,

payment\_date date,

rental\_id integer,

payment\_amount numeric(5,2) );

--copy data from csv to rental fact table

copy temp\_fact\_table(rental\_date, film\_key, actor\_key,

customer\_key, staff\_key, category\_key,

store\_key, rental\_return\_date, payment\_date, rental\_id, payment\_amount)

from '/users/public/rental\_fact.csv'

delimiter ','

csv header;

--copy date key from date dimension for corresponding rental dates to temporary table

update temp\_fact\_table

set date\_key=(select date\_idfrom date where temp\_fact\_table.rental\_date=date.date\_name);

--copy date key from date dimension for corresponding rental return date to temporary table

update temp\_fact\_table

set rental\_return\_date\_key=(select date\_id from date where temp\_fact\_table.rental\_return\_date=date.date\_name);

--alter rental\_fact\_id to automatically populate as surrogate key

alter table rental\_fact

alter column rental\_fact\_id add generated always as identity;

--delete rows with missing foregn key data due to not null constraint

Delete from temp\_fact\_table

where date\_key is null or film\_key is null or actor\_key is null or

customer\_key is null or staff\_key is null or category\_key is null or

store\_key is null or rental\_return\_date\_key is null or payment\_date\_key is null;

--insert data except dates from temporary table into rental fact table

insert into rental\_fact(date\_key, film\_key, actor\_key, customer\_key, staff\_key, category\_key,

store\_key, rental\_return\_date, payment\_date, rental\_id, payment\_amount)

select date\_key, film\_key, actor\_key, customer\_key, staff\_key, category\_key,

store\_key, rental\_return\_date\_key, payment\_date\_key, rental\_id, payment\_amount from temp\_fact\_table;