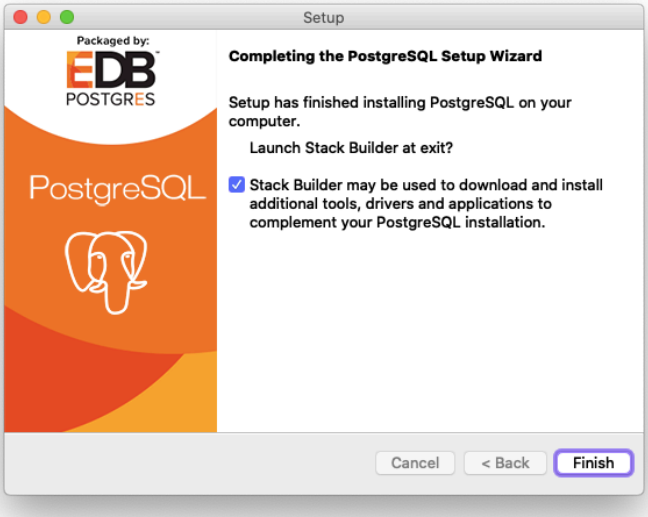
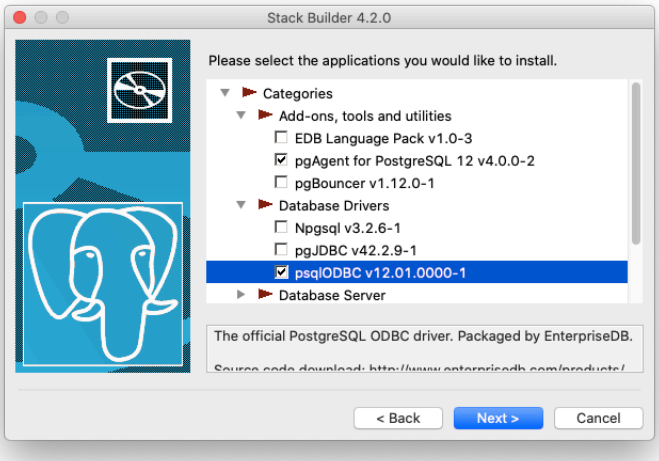
MSDE 631: Lab 4
Etana Disasa

Part 1: Install PostgreSQL

Pre Installation Summary

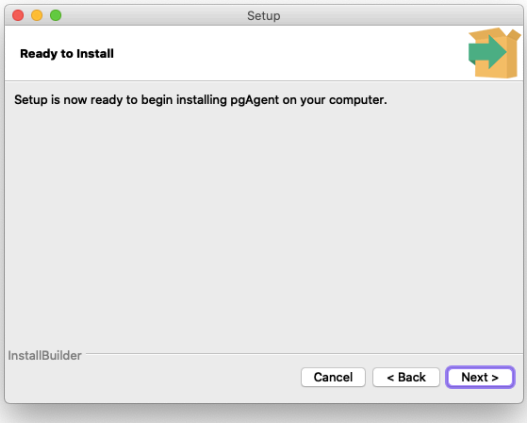
Installation Directory: /Library/PostgreSQL/10
Server Installation Directory: /Library/PostgreSQL/10
Data Directory: /Library/PostgreSQL/10/data
Database Port: 5433
Database Superuser: postgres
Operating System Account: postgres
Database Service: postgresql-10
Command Line Tools Installation Directory: /Library/PostgreSQL/10
pgAdmin4 Installation Directory: /Library/PostgreSQL/10/pgAdmin 4
Stack Builder Installation Directory: /Library/PostgreSQL/10

