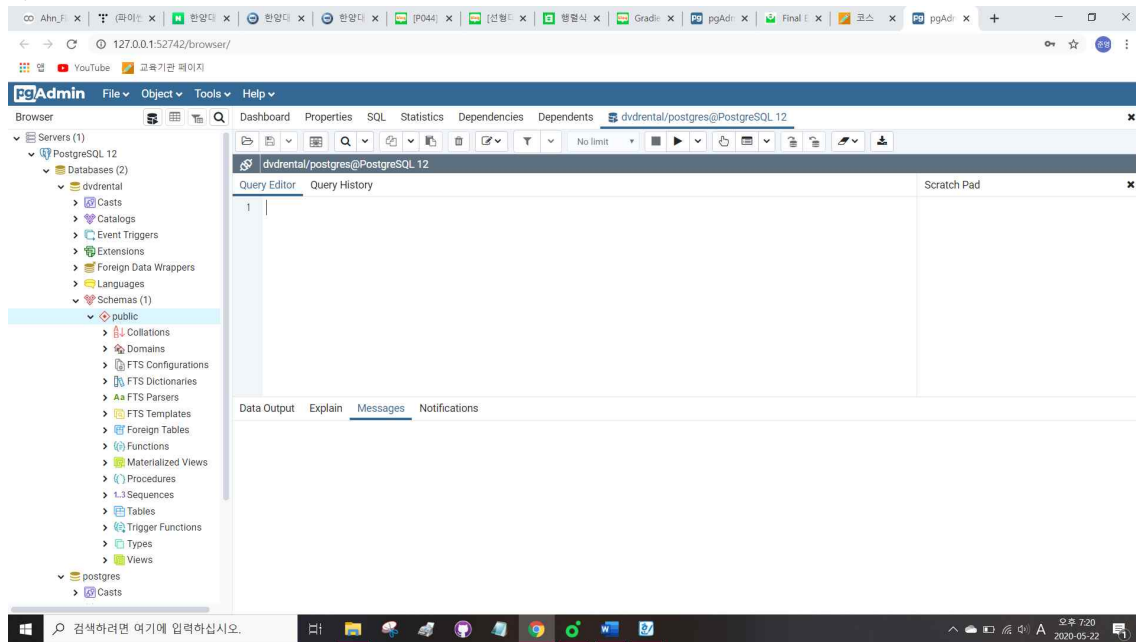
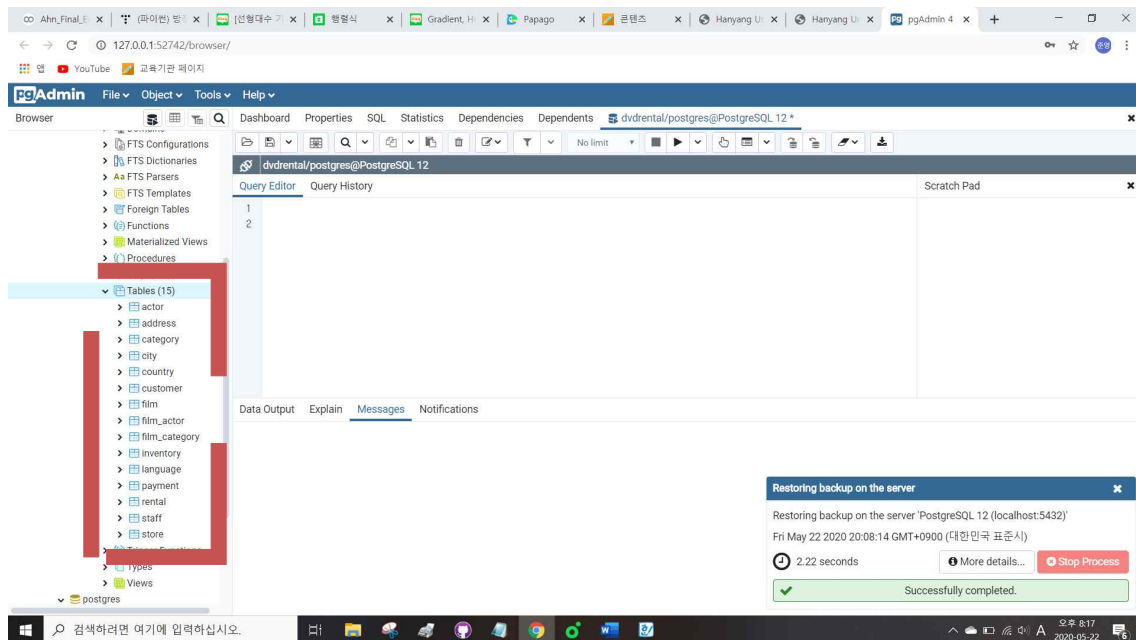


