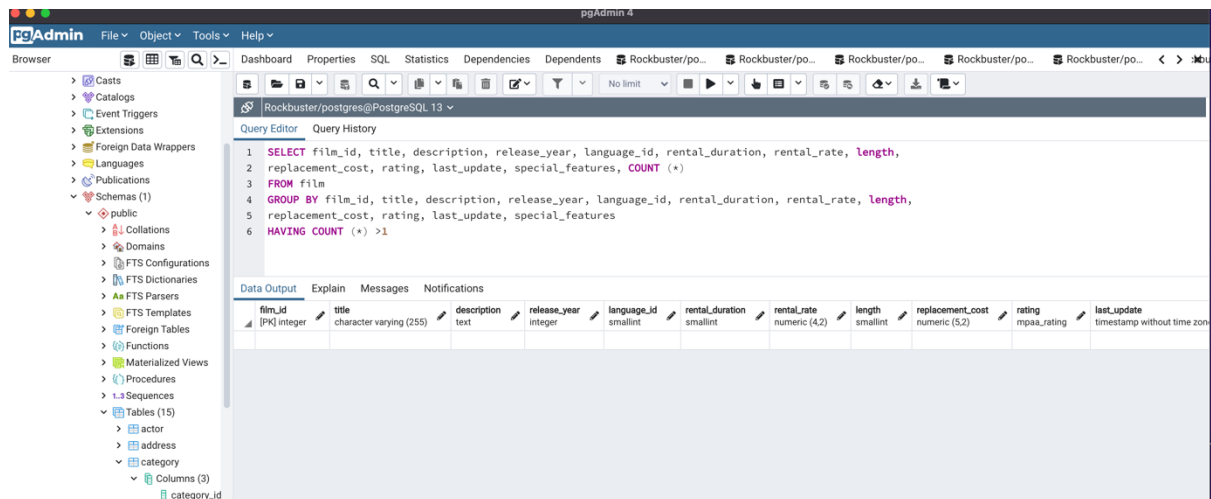


## TASK\_6: Summarizing & Cleaning Data in SQL

### 1. Check for and clean dirty data:

#### Duplicate Data from Film



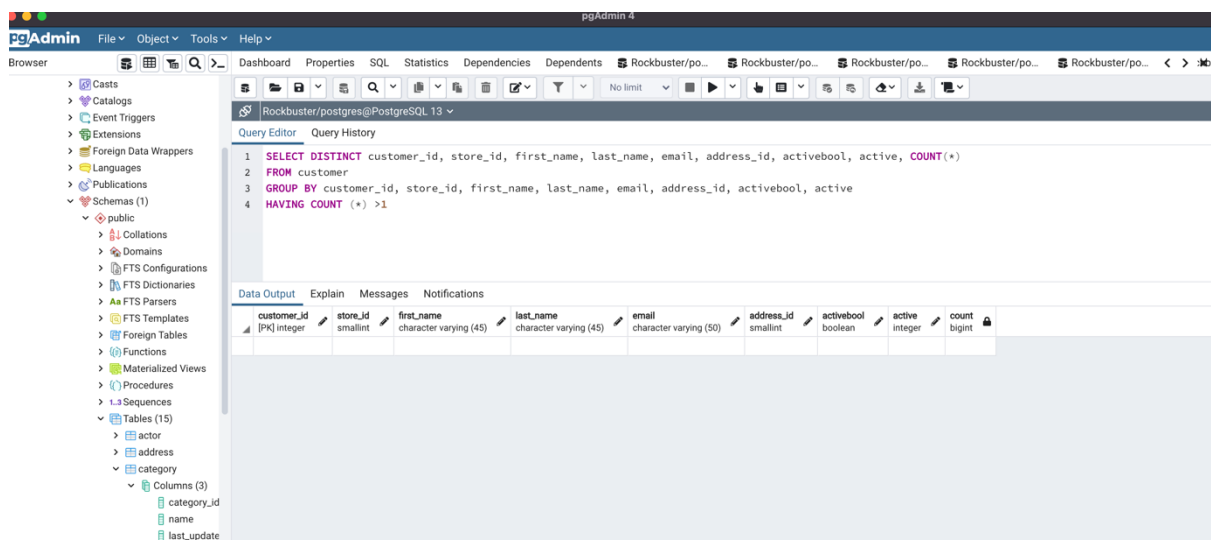
The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema and the 'film' table. The central Query Editor contains the following SQL query:

```
1 SELECT film_id, title, description, release_year, language_id, rental_duration, rental_rate, length,  
2 replacement_cost, rating, last_update, special_features, COUNT (*)  
3 FROM film  
4 GROUP BY film_id, title, description, release_year, language_id, rental_duration, rental_rate, length,  
5 replacement_cost, rating, last_update, special_features  
6 HAVING COUNT (*) > 1
```