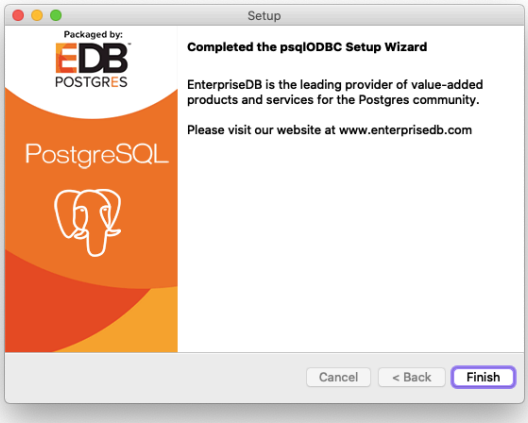




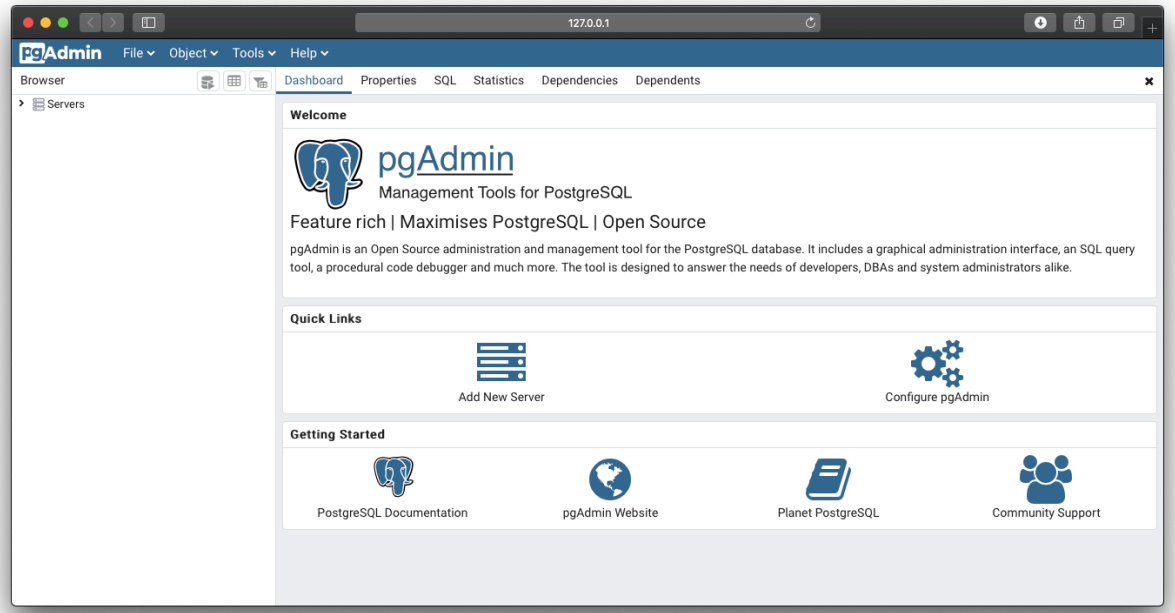
MSDE 631: Lab 4
Etana Disasa

During this installation, pgAgent and psqlODBC failed to install. I had individually run the installation files from where Stack Builder had downloaded them for the installation.





And now pgAdmin 4 is up and running.

