Task 6. Clean and summarize data

Answers 3.6

Contents

1. Check for and clean dirty data:

Find out if the **film table** and the **customer table** contain any dirty data, specifically **non-uniform or duplicate data, or missing values**. Next to each query write 2 to 3 sentences explaining how you would clean the data (even if the data is not dirty).

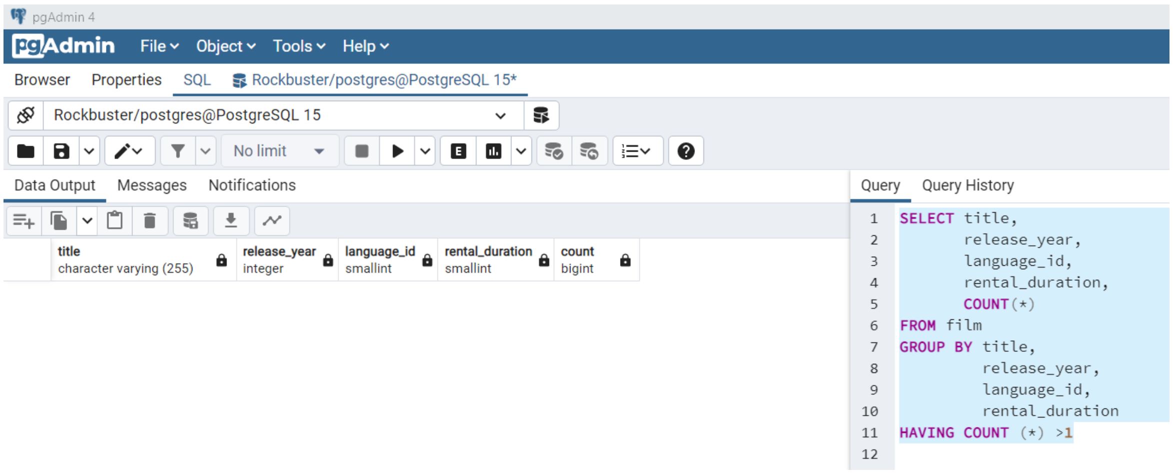
Film table

Looking for Duplicates:

```
SELECT title,
       release_year,
       language_id,
       rental_duration,
       COUNT(*)
FROM film
GROUP BY title,
         release_year,
         language_id,
         rental_duration
```

HAVING COUNT (*) >1

Printscreen:


pgAdmin 4
File Object Tools Help
Browser Properties SQL Rockbuster/postgres@PostgreSQL 15*
Rockbuster/postgres@PostgreSQL 15
No limit
Data Output Messages Notifications
Query Query History

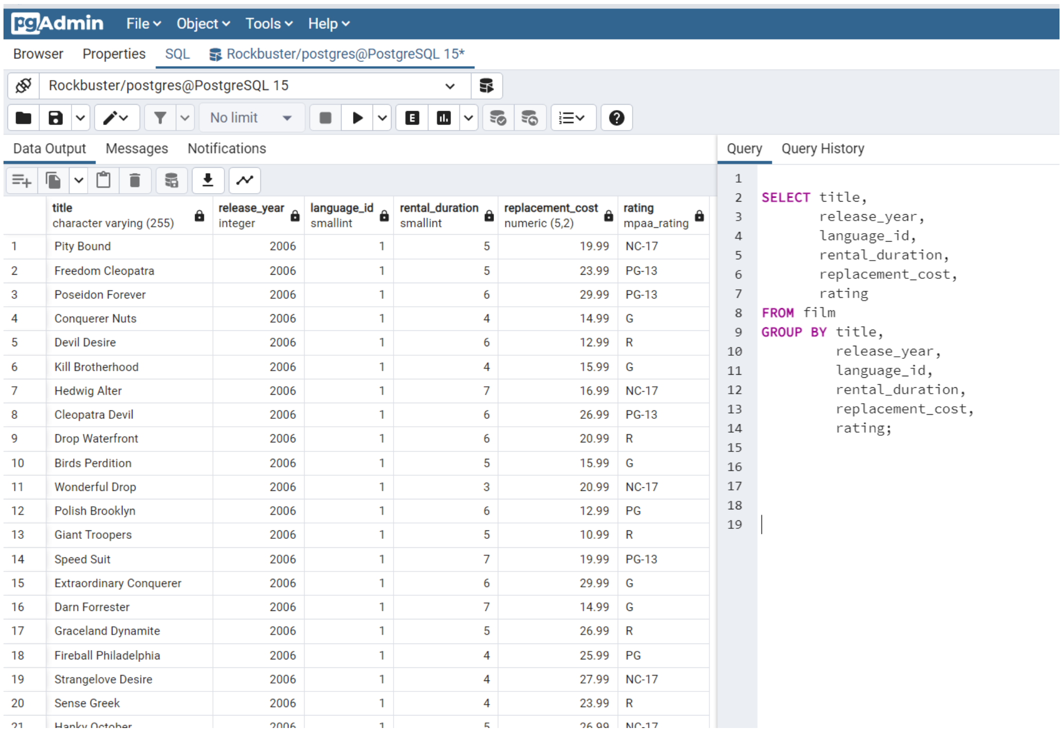
title character varying (255)	release_year integer	language_id smallint	rental_duration smallint	count bigint

```
SELECT title,
       release_year,
       language_id,
       rental_duration,
       COUNT(*)
FROM film
GROUP BY title,
         release_year,
         language_id,
         rental_duration
HAVING COUNT (*) >1
```

There are no duplicates returned in the film table. If there were any, I would keep them but use other queries, such as GROUP BY or DISTINCT so unique records could be analysed.