1)



2)

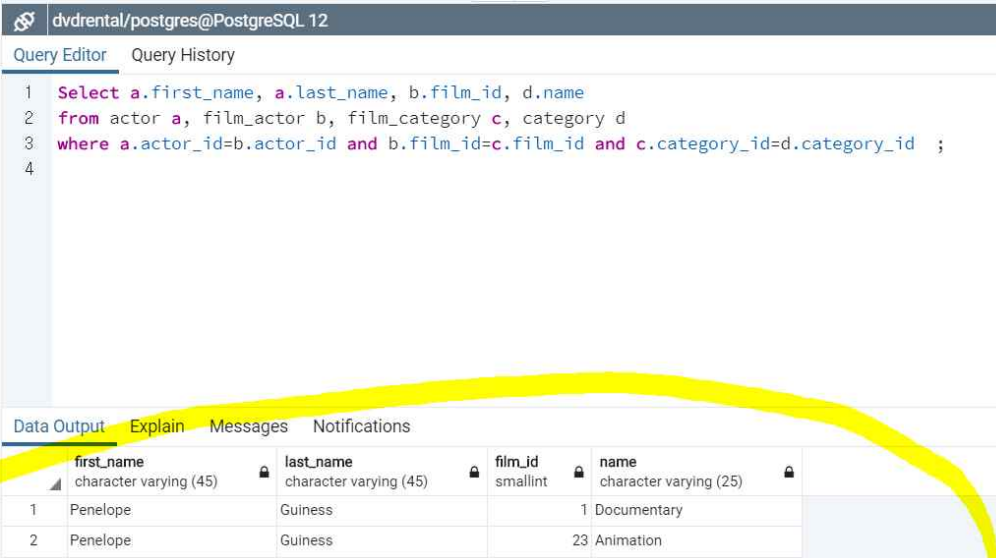


3)

1. 'actor' shows many actor's name divided by first name and last name
2. 'address' shows their residence address with subdivided informations such as district, city-ID, postal code and phone number.
3. 'category' represents all the genres.
4. 'city' shows city names and their country-ID, ordered by alphabet.
5. 'country' represents 109 countries' names, ordered by alphabet.
6. 'customer' includes customers' names (subdivided into first name and last name), their email addresses, address-ID, activebool and created-date, sorted by address\_ID.
7. 'film' suggests movie titles (ordered by alphabet), description, release year, language-ID, rental duration, rental rate, length(runtime) and replacement cost.
8. 'film\_actor' shows actor-IDs and what films each of them shoot.
9. 'film\_category' shows which category each film belongs to.
10. 'inventory' represents film-IDs and store-IDs of each film, sorted by film-ID.
11. 'language' includes some of the languages' name.
12. 'payment' shows customers' IDs and their payment-IDs, staff\_IDs, rental-IDs, amount and payment date.
13. 'rental' presents rental-IDs, rental date, inventory-IDs, customer-IDs, return date and staff-IDs.
14. 'staff' includes the informations about the staffs with their first name, last name, address\_ID, email, store-ID, active, username and password.
15. 'store' includes the informations about its store-ID, manager-ID and address-ID.