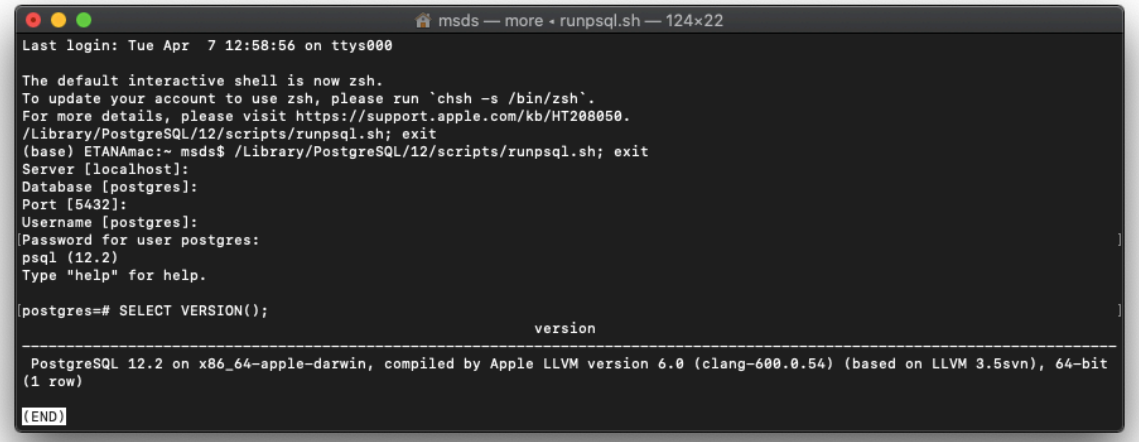


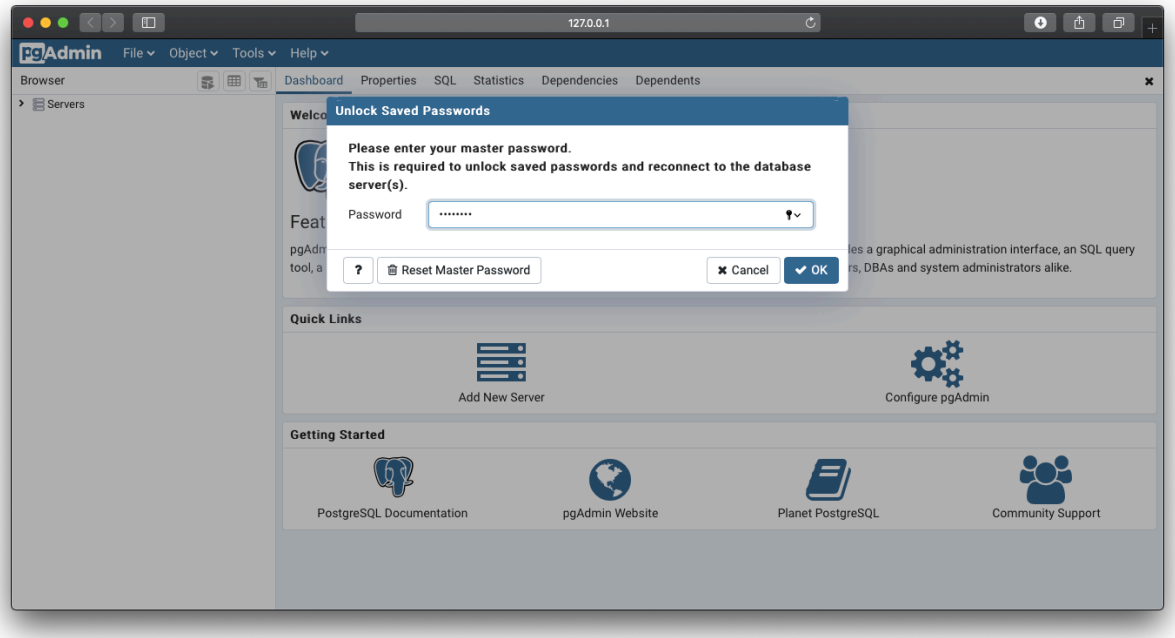
END OF PART ONE

MSDE 631: Lab 4
Etana Disasa

Part 2: How to Use PostgreSQL

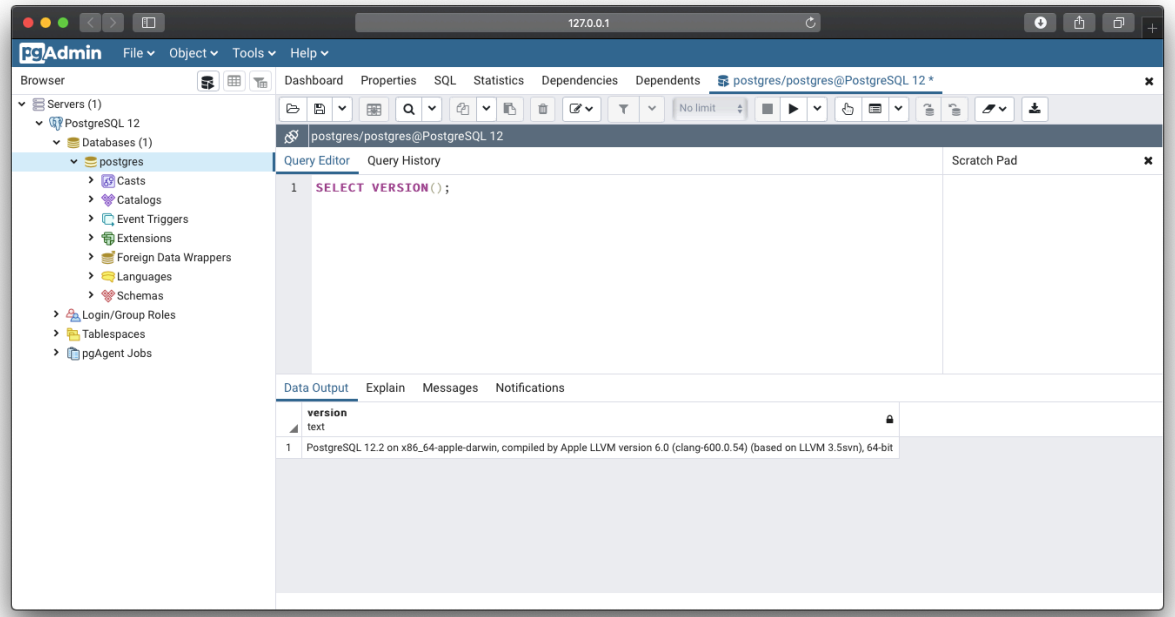
SQL Shell (psql)