Looking for missing values and non-uniform data:

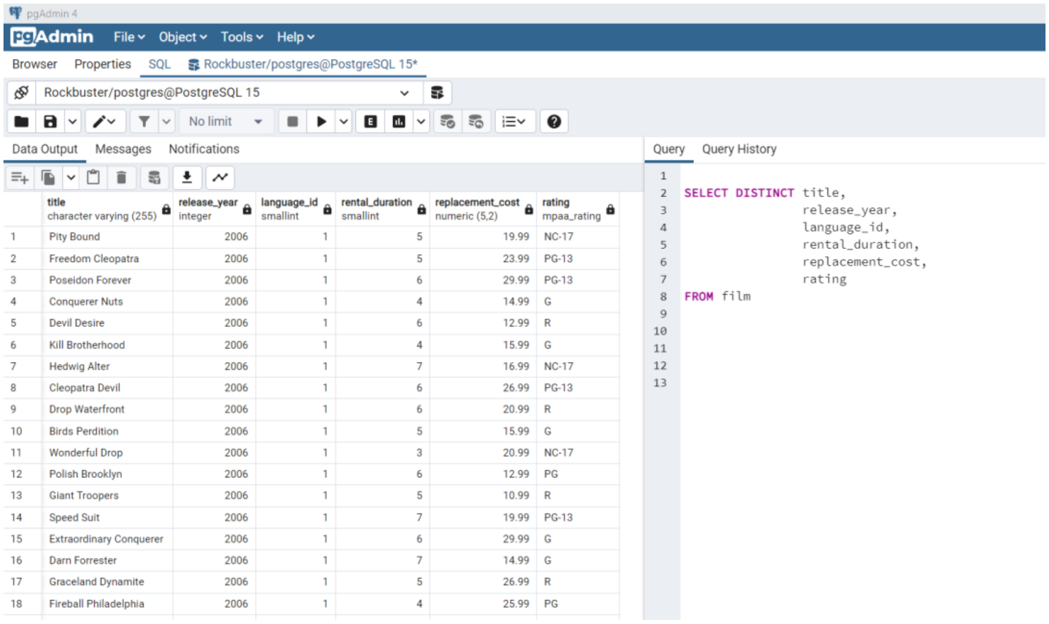
```
SELECT title,
       release_year,
       language_id,
       rental_duration,
           replacement_cost,
           rating
FROM film
GROUP BY title,
         release_year,
         language_id,
         rental_duration,
                replacement_cost,
                rating;
```



```sql
SELECT title,
       release_year,
       language_id,
       rental_duration,
       replacement_cost,
       rating
FROM film
GROUP BY title,
         release_year,
         language_id,
         rental_duration,
         replacement_cost,
         rating;
```

	title character varying (255)	release_year integer	language_id smallint	rental_duration smallint	replacement_cost numeric (5,2)	rating mpaa_rating
1	Pity Bound	2006	1	5	19.99	NC-17
2	Freedom Cleopatra	2006	1	5	23.99	PG-13
3	Poseidon Forever	2006	1	6	29.99	PG-13
4	Conquerer Nuts	2006	1	4	14.99	G
5	Devil Desire	2006	1	6	12.99	R
6	Kill Brotherhood	2006	1	4	15.99	G
7	Hedwig Alter	2006	1	7	16.99	NC-17
8	Cleopatra Devil	2006	1	6	26.99	PG-13
9	Drop Waterfront	2006	1	6	20.99	R
10	Birds Perdition	2006	1	5	15.99	G
11	Wonderful Drop	2006	1	3	20.99	NC-17
12	Polish Brooklyn	2006	1	6	12.99	PG
13	Giant Troopers	2006	1	5	10.99	R
14	Speed Suit	2006	1	7	19.99	PG-13
15	Extraordinary Conquerer	2006	1	6	29.99	G
16	Darn Forrester	2006	1	7	14.99	G
17	Graceland Dynamite	2006	1	5	26.99	R
18	Fireball Philadelphia	2006	1	4	25.99	PG
19	Strangelove Desire	2006	1	4	27.99	NC-17
20	Sense Greek	2006	1	4	23.99	R
21	Hanky October	2006	1	5	26.99	NC-17

Or using the distinct statement instead:



```sql
SELECT DISTINCT title,
                release_year,
                language_id,
                rental_duration,
                replacement_cost,
                rating
FROM film
```

	title character varying (255)	release_year integer	language_id smallint	rental_duration smallint	replacement_cost numeric (5,2)	rating mpaa_rating
1	Pity Bound	2006	1	5	19.99	NC-17
2	Freedom Cleopatra	2006	1	5	23.99	PG-13
3	Poseidon Forever	2006	1	6	29.99	PG-13
4	Conquerer Nuts	2006	1	4	14.99	G
5	Devil Desire	2006	1	6	12.99	R
6	Kill Brotherhood	2006	1	4	15.99	G
7	Hedwig Alter	2006	1	7	16.99	NC-17
8	Cleopatra Devil	2006	1	6	26.99	PG-13
9	Drop Waterfront	2006	1	6	20.99	R
10	Birds Perdition	2006	1	5	15.99	G
11	Wonderful Drop	2006	1	3	20.99	NC-17
12	Polish Brooklyn	2006	1	6	12.99	PG
13	Giant Troopers	2006	1	5	10.99	R
14	Speed Suit	2006	1	7	19.99	PG-13
15	Extraordinary Conquerer	2006	1	6	29.99	G
16	Darn Forrester	2006	1	7	14.99	G
17	Graceland Dynamite	2006	1	5	26.99	R
18	Fireball Philadelphia	2006	1	4	25.99	PG

There are no non-uniform and missing values. Also, the data type has been set for all the columns, using constraints, which prevents the entry of non-uniform data.