4)

There are 4 rows of output in total.



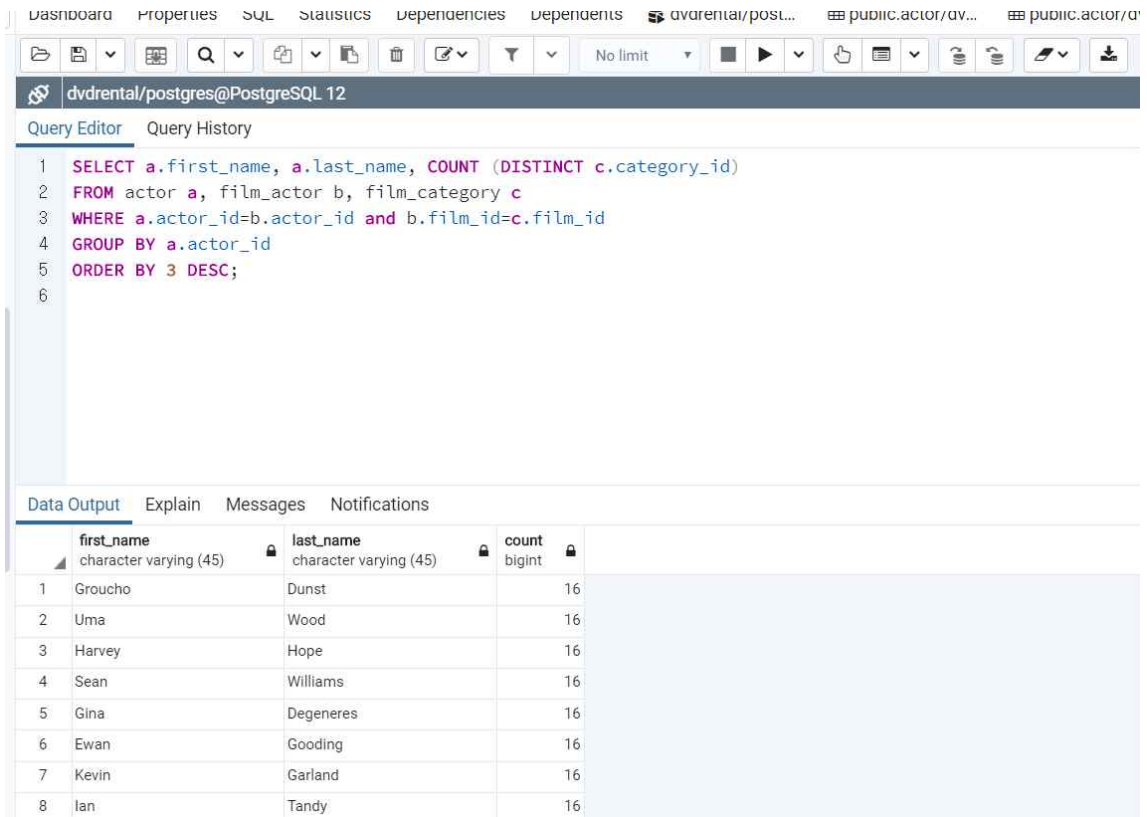
The screenshot shows a PostgreSQL query editor window titled 'dvdrental/postgres@PostgreSQL 12'. The 'Query Editor' tab is active, displaying the following SQL query:

```
1 Select a.first_name, a.last_name, b.film_id, d.name
2 from actor a, film_actor b, film_category c, category d
3 where a.actor_id=b.actor_id and b.film_id=c.film_id and c.category_id=d.category_id ;
4
```

The 'Data Output' tab is also visible, showing the results of the query. The results are displayed in a table with the following columns: first\_name, last\_name, film\_id, and name. The first two rows of data are highlighted with a yellow curved line.

	first_name character varying (45)	last_name character varying (45)	film_id smallint	name character varying (25)
1	Penelope	Guiness	1	Documentary
2	Penelope	Guiness	23	Animation

