TASK 3.6

Summarizing & Cleaning Data in SQL

1:
DUPLICATE

Rockbuster/postgres@PostgreSQL 16

```
SELECT film_id, title, description, release_year, language_id, rental_rate, rental_duration,
length, replacement_cost, rating, last_update, special_features, fulltext,
COUNT (*)
from film
Group By film_id, title, description, release_year, language_id, rental_rate, rental_duration
length, replacement_cost, rating, last_update, special_features, fulltext
HAVING COUNT (*)>1;
```

Data Output

film_id [PK] integer	title character varying (255)	description text	release_year integer	language_id smallint	rental_rate numeric (4,2)	rental_duration smallint	length smallint	replacement_cost numeric (5,2)	rating mpaa_rating	last tim

Rockbuster/postgres@PostgreSQL 16

```
SELECT customer_id, store_id, first_name,last_name, email, address_id, activebool, create_date
COUNT(*)
FROM customer
Group By customer_id, store_id, first_name,last_name, email, address_id, activebool, create_da
HAVING COUNT (*)>1;
```

Data Output

customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone

I didn't come across any duplicate entries in the film_table or customer_table. But if there were any, I would address them by either removing the duplicate entry or creating a "view table" that selects only one instance of the duplicates.

MISSING VALUES

Rockbuster/postgres@PostgreSQL 16

No limit

Query | Query History | Scratch Pad

```
SELECT *
FROM customer
Where (customer_id,store_id, first_name,last_name, email, address_id, activebool, create_date
is NULL;
```

Data Output | Messages | Notifications

customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone

Rockbuster/postgres@PostgreSQL 16

No limit

Query | Query History | Scratch Pad

```
SELECT *
FROM film
Where (film_id,title, description, release_year,
       language_id, rental_rate, rental_duration, length,
       replacement_cost, last_update, rating, special_features, fulltext)
       IS NULL;
```

Data Output | Messages | Notifications

film_id [PK] integer	title character varying (255)	description text	release_year integer	language_id smallint	rental_duration smallint	rental_rate numeric (4,2)	length smallint	replacement_cost numeric (5,2)	rating mpaa_rating	last tim

I haven't encountered any missing values in the tables so far. However, if any were present, I would fill them in as needed. For instance, if the last_name field in the customer_table was blank, I would skip it. But if the release_year of a film was missing, I would make sure to find the information and populate the record accordingly.

2:

NUMERICAL

Rockbuster/postgres@PostgreSQL 16

No limit

Execute script F5

Query | Query History | Scratch Pad

```
select min(release_year) AS min_re
       min (rental_duration) AS min_rental_duration,
       min (rental_rate) As min_rental_rate,
       min (length) AS min_length,
       min (replacement_cost) as min_replacement_cost,
       max (release_year) AS max_release_year,
       max (rental_duration) AS max_rental_duration,
       max(rental_rate) As max_rental_rate,
       max(length) AS max_length,
       max (replacement_cost) as max_replacement_cost,
       avg (release_year) AS avg_release_year,
       avg (rental_duration) AS avg_rental_duration,
       avg (rental_rate) As avg_rental_rate,
       avg (length) AS max_length,
       avg (replacement_cost) as avg_replacement_cost
       FROM film
```

Data Output | Messages | Notifications

	min_release_year integer	min_rental_duration smallint	min_rental_rate numeric	min_length smallint	min_replacement_cost numeric	max_release_year integer	max_rental_duration smallint	max_rental_rate numeric	max_length smallint
1	2006	3	0.99	46	9.99	2006	7	4.99	18

These data records offer crucial insights into several aspects:

- The overall satisfaction level of customers with the rented movies.
- The average duration customers typically keep a rented movie.
- The potential cost associated with replacing a specific movie.
- Additionally, analyzing the popularity of different movie lengths.

In the Customer_table, there isn't pertinent numerical data for retrieving minimum, maximum, and average values. For instance, customer_id, being a unique identifier for each customer, wouldn't yield meaningful insights through minimum, maximum, or average calculations.

NON NUMERICAL

Rockbuster/postgres@PostgreSQL 16

No limit

Query | Query History

```
SELECT MODE () WITHIN GROUP (ORDER BY language_id) AS longest,
       MODE () WITHIN GROUP (ORDER BY rating) AS most_rated
from film
```

Data Output | Messages | Notifications

	longest smallint	most_rated mpaa_rating
1	1	PG-13

In this analysis, it's feasible to determine the most prevalent language among the movie collection and identify which audience demographic could derive the most value from the current selection. In my perspective, querying data from text fields like description wouldn't be pertinent as the information contained within those columns wouldn't offer substantial insights.

CUSTOMER TABLE

Rockbuster/postgres@PostgreSQL 16

No limit

Query | Query History

```
SELECT MODE () WITHIN GROUP (ORDER BY store_id) AS most_store,
       MODE () WITHIN GROUP (ORDER BY activebool) AS most_activebool
from customer
```

Data Output | Messages | Notifications

	most_store smallint	most_activebool boolean
1	1	true

In this case, the analysis shows that Store 1 has the most customers signed up. It also asks if there are more active customers than inactive ones. But checking things like first names or last names wouldn't really help find useful information, like the most common names in the database.

3:
SQL is really handy because it gives you exactly what you ask for when you make a query. It's great for connecting data from different places and makes sure your data stays safe from accidental changes. But since I'm still learning, I sometimes find it hard to use the right words. With practice, though, I think I'll get better at it.
Excel is good too because it helps automate tasks with lots of functions. I'm more used to Excel, so I can do similar things by moving tables around and using functions. But when it comes to big amounts of data, Excel can struggle a bit compared to SQL. Even though Excel is easier to use and understand, I'm enjoying learning SQL and getting better at it. Just like learning any new language, it takes time to get the hang of it.