If they existed, to fix non-uniform values we could use the UPDATE statement to update the values, like this:

UPDATE table_name

SET column_name = ‘here list correct value’

WHERE column_name IN (…, here list the non-uniform values that we found)

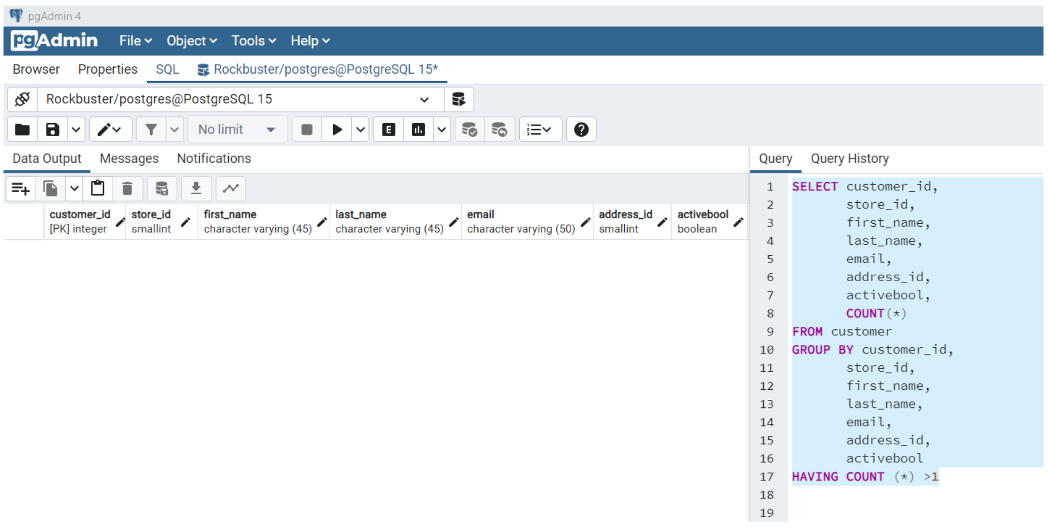
If there were missing values, we could either ignore the column if there were a lot of values missing or we could impute values, if there were only a few values missing. For instance, if the value is numeric, we could input the mean.

Customer table

Looking for duplicates

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

Screenshot:



Rockbuster/postgres@PostgreSQL 15

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

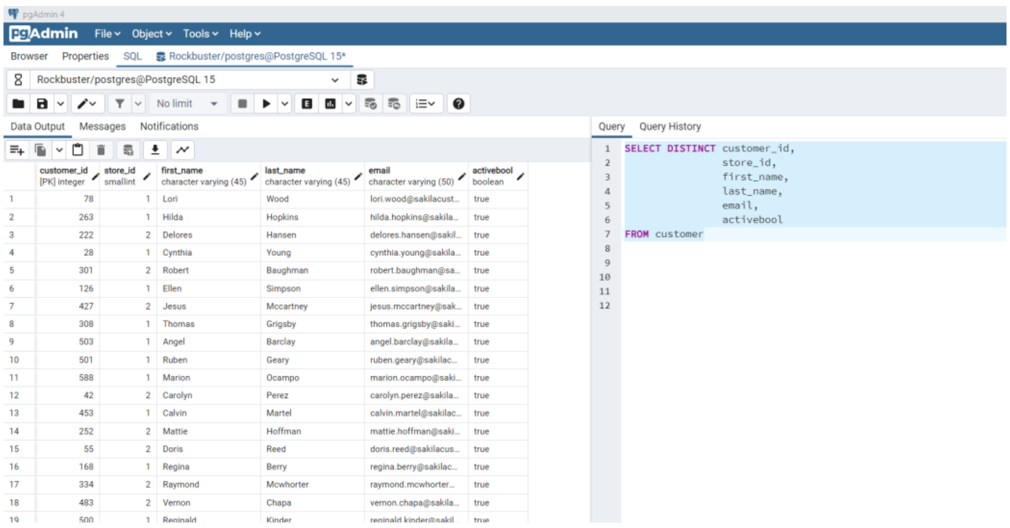
```
SELECT customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       activebool,
       COUNT(*)
FROM customer
GROUP BY customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       activebool
HAVING COUNT (*)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

There are also no duplicate values. If there were, they should be handled very carefully, yet because the table includes unique ID values, it would be possible to use a delete statement to remove the duplicate value. For instance:

```
DELETE
FROM costumer
WHERE customer_id NOT IN
(SELECT MIN (customer_id)
FROM customer
GROUP BY customer_id,
                    store_id)
```

Look for non-uniform data and missing values:

```
SELECT DISTINCT customer_id,
                store_id,
                first_name,
                last_name,
                        email,
               activebool
FROM customer
```



Rockbuster/postgres@PostgreSQL 15