Below the query editor, the 'Data Output' tab is active, showing the column headers for the query results:

film_id	title	description	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating	last_update
---------	-------	-------------	--------------	-------------	-----------------	-------------	--------	------------------	--------	-------------

#### Duplicate Data from Customer



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema and the 'customer' table. The central Query Editor contains the following SQL query:

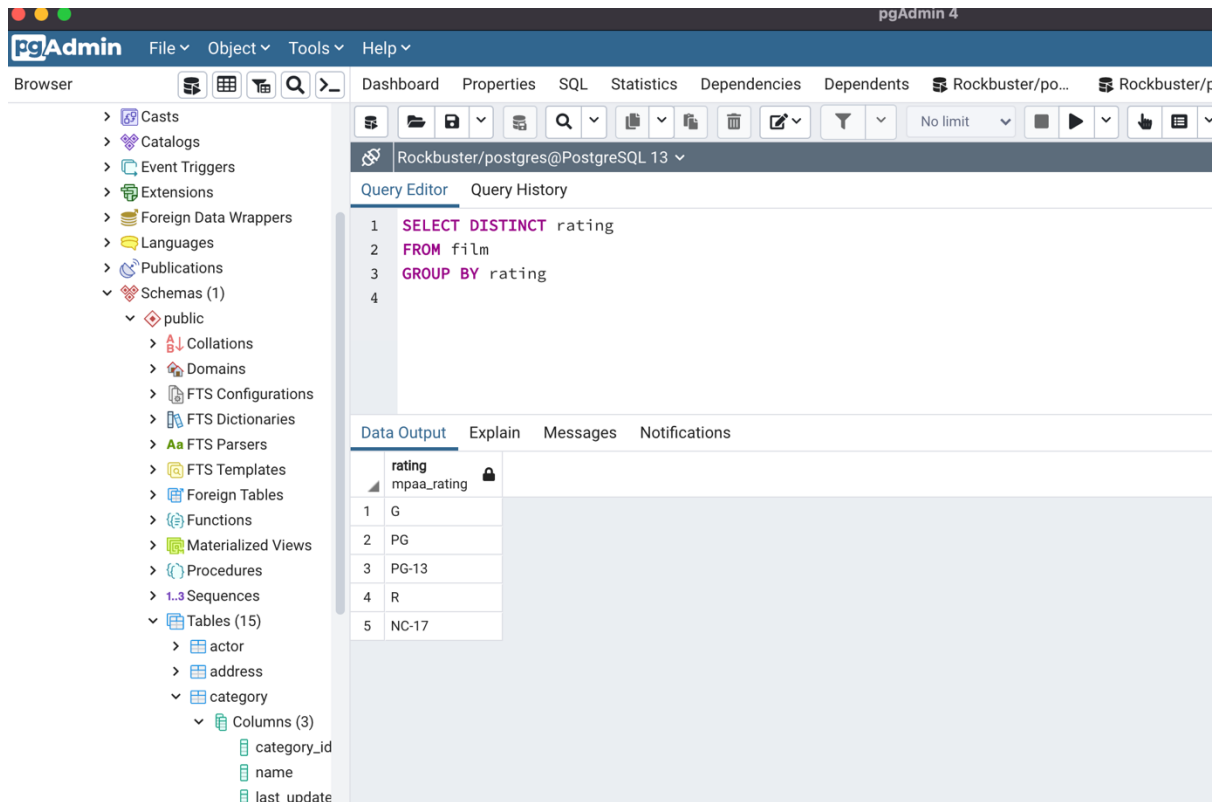
```
1 SELECT DISTINCT customer_id, store_id, first_name, last_name, email, address_id, activebool, active, COUNT(*)  
2 FROM customer  
3 GROUP BY customer_id, store_id, first_name, last_name, email, address_id, activebool, active  
4 HAVING COUNT (*) > 1
```

Below the query editor, the 'Data Output' tab is active, showing the column headers for the query results:

customer_id	store_id	first_name	last_name	email	address_id	activebool	active	count
-------------	----------	------------	-----------	-------	------------	------------	--------	-------

In both cases there is no duplicate data.