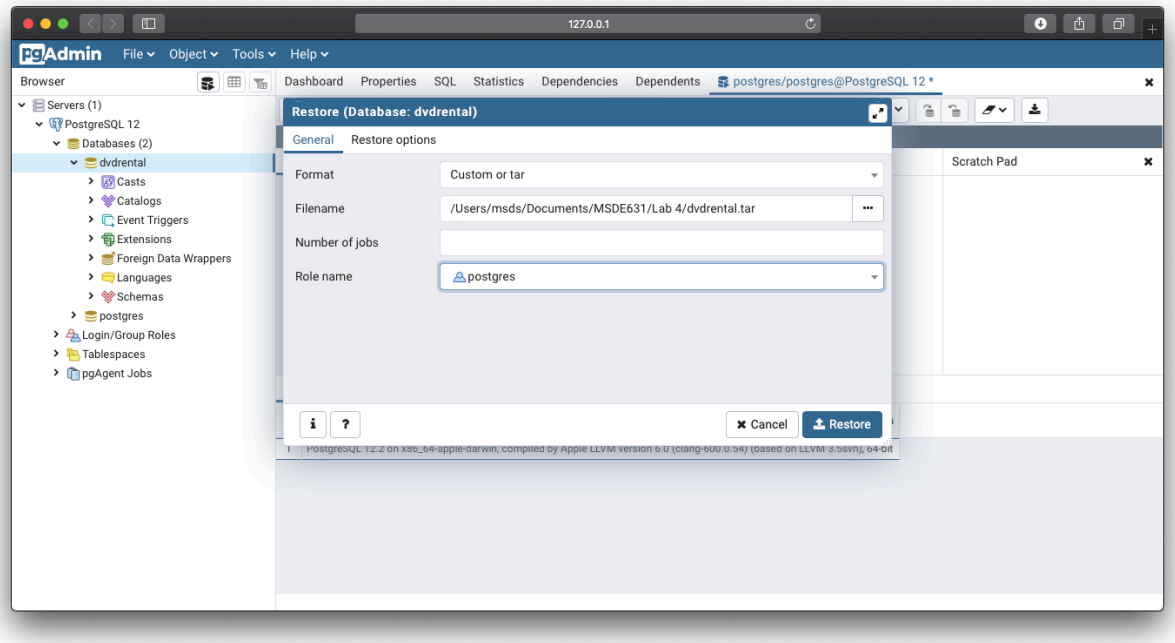
MSDE 631: Lab 4
Etana Disasa

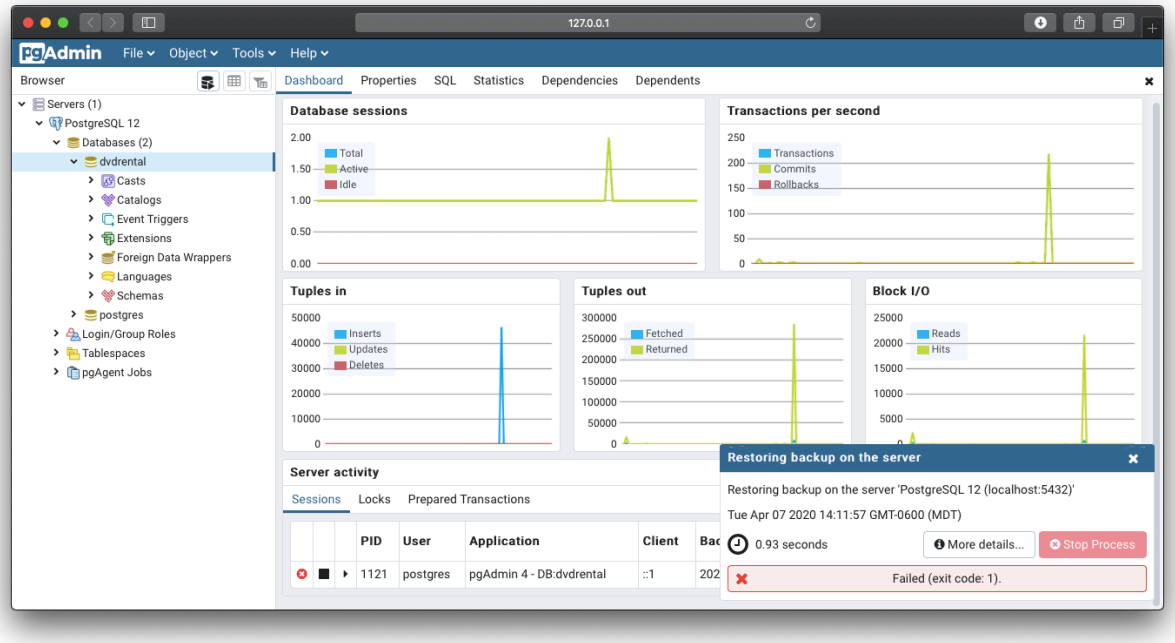
In this new version the ⚡ (lightening bolt) is not changed into a traditional ▶ (execute) button.