5)



The screenshot shows a PostgreSQL query editor interface. The query editor displays the following SQL query:

```
1 SELECT a.first_name, a.last_name, COUNT (DISTINCT c.category_id)
2 FROM actor a, film_actor b, film_category c
3 WHERE a.actor_id=b.actor_id and b.film_id=c.film_id
4 GROUP BY a.actor_id
5 ORDER BY 3 DESC;
6
```

Below the query editor, the 'Data Output' tab is selected, showing the results of the query in a table format:

	first_name character varying (45)	last_name character varying (45)	count bigint
1	Groucho	Dunst	16
2	Uma	Wood	16
3	Harvey	Hope	16
4	Sean	Williams	16
5	Gina	Degeneres	16
6	Ewan	Gooding	16
7	Kevin	Garland	16
8	Ian	Tandy	16

```
SELECT a.first_name, a.last_name, COUNT (DISTINCT c.category_id)
FROM actor a, film_actor b, film_category c
WHERE a.actor_id=b.actor_id and b.film_id=c.film_id
GROUP BY a.actor_id
ORDER BY 3 DESC;
```

*The highest number of film categories for an individual actor was '16' and total '11' actors had that number.*

dvdrental/postgres@PostgreSQL 12

Query Editor

Query History

```
1 SELECT AVG(categories) AS categories_avg
2 FROM
3 (SELECT a.first_name, a.last_name, COUNT (DISTINCT c.category_id) AS "categories"
4 FROM actor a, film_actor b, film_category c
5 WHERE a.actor_id=b.actor_id and b.film_id=c.film_id
6 GROUP BY a.actor_id
7 ORDER BY 3 DESC) AS z ;
```

Data Output

Explain

Messages

Notifications

categories\_avg  
numeric

1 13.0350000000000000

The Overall Average: 13.035  
 This number represents how many categories of movies on average actors performed in.

7)

The screenshot shows a PostgreSQL query editor interface. The query editor contains the following SQL query:

```
1 SELECT CORR(rental_rate, release_year) AS Correlation
2 FROM film;
3
```

The query is executed, and the results are displayed in the 'Data Output' tab. The results show a single row with the value [null] for the column 'Correlation'.

Correlation
[null]

The correlation : Null (not 0)

The value 'Null' means literally 'having no meaning' in computer language. The value range of correlation is from -1 to 1, where it means *less relevant* if the value get closer to 0. However because all the values of the column 'rental-year' are the same, we cannot calculate the correlation between 'rental rate' and 'release year'(not neither irrelevant or relevant, but just cannot judge)

8) Key is the attribute value used to check records or tuples, and the key used to identify a single record or two-plus is called the primary key. And a foreign key is a key in which a property or a set of attributes belonging to one relation becomes the primary key for another relation. In other words, the set of attributes that refer the primary key of another relation is a foreign key. Foreign keys are necessary to correctly express the relationship between relations.

- constraints 1) rental\_customer\_id\_fkey / This maps to 'customer' table
- constraints 2) rental\_inventory\_id\_fkey / This maps to 'inventory' table
- constraints 3) rental\_pkey / This does not map to certain table but is mapped by others.
- constraints 4) rental\_staff\_id\_key / This maps to 'staff' table

9)

dvdrental/postgres@PostgreSQL 12				
Query Editor   Query History				
<pre>1 SELECT a.actor_id, a.first_name, a.last_name, COUNT(r.rental_id) 2 FROM actor a, rental r, inventory i, film_actor fa 3 WHERE a.actor_id=fa.actor_id and r.inventory_id=i.inventory_id and i.film_id=fa.film_id 4 GROUP BY a.actor_id 5 ORDER BY COUNT(r.rental_id) DESC;</pre>				
Data Output   Explain   Messages   Notifications				
	actor_id [PK] Integer	first_name character varying (45)	last_name character varying (45)	count bigint
1	107	Gina	Degeneres	753
2	181	Matthew	Carrey	678
3	198	Mary	Keitel	674
4	144	Angela	Witherspoon	654
5	102	Walter	Torn	640
6	60	Henry	Berry	612

```
SELECT a.actor_id, a.first_name, a.last_name, COUNT(r.rental_id)
FROM actor a, rental r, inventory i, film_actor fa
WHERE a.actor_id=fa.actor_id and r.inventory_id=i.inventory_id and
i.film_id=fa.film_id
GROUP BY a.actor_id
ORDER BY COUNT(r.rental_id) DESC;
```