```
SELECT DISTINCT customer_id,
                store_id,
                first_name,
                last_name,
                email,
                activebool
FROM customer
```

	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	activebool boolean
1	78	1	Lori	Wood	lori.wood@sakilacust...	true
2	263	1	Hilda	Hopkins	hilda.hopkins@sakila...	true
3	222	2	Delores	Hansen	delores.hansen@sakil...	true
4	28	1	Cynthia	Young	cynthia.young@sakila...	true
5	301	2	Robert	Baughman	robert.baughman@sa...	true
6	126	1	Ellen	Simpson	ellen.simpson@sakila...	true
7	427	2	Jesus	Mccartney	jesus.mccartney@sak...	true
8	308	1	Thomas	Grigsby	thomas.grigsby@saki...	true
9	503	1	Angel	Barclay	angel.barclay@sakila...	true
10	501	1	Ruben	Geary	ruben.geary@sakilac...	true
11	588	1	Marion	Ocampo	marion.ocampo@saki...	true
12	42	2	Carolyn	Perez	carolyn.perez@sakila...	true
13	453	1	Calvin	Martel	calvin.martel@sakilac...	true
14	252	2	Mattie	Hoffman	mattie.hoffman@saki...	true
15	55	2	Doris	Reed	doris.reed@sakilacus...	true
16	168	1	Regina	Berry	regina.berry@sakilac...	true
17	334	2	Raymond	Mcwhorter	raymond.mcwhorter...	true
18	483	2	Vernon	Chapa	vernon.chapa@sakila...	true
19	500	1	Reginald	Kinder	reginald.kinder@sakil...	true

There were also no missing values and non-uniform data in the customer table. If there were, the solutions would be similar those in the film table. We could update data, to ensure all values were uniform, assuming we knew the right form of those values. We would either ignore columns with missing values if these were extensive, or input values to columns, if there were only a few missing values.

2. Summarize your data:

Use SQL to calculate **descriptive statistics** for both the **film table and the customer table**. For numerical columns, this means finding the minimum, maximum, and average values. For non-numerical columns, calculate the mode value. Copy-paste your SQL queries and their outputs into your answers document.

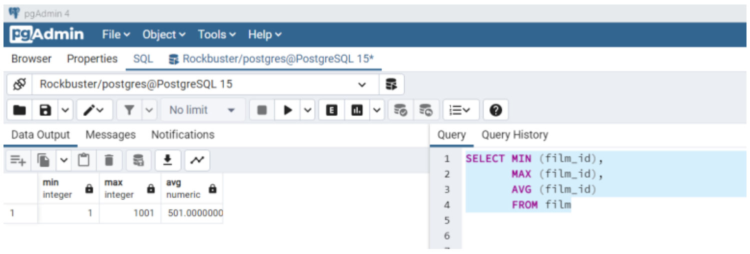
Descriptive statistics of film table and customer table

Film Table

Numeric values: film_id, release_year, language_id, rental_duration, rental_rate, length, replacement_cost

film_id:

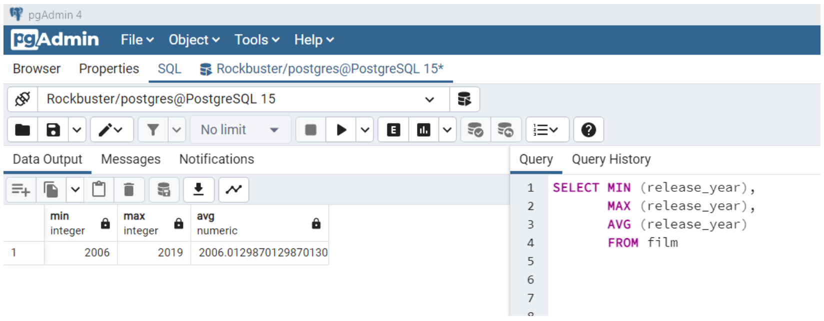
```
SELECT MIN (film_id),
       MAX (film_id),
           AVG (film_id)
           FROM film
```



```
SELECT MIN (film_id),
       MAX (film_id),
       AVG (film_id)
       FROM film
```

	min integer	max integer	avg numeric
1	1	1001	501.0000000

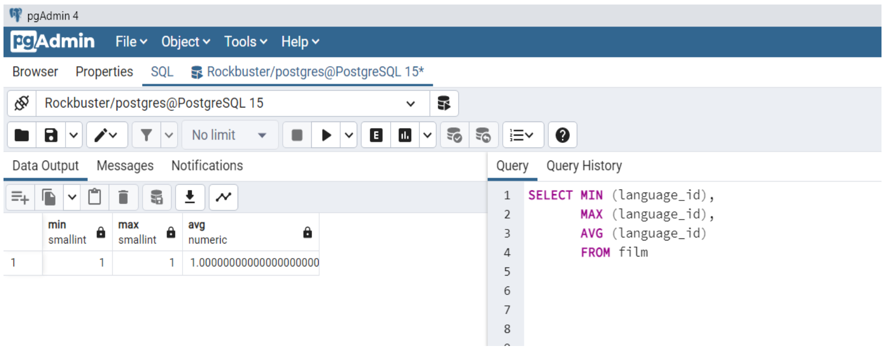
release_year:



```
SELECT MIN (release_year),
       MAX (release_year),
       AVG (release_year)
       FROM film
```

	min integer	max integer	avg numeric
1	2006	2019	2006.0129870129870130

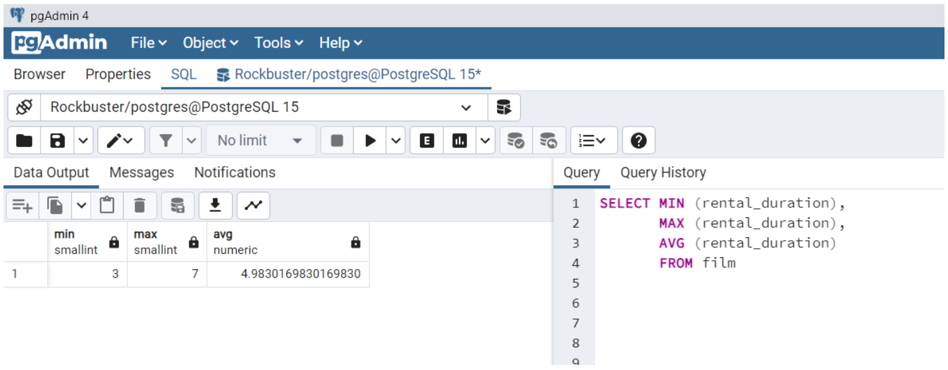
language_id:



```
SELECT MIN (language_id),
       MAX (language_id),
       AVG (language_id)
       FROM film
```

	min smallint	max smallint	avg numeric
1	1	1	1.00000000000000000000

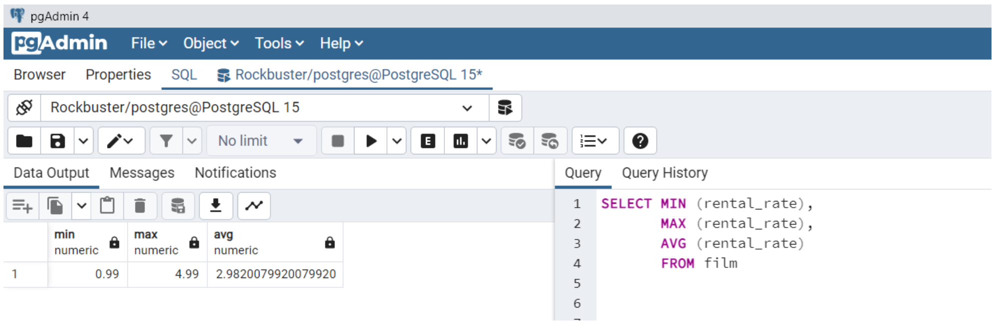
rental_duration:


pgAdmin 4
File Object Tools Help
Browser Properties SQL Rockbuster/postgres@PostgreSQL 15*
Rockbuster/postgres@PostgreSQL 15
No limit
Data Output Messages Notifications
Query Query History

	min smallint	max smallint	avg numeric
1	3	7	4.9830169830169830

```
SELECT MIN (rental_duration),
       MAX (rental_duration),
       AVG (rental_duration)
       FROM film
```

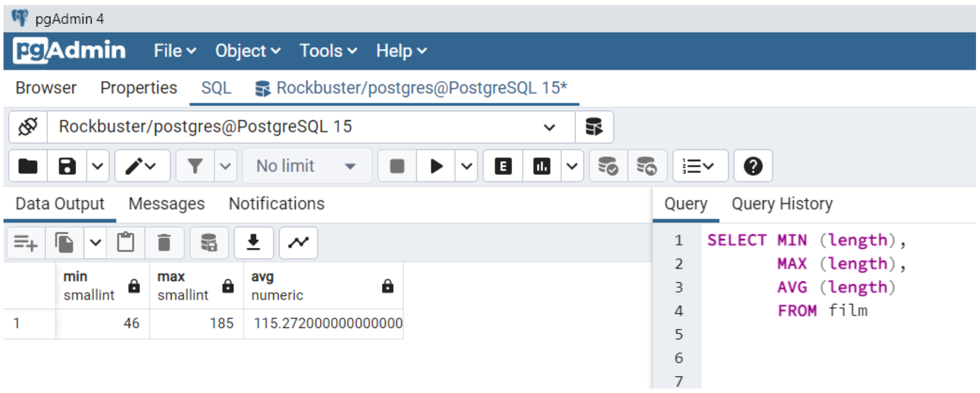
rental_rate:


pgAdmin 4
File Object Tools Help
Browser Properties SQL Rockbuster/postgres@PostgreSQL 15*
Rockbuster/postgres@PostgreSQL 15
No limit
Data Output Messages Notifications
Query Query History

	min numeric	max numeric	avg numeric
1	0.99	4.99	2.9820079920079920

```
SELECT MIN (rental_rate),
       MAX (rental_rate),
       AVG (rental_rate)
       FROM film
```

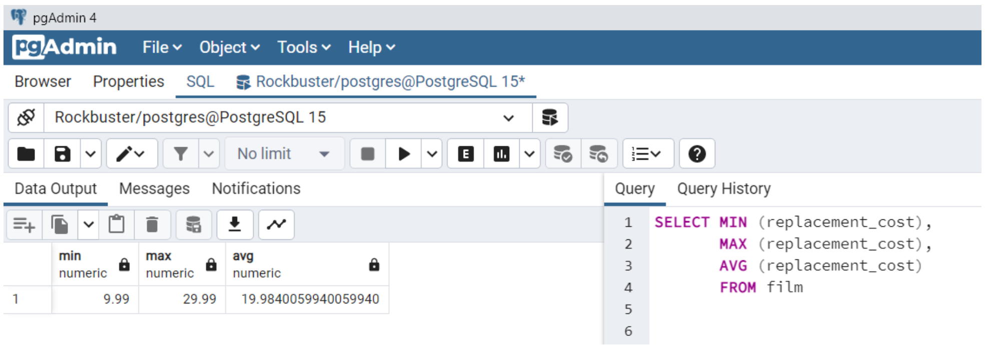
length:


pgAdmin 4
File Object Tools Help
Browser Properties SQL Rockbuster/postgres@PostgreSQL 15*
Rockbuster/postgres@PostgreSQL 15
No limit
Data Output Messages Notifications
Query Query History

	min smallint	max smallint	avg numeric
1	46	185	115.2720000000000000