## Non-Uniform Data



The screenshot shows the pgAdmin 4 web interface. On the left is a 'Browser' pane with a tree view of database objects. The 'public' schema is expanded, showing various objects like Collations, Domains, FTS Configurations, etc. The main area is the 'Query Editor' for the 'Rockbuster/postgres@PostgreSQL 13' connection. It contains a SQL query: `SELECT DISTINCT rating FROM film GROUP BY rating`. Below the query editor, the 'Data Output' tab is active, displaying a table with 5 rows and 1 column named 'rating'. The rows contain the values 'G', 'PG', 'PG-13', 'R', and 'NC-17'. The table has a header row with 'rating' and a sub-header 'mpaa\_rating'.

rating
mpaa_rating
G
PG
PG-13
R
NC-17

Based on our results, there is also none of non-uniform values. If there would be, we can use the UPDATE command in combination with SET and WHERE.

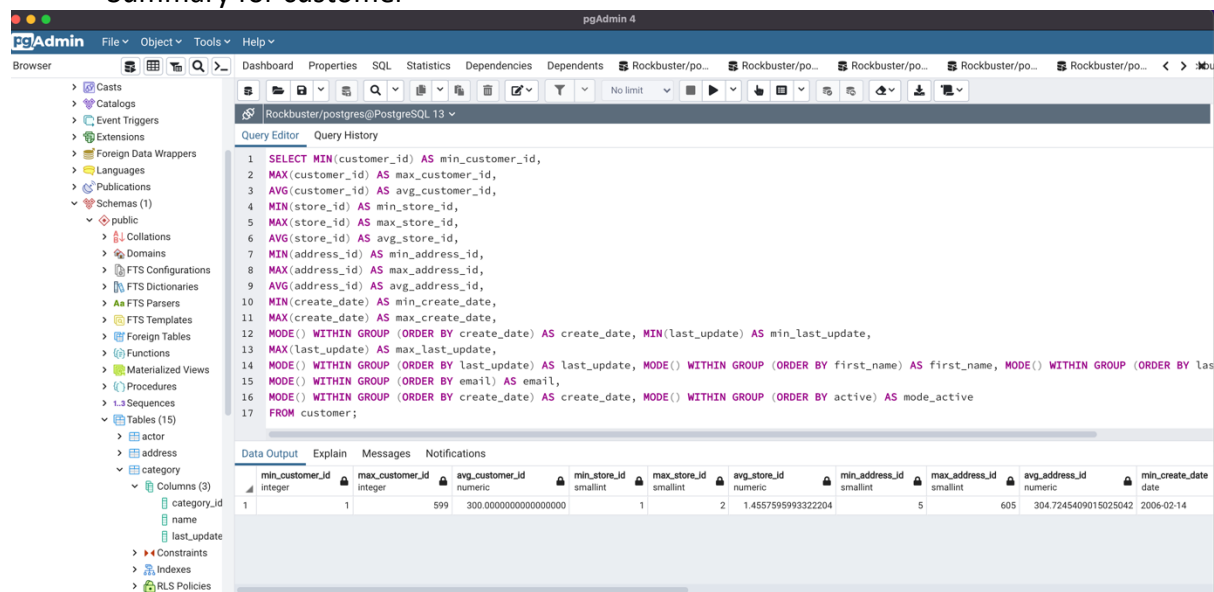
For Incorrect data, I would discuss with my team to find a solution, because in this case instead of writing commands, logical and critical thinking is more important.

To find missing information, firstly I would visualize the data. After detecting the missing value, we might replace them. As example:

```
UPDATE tablename  
SET = AVG(coll)  
WHERE coll IS NULL
```

## 2. Summarize your data

### Summary for customer



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'public' schema and the 'customer' table. The main window shows a SQL query in the Query Editor, which is a SELECT statement summarizing various attributes of the 'customer' table. Below the query editor, the 'Data Output' tab is active, displaying the results of the query in a table format. The table has 11 columns, each corresponding to a specific attribute of the customer table, and one row of data.

	min_customer_id	max_customer_id	avg_customer_id	min_store_id	max_store_id	avg_store_id	min_address_id	max_address_id	avg_address_id	min_create_date
1	1	599	300.0000000000000000	1	2	1.4557595993322204	5	605	304.7245409015025042	2006-02-14

--descriptive statistics for customer table

```
SELECT MIN(customer_id) AS min_customer_id, MAX(customer_id) AS  
max_customer_id,  
AVG(customer_id) AS avg_customer_id,  
MIN(store_id) AS min_store_id,  
MAX(store_id) AS max_store_id,  
AVG(store_id) AS avg_store_id,  
MIN(address_id) AS min_address_id,  
MAX(address_id) AS max_address_id,  
AVG(address_id) AS avg_address_id,  
MIN(create_date) AS min_create_date,  
MAX(create_date) AS max_create_date,  
MODE() WITHIN GROUP (ORDER BY create_date) AS create_date, MIN(last_update) AS  
min_last_update, MAX(last_update) AS max_last_update,  
MODE() WITHIN GROUP (ORDER BY last_update) AS last_update, MODE() WITHIN  
GROUP (ORDER BY first_name) AS first_name, MODE() WITHIN GROUP (ORDER BY  
last_name) AS last_name, MODE() WITHIN GROUP (ORDER BY email) AS email,  
MODE() WITHIN GROUP (ORDER BY create_date) AS create_date, MODE() WITHIN  
GROUP (ORDER BY active) AS mode_active
```

```
FROM customer;
```