1. Gina Degeneres
2. Matthew Carrey
3. Mary Keitel
4. Angela Witherspoon
5. Walter Tom

10)

Query Editor | Query History

```
1 CREATE TABLE language_rentals AS
2 SELECT f.title, l.name, COUNT(f.film_id)
3 FROM film f, language l, rental r, inventory i
4 WHERE f.language_id=l.language_id and r.inventory_id=i.inventory_id and i.film_id=f.film_id
5 GROUP BY f.title, l.name ;
```

Data Output | Explain | Messages | Notifications

```
CREATE TABLE language_rentals AS
SELECT f.title, l.name, COUNT(f.film_id)
FROM film f, language l, rental r, inventory i
WHERE f.language_id=l.language_id and r.inventory_id=i.inventory_id and
i.film_id=f.film_id
GROUP BY f.title, l.name ;
```

public.language\_rentals/dvdrental/postgres@PostgreSQL 12

Query Editor | Query History

```
1 SELECT * FROM public.language_rentals
2
```

Data Output | Explain | Messages | Notifications

	title	name	count
	character varying (255)	character (20)	bigint
1	Reign Gentlemen	English	20
2	Blues Instinct	English	21
3	South Wait	English	22

There are **three** columns in my newly-created table.

11)

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

As a result, in general, the more bulky the table become, the more efficient to use indexes are. Therefore seemingly it looks better to utilize the concept of 'index' now because we have very bulky size of total 958 rows. However, all the values of the column 'name' of the table 'language\_rental' are equal to 'English' - in this case, *now, size doesn't matter*. So it is not only unnecessary but also even useless and inefficient to use indexes in this situation (we don't have to do numbering each molecule of milk so that differentiate it when drinking milk).

12)

Almost all the functions (or clauses) we've learned so far such as SELECT and CREATE are mainly focusing on creating something new (although there is exception such as data extraction which is fundamentally not creating new data), not on maintaining and modifying the table and any components of it. However, by using ALTER clause which is optimized for maintaining the table and others and modifying(altering) something into another, we can handle the real-time data in order to update our database every night properly. There are four main functions : ADD(used when adding new columns), DROP(used when deleting some columns), MODIFY(used when modifying data formats, default values and constraint conditions for 'NOT NULL' of previously existed columns), RENAME(used when renaming prior columns). By conjugating these techniques, we can process this again and again : Crawl new data about rentals, reflect them into our database by adding them or deleting something previously existed and refresh the status quo at a proper interval (it could be either 30minutes or 30hours).



13)

Scalability has three components : Load(How many requests can the system handle at once), Latency(The wait time for the request to be handled, not including network or other delays), Response Time(Wait time the end user sees, including network delays and queueing delays). The app #1 would be better than version #2 in terms of 'Load' because version #1 has more features that it is able to handle much more requests at once than version #2. However, there are 4 seconds gap between #1's response time (6.2 seconds) and #2's response time (2.2 seconds). We humans really care about how long it takes to get response especially when using electronic devices, which means 4 seconds are short time in itself but also intolerable time to be waited for in general. In addition, it is unclear that exactly how profitable having more features is while it is pretty obvious that how the increase of response time influences on revenue actually and directly(thanks to pilot focus groups who showed 1 second increase -> 7% user drop), which means choosing #2 is more predictable and safe way to get more revenue in terms of scalability. Also, in terms of maintainability (higher, the more user adoption), more complex systems are more fragile, more prone to break and less maintainable so I will advice my boss to choose version #2.