MSDE 631: Lab 4
Etana Disasa

Creating and Restoring Database (dvdrental)



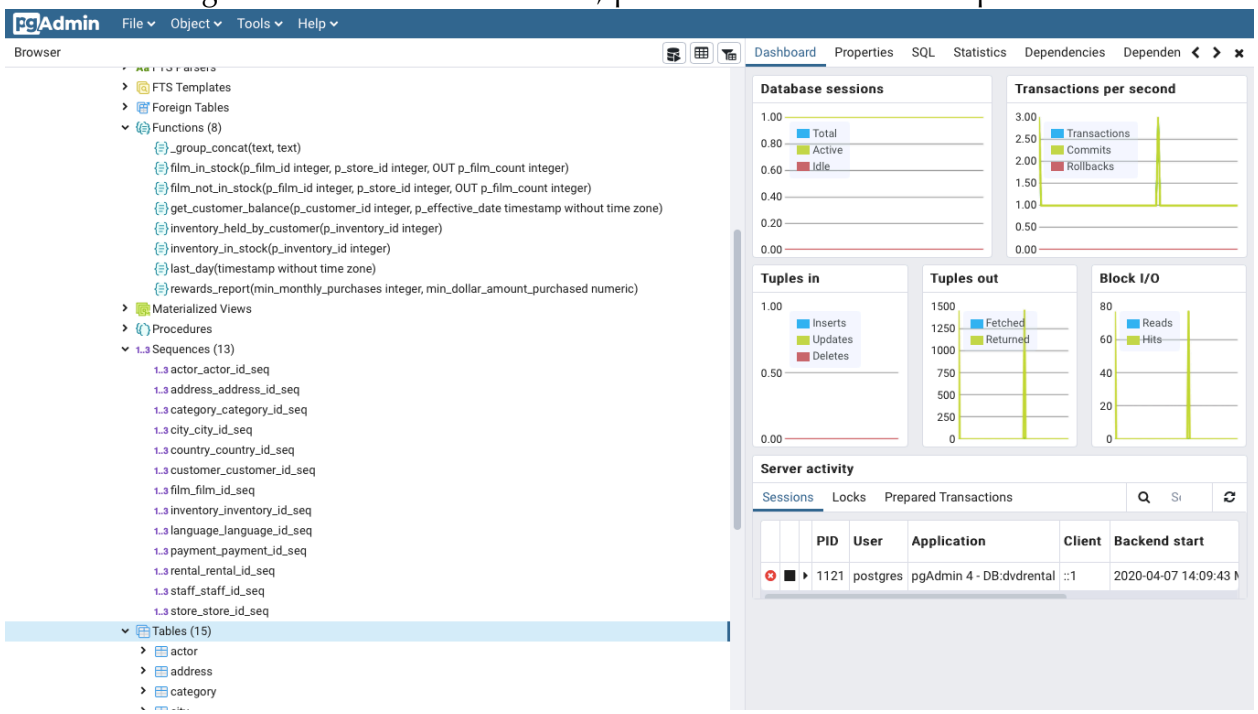


My installation was giving me problems. It appears that this exit code-1 error appears if the restore process is attempting to add an IPS_name to the Schema. "If the IPS_username is already configured in the Schema or is different from the IPS_username in the old PostgreSQL database, it will return exit code 1. This error usually can be ignored."[1]