## Summary for Film

pgAdmin

FileObjectToolsHelp

DashboardPropertiesSQLStatisticsDependenciesDependentsRockbuster/postgres@PostgreSQL 13

Browser

Cast

Catalogs

Event Triggers

Extensions

Foreign Data Wrappers

Languages

Publications

Schemas (1)

public

Collations

Domains

FTS Configurations

FTS Dictionaries

FTS Parsers

FTS Templates

Foreign Tables

Functions

Materialized Views

Procedures

Sequences

Tables (15)

actor

address

category

Columns (3)

category\_id

name

last\_update

Constraints

Query Editor

Query History

```
1 SELECT MIN(rental_rate) AS min_rental_rate, MAX(rental_rate) AS max_rental_rate, AVG(rental_rate) AS avg_rental_rate,
2 MIN(rental_duration) AS min_rental_duration, MAX(rental_duration) AS max_rental_duration, AVG(rental_duration) AS avg_rental_duration,
3 MIN(film_id) AS min_film,
4 MAX(film_id) AS max_film,
5 AVG(film_id) AS avg_film,
6 MIN(language_id) AS min_language, MAX(language_id) AS max_language, AVG(language_id) AS avg_language, MIN(length) AS min_length,
7 MAX(length) AS max_length,
8 AVG(length) AS avg_length,
9 MIN(replacement_cost) AS min_replacement_cost,
10 MAX(replacement_cost) AS max_replacement_cost, AVG(replacement_cost) AS avg_replacement_cost,
11 MODE() WITHIN GROUP (ORDER BY rating) AS rating_value,
12 MODE() WITHIN GROUP (ORDER BY special_features) AS feature_value, MODE() WITHIN GROUP (ORDER BY release_year) AS release_year,
13 MODE() WITHIN GROUP (ORDER BY title) AS title_value,
14 MODE() WITHIN GROUP (ORDER BY fulltext) AS fulltext
15 FROM film
16
```

Data Output

Explain

Messages

Notifications

	min_rental_rate	max_rental_rate	avg_rental_rate	min_rental_duration	max_rental_duration	avg_rental_duration	min_film	max_film	avg_film	min_language	max_language	avg_language	min_length	max_length	avg_length	rating_value	feature_value	release_year	title_value	fulltext
	numeric	numeric	numeric	smallint	smallint	numeric	integer	integer	numeric	smallint	smallint	numeric	smallint	smallint	numeric					
1	0.99	4.99	2.9800000000000000	3	7	4.9850000000000000	1	1000	500.5000000000000000	1	1	1	1	1	1	1	1	1	1	

--descriptive statistics for film table

```
SELECT MIN(rental_rate) AS min_rental_rate, MAX(rental_rate) AS max_rental_rate,
AVG(rental_rate) AS avg_rental_rate, MIN(rental_duration) AS min_rental_duration,
MAX(rental_duration) AS max_rental_duration, AVG(rental_duration) AS
avg_rental_duration, MIN(film_id) AS min_film,
MAX(film_id) AS max_film,
AVG(film_id) AS avg_film,
MIN(language_id) AS min_language, MAX(language_id) AS max_language,
AVG(language_id) AS avg_language, MIN(length) AS min_length, MAX(length) AS
max_length,
AVG(length) AS avg_length,
MIN(replacement_cost) AS min_replacement_cost,
MAX(replacement_cost) AS max_replacement_cost, AVG(replacement_cost) AS
avg_replacement_cost,
MODE() WITHIN GROUP (ORDER BY rating) AS rating_value,
MODE() WITHIN GROUP (ORDER BY special_features) AS feature_value, MODE()
WITHIN GROUP (ORDER BY release_year) AS release_year, MODE() WITHIN GROUP
(ORDER BY title) AS title_value,
MODE() WITHIN GROUP (ORDER BY fulltext) AS fulltext
FROM film
```

3. Excel is useful for quick visualizations and summaries of data, whereas SQL is necessary for working with large volumes of data, managing databases, and using relational databases to their full potential. Excel gets slower the more data you ask it to handle. Excel cannot store more than one million lines of data. SQL is fast and can handle large loads of data. Unlike Excel, SQL can handle well over one million fields of data with ease. SQL queries are also more flexible and powerful.