(I considered deployability except reliability, for it is hard to compare their reliabilities directly based on given information.)

14)

Well... I'm not sure because numeric figures of all the factors having influences on company's revenue are not given but I'll not change my answer (I judged **thoroughly in terms of deployability**). Let me consider whether the increase of the amount of times(A) each user used the app will have impact on revenue-increasing.

<1> A does not have to do with Load directly, because Load is about work-capacity at once but A is only about the time increase. Also A is about 'using time' but Latency and Response Time are about 'delay time' so there are no correlation between them.

<2> It is clear that A is not associated with Reliability (It doesn't make any

senses like the case <1>).

<3> A has no correlation with Maintainability - A does not make the system neither more complex nor more simple.

Thus, I thought the version #2 is still better than the version #1 because A has improved none of the factors of *deployability*.

15)

MapReduce (literally Map & Reduce methods) is a distribution programming model and also a parallel-processing technique which is designed to process large data more easily and that consists of this whole series of process ;

1. Input

Way too large file enters our mapreducing system. This file would be so big that it is tough and costly to fetch and process at once.

2. Splitting Input

In this step, Hadoop splits given file into proper-sized ones and store them in HDFS (Hadoop Distributed File System), being sure not to make the size of split too small so that the system could avoid adverse effects caused by overhead of split management and map-test generation. (higher split than 64MB is recommended in Hadoop)

3. Mapping

Receive the data and pass each data item to the mapper, and the mapper filters the input data and transforms it into (map it in) a (Key, Value) form that can be processed by the reducer.

4. Shuffling

In this step, the mapreduce system sorts, merges the mapped data, classifies data with the same key then carry them to the reducer.

5. Reducing

After shuffling, the system receives the result value as inputs from the mapper and process it. The reducers receive all the values and integrate to print them out.

## 6. Final Output

In this step, the output data of the reduce methods are aggregated and stored in the Hadoop file system.

Through these steps, MapReduce distributes tasks to several nodes, being able to process even previously untreatable data (due to size).

16)

A data lake is a system or repository of data stored in its natural(raw) format. A data lake is usually a single store of all enterprise data including raw copies of source system data and transformed data used for tasks such as reporting, visualization, advanced analytics and machine learning. A data lake can include structured data from relational databases (rows and columns), semi-structured data (CSV, XML), unstructured data (documents, PDFs) and binary data (images, audio, video). So 'organizing' (which does not mean 'no censoring or manipulation') is hardly processed when in-putting raw/not-raw data unlike a Data Warehouse.

However, if we throw all the data into a data lake without any censoring, our lake could become a Data Swamp where there are so many deteriorated and unmanaged data that it becomes either inaccessible to its intended users or little-value provider. As we learned, **it is like throwing everything into a black hole and hoping 'magic' comes out**. So I'mma go to my boss and say like this :

*Please no, boss. Can we row in the lake full of debris? Can we go forward? Can you escape from the sand dummy with all of your body buried? Likewise, If you throw all the datasets about customers you've bought into our data lake, our data will be unable to row ; unable to go forward ; unable to come out.*

17) ETL is the terminology related to data movement and transformation procedures.

**Extraction(E)** is about obtaining data from one or more data sources.

**Transformation(T)** is about cleansing data, transforming format, standardizing, integrating, or applying specific rules embedded in multiple applications.

**Loading(L)** is about loading data that has been processed with the above transformation steps into a specific target system.

And Data Cleaning (Data Cleansing) is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Without these two processes, user entry errors, corruption in transmission or storage, or different data dictionary definitions of similar entities in different stores will easily lead to the inconsistencies, which means that a data set would still remain incoherent and inconsistent with other similar data sets in the system. So most of the result from the data without the two processes would be worthless and out of original purpose.