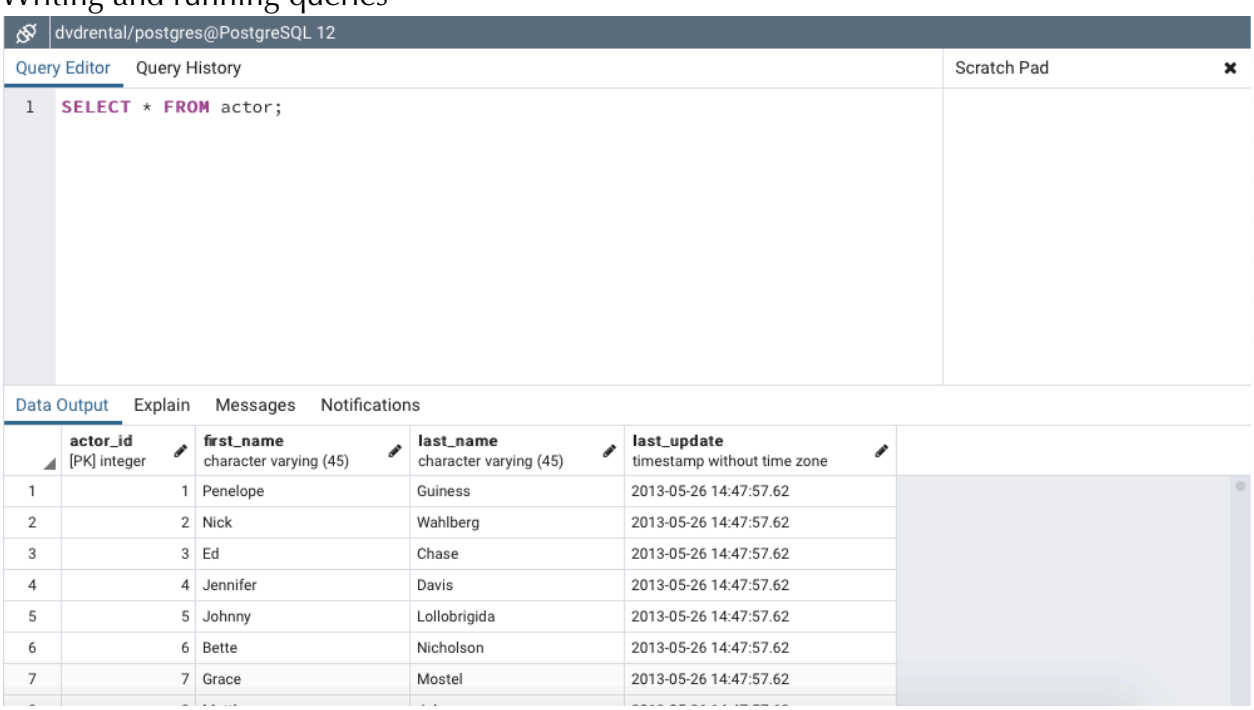
[1] https://community.ipswitch.com/s/article/Exit-Code-1-When-Restoring-PostgreSQL-Database?r=6&ui-force-components-controllers-recordGlobalValueProvider.RecordGvp.getRecord=1

MSDE 631: Lab 4
Etana Disasa

Below the image shows that all the schema, procedures and tables are present.



Writing and running queries



MSDE 631: Lab 4
Etana Disasa

Views

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History | Scratch Pad

```
SELECT * FROM customer_list;
```

Data Output | Explain | Messages | Notifications

	id integer	name text	address character varying (50)	zip code character varying (10)	phone character varying (20)	city character varying (50)	country character varying (50)	notes text
1	524	Jared Ely	1003 Qinhuangdao Street	25972	35533115997	Purwakarta	Indonesia	active
2	1	Mary S...	1913 Hanoi Way	35200	28303384290	Sasebo	Japan	active
3	2	Patricia...	1121 Loja Avenue	17886	838635286649	San Bernardino	United States	active
4	3	Linda ...	692 Joliet Street	83579	448477190408	Athenai	Greece	active
5	4	Barbar...	1566 Inegl Manor	53561	705814003527	Myingyan	Myanmar	active
6	5	Elizabe...	53 Idfu Parkway	42399	10655648674	Nantou	Taiwan	active
7	6	Jennife...	1795 Santiago de Composte...	18743	860452626434	Laredo	United States	active
8	7	Maria ...	900 Santiago de Compostel...	93896	716571220373	Kragujevac	Yugoslavia	active
9	8	Susan ...	478 Joliet Way	77948	657282285970	Hamilton	New Zealand	active

View properties in PostgreSQL 12 appears a bit different. The *Code* tab displays the sql code for the view which would have been found in the *Definition* tab in older versions.