```
SELECT MIN (length),
       MAX (length),
       AVG (length)
       FROM film
```

replacement_cost:


pgAdmin 4
File Object Tools Help
Browser Properties SQL Rockbuster/postgres@PostgreSQL 15*
Rockbuster/postgres@PostgreSQL 15
No limit
Data Output Messages Notifications
Query Query History

	min numeric	max numeric	avg numeric
1	9.99	29.99	19.9840059940059940

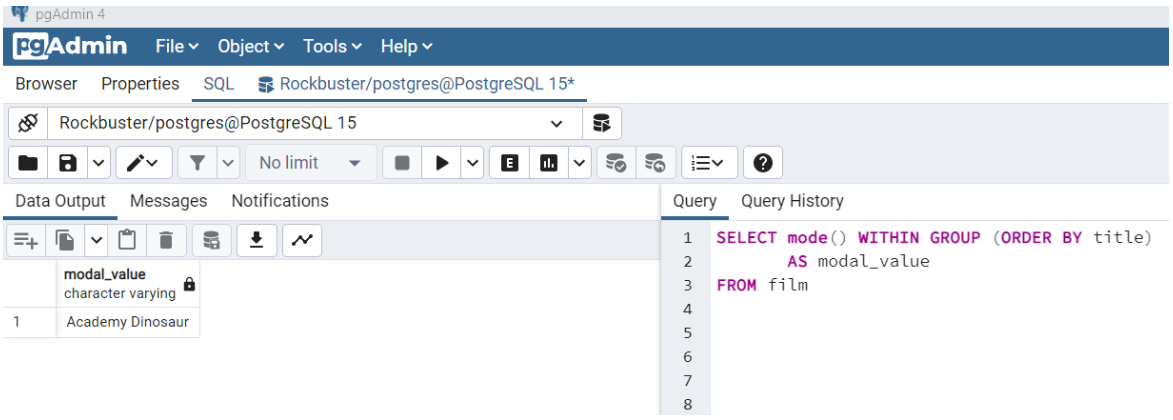
```
SELECT MIN (replacement_cost),
       MAX (replacement_cost),
       AVG (replacement_cost)
       FROM film
```

Non-numeric values: title, rating, special_features, fulltext

Title:

Statement for MODE:

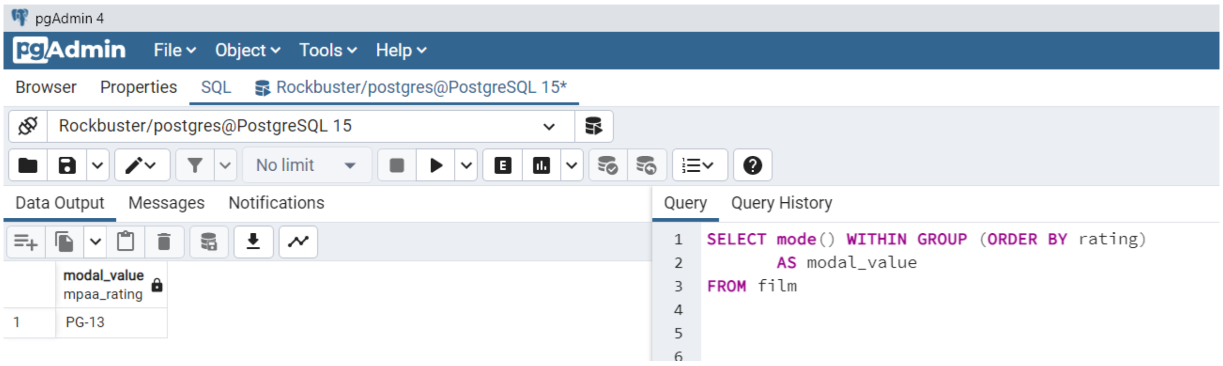
```
SELECT mode() WITHIN GROUP (ORDER BY title)
        AS modal_value
FROM film
```



```
SELECT mode() WITHIN GROUP (ORDER BY title)
        AS modal_value
FROM film
```

	modal_value character varying
1	Academy Dinosaur

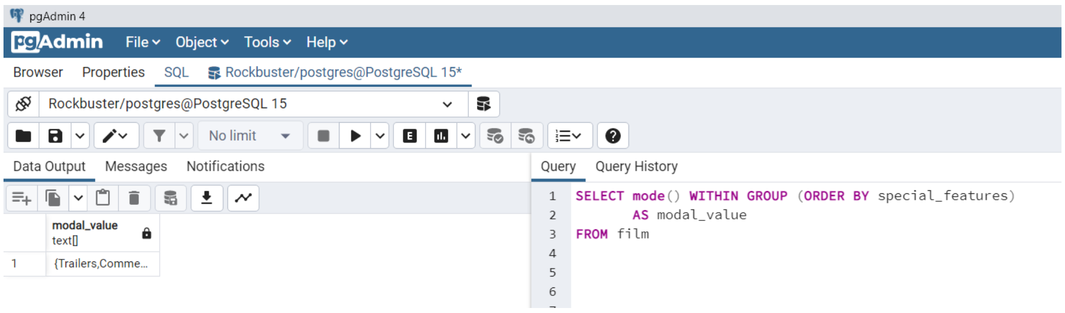
Rating:



```
SELECT mode() WITHIN GROUP (ORDER BY rating)
        AS modal_value
FROM film
```

	modal_value mpaa_rating
1	PG-13

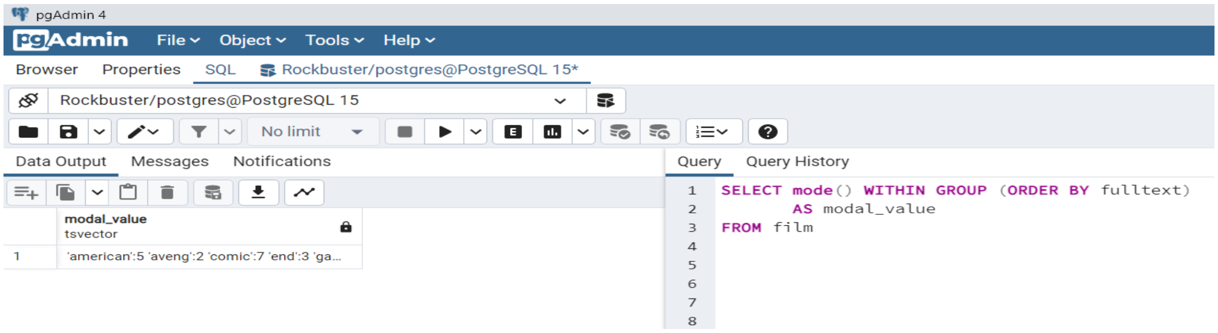
special_features:



```
SELECT mode() WITHIN GROUP (ORDER BY special_features)
        AS modal_value
FROM film
```

	modal_value text[]
1	{Trailers,Comme...

Fulltext:



```sql
SELECT mode() WITHIN GROUP (ORDER BY fulltext)
       AS modal_value
FROM film
```

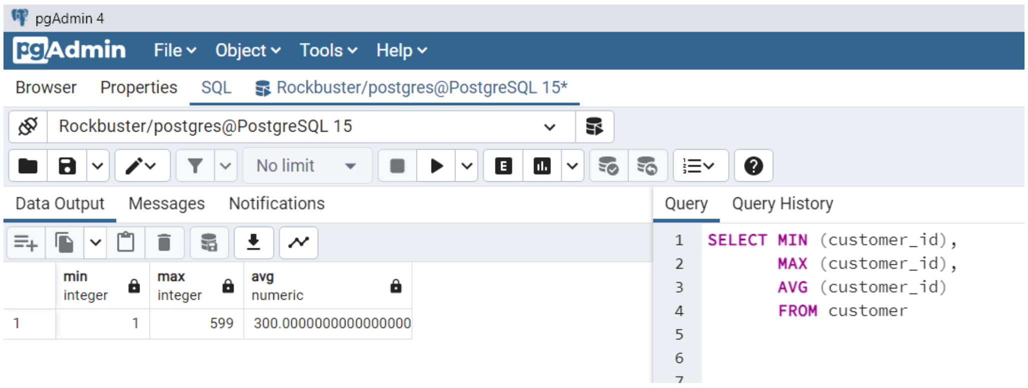
	modal_value tsvector
1	'american':5 'aveng':2 'comic':7 'end':3 'ga...

Customer table

Numeric values: customer_id, store_id, address_id

(note: the activebool and active, are both bolean values, - i.e., true or false)

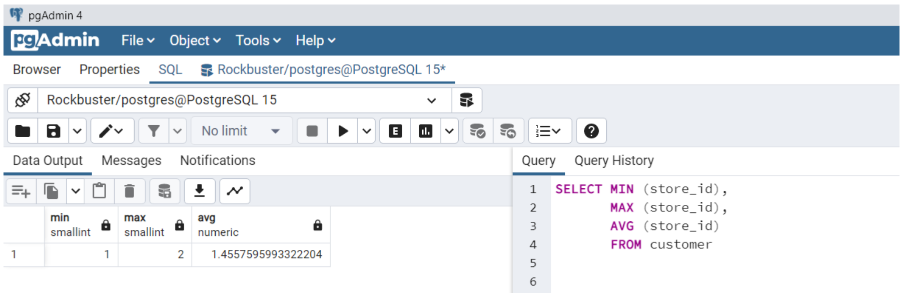
customer_id:



```sql
SELECT MIN (customer_id),
       MAX (customer_id),
       AVG (customer_id)
       FROM customer
```

	min integer	max integer	avg numeric
1	1	599	300.0000000000000000

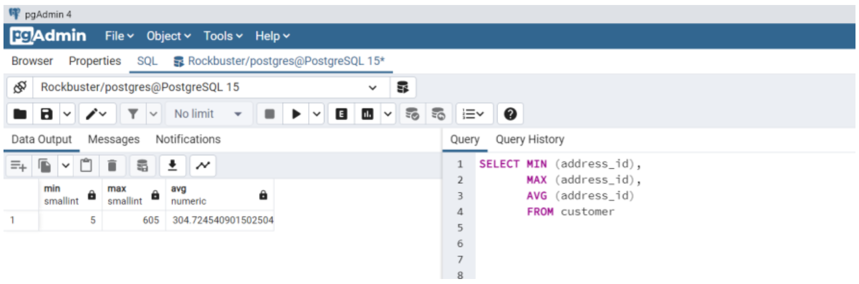
store_id:



```sql
SELECT MIN (store_id),
       MAX (store_id),
       AVG (store_id)
       FROM customer
```

	min smallint	max smallint	avg numeric
1	1	2	1.4557595993322204

address_id:

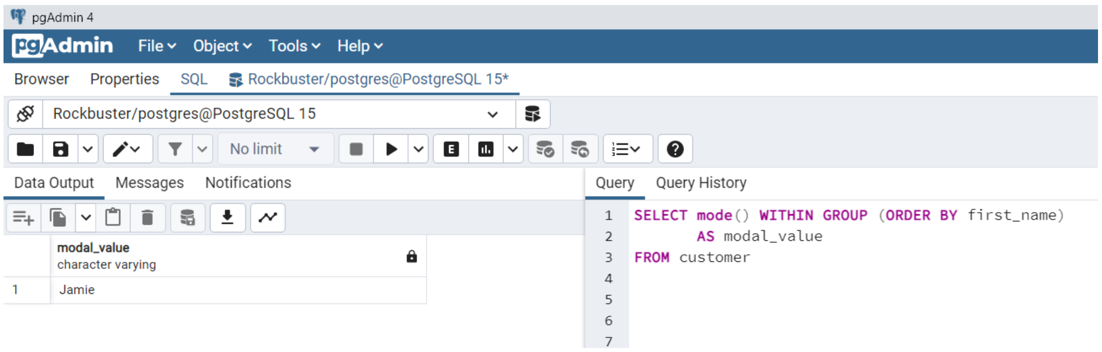


```sql
SELECT MIN (address_id),
       MAX (address_id),
       AVG (address_id)
       FROM customer
```

	min smallint	max smallint	avg numeric
1	5	605	304.724540901502504

Non-numeric values: first_name, last_name, email

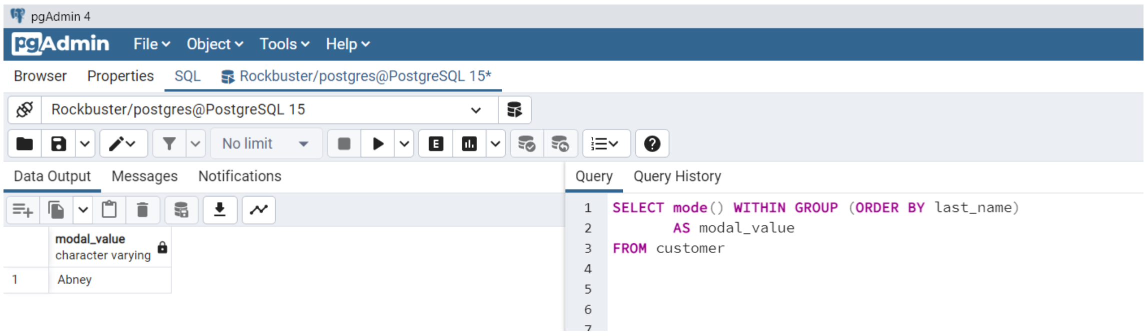
first_name:



```
SELECT mode() WITHIN GROUP (ORDER BY first_name)
       AS modal_value
FROM customer
```

```
   modal_value
   character varying
1  Jamie
```

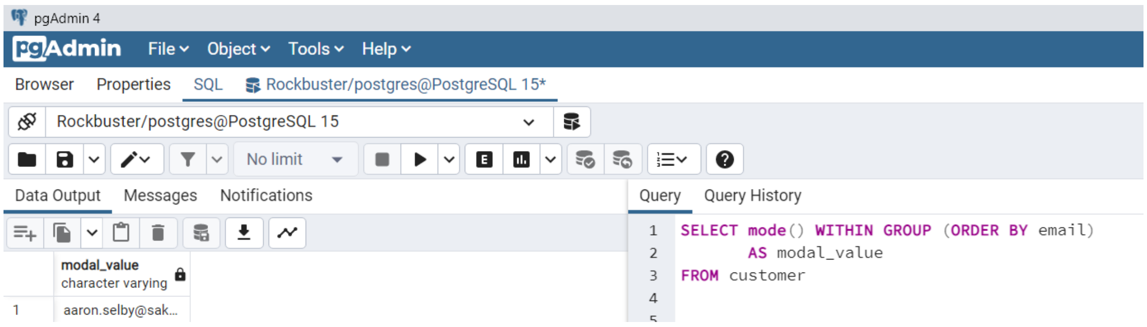
last_name



```
SELECT mode() WITHIN GROUP (ORDER BY last_name)
       AS modal_value
FROM customer
```

```
   modal_value
   character varying
1  Abney
```

email:



```
SELECT mode() WITHIN GROUP (ORDER BY email)
       AS modal_value
FROM customer
```

```
   modal_value
   character varying
1  aaron.selby@sak...
```

3. Reflect on your work:
Back in Achievement 1 you learned about data profiling in Excel. Based on your previous experience, which tool (Excel or SQL) do you think is more effective for data profiling, and why? Consider their respective functions, ease of use, and speed. Write a short paragraph in the running document that you have started.

I think it is very easy to clean data in SQL, once you become comfortable with the statements and how to use them. In terms of data profiling, I also find it easier in SQL, we can check the data type in the columns of the table, while in excel the value could be 'general' or 'numeric' without specifications, so it is not always obvious exactly what the range of the numbers are for instance. It is fast to perform a quick summary of the data (using descriptive statistics) in SQL, compared to the time it would take to the same calculations in excel.

Yet, I think excel is far superior when it comes to types of statistical analysis that can be done. In this aspect SQL is very basic. That said, using the MODE in SQL is interesting, I don’t think we can retrieve this type of result with all text data in excel.