customer_list

General | Definition | Code | Security | SQL

```
SELECT cu.customer_id AS id,
   (cu.first_name::text || ' '::text) || cu.last_name::text AS name,
   a.address,
   a.postal_code AS "zip code",
   a.phone,
   city.city,
   country.country,
       CASE
           WHEN cu.activebool THEN 'active'::text
           ELSE ''::text
       END AS notes,
   cu.store_id AS sid
  FROM customer cu
    JOIN address a ON cu.address_id = a.address_id
    JOIN city ON a.city_id = city.city_id
    JOIN country ON city.country_id = country.country_id;
```

Cancel | Reset | Save

MSDE 631: Lab 4
Etana Disasa

Writing queries with views

```
SELECT * FROM sales_by_film_category
WHERE total_sales > 4000;
```

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
SELECT * FROM sales_by_film_category
WHERE total_sales > 4000;
```

Data Output | Explain | Messages | Notifications

	category character varying (25)	total_sales numeric
1	Sports	4892.19
2	Sci-Fi	4336.01
3	Animation	4245.31
4	Drama	4118.46
5	Comedy	4002.48

The sales_by_film_category view looks like this

sales_by_film_category

General | Definition | Code | Security | SQL

```
SELECT c.name AS category,
   sum(p.amount) AS total_sales
  FROM payment p
    JOIN rental r ON p.rental_id = r.rental_id
    JOIN inventory i ON r.inventory_id = i.inventory_id
    JOIN film f ON i.film_id = f.film_id
    JOIN film_category fc ON f.film_id = fc.film_id
    JOIN category c ON fc.category_id = c.category_id
 GROUP BY c.name
 ORDER BY (sum(p.amount)) DESC;
```

Cancel | Reset | Save

MSDE 631: Lab 4
Etana Disasa

```
SELECT * FROM sales_by_store
```

dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT * FROM sales_by_store;
```

Data Output | Explain | Messages | Notifications

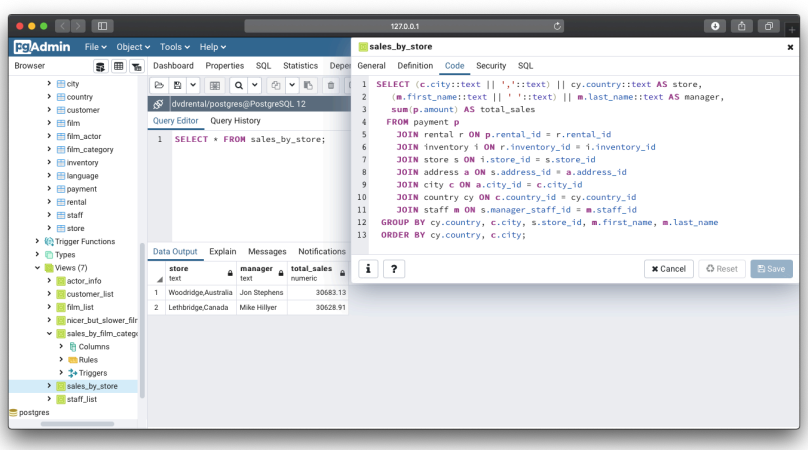
	store text	manager text	total_sales numeric
1	Woodridge,Australia	Jon Stephens	30683.13
2	Lethbridge,Canada	Mike Hillyer	30628.91

Question

1. How many tables are joined by this SQL statement within the view?
2. Why does the "group by" clause contain so many column names?

Answer

1. There are 8 tables that are joined together. And these tables are **`payment, rental, inventory, store, address, city, country,`** and **`staff.`**
2. The "group by" clause contains these many columns in order to determine the total sum of sales completed by individuals (**`m.first_name, m.last_name`**), by their store id (**`s.store_id),`** by city (**`c.city)`** and by country (**`cy.country`**). Otherwise, we will have multiple individual's that share the same first/last names from different cities will have their sales added together. The same thing to stores with similar id but in different cities or even countries can have their sales added together. Therefore, the "group by" columns rule out any error of adding the sales of individuals by error.



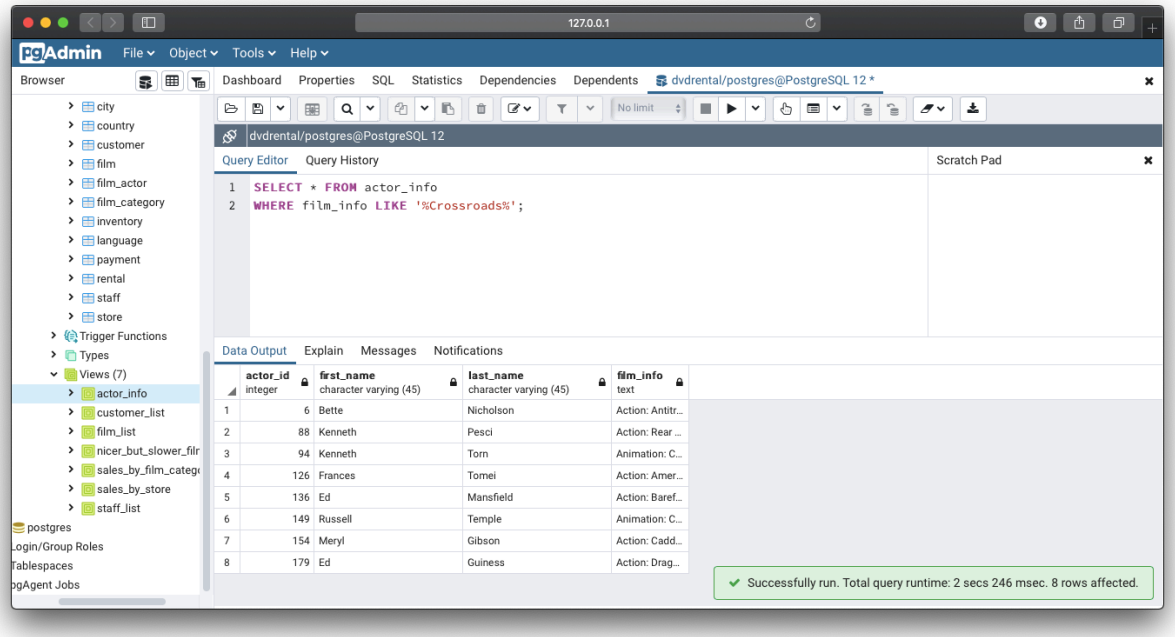
MSDE 631: Lab 4
Etana Disasa

Problem

Modify the query to select from the actor_info view. Your result set should match the results shown below exactly using some type of filtering command. You want to find rows that contain the words 'Crossroads' as shown below.

Solution

```
SELECT * FROM public.actor_info
WHERE film_info LIKE '%Crossroads%';
```



Question

3. What does the 'Group By' do in the SQL?
4. What does the nested query do in the SQL? If you are not sure, grab the nested SQL and run the SQL by itself.
5. Why did they use Left Joins in this query instead of regular inner joins which we saw in prior queries?

General Definition Code Security SQL

```
SELECT a.actor_id,
   a.first_name,
   a.last_name,
   group_concat(DISTINCT (c.name::text || ': '::text) || (( SELECT group_concat(f.title::text) AS group_concat
          FROM film f
            JOIN film_category fc_1 ON f.film_id = fc_1.film_id
            JOIN film_actor fa_1 ON f.film_id = fa_1.film_id
         WHERE fc_1.category_id = c.category_id AND fa_1.actor_id = a.actor_id
         GROUP BY fa_1.actor_id))) AS film_info
  FROM actor a
    LEFT JOIN film_actor fa ON a.actor_id = fa.actor_id
    LEFT JOIN film_category fc ON fa.film_id = fc.film_id
    LEFT JOIN category c ON fc.category_id = c.category_id
 GROUP BY a.actor_id, a.first_name, a.last_name;
```

MSDE 631: Lab 4
Etana Disasa

<u>Answer</u>

3. The "`GROUP BY fa_1.actor_id`" clause groups the joined tables by the actor_id of the film_actor table.
4. The nested query is about the fourth column to be called which is "film_info". The first portion of the query tries to find the distinct "c.name" (category name) separated by ":". Then the nested query fetches all the films with the **same category** (with the same film_category.category_id and category.category_id) and with the **same actor** (film_actor.actor_id and actor.actor_id) and populates them all in the "film_info" column.
5. Left join still joins and captures entries in the table on the left, even though there are missing observations in the table to the right. In other words, there will be entries in the actor table that do not have corresponding entry in the film_actor table with a null value. When the happens, the joined table will still manage to join the tables. Therefore, there will not be rows from the table on the left, that will be left out in the merger just because there are not any corresponding values in columns in table to the right.

END OF PART TWO