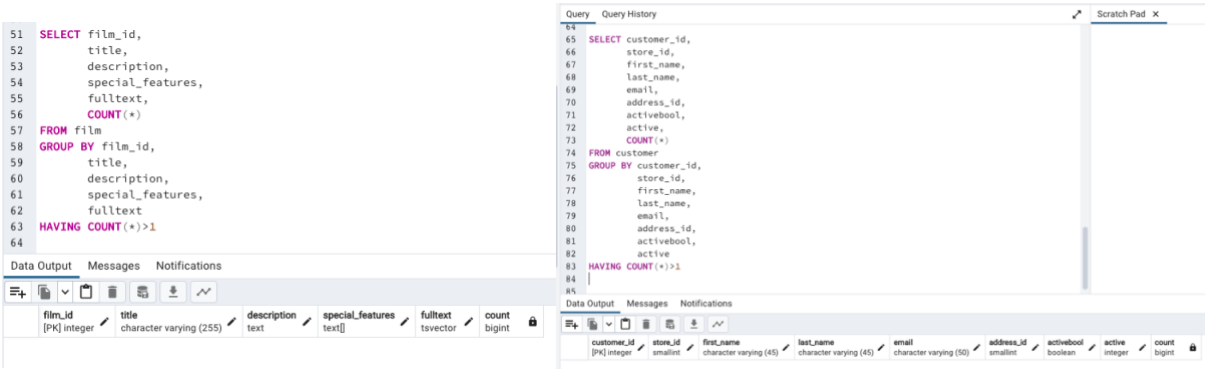
STEP 1

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.

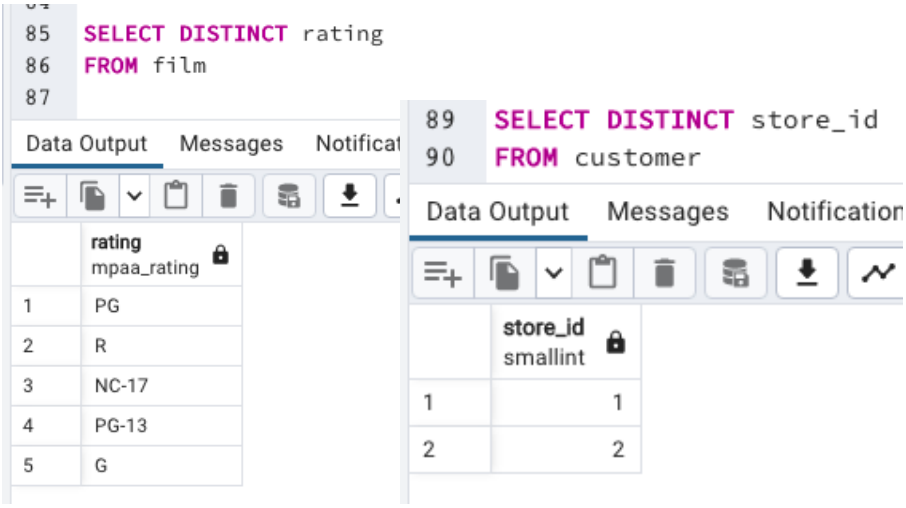
- Check for duplicate data in film and customer tables:



Both queries select needed columns and count all the rows. HAVING COUNT(*)>1 will filter out records where the count is greater than 1, so the query will return a list of duplicate records.

In case that we find duplicate records, we can either delete them (if we have access to) or create a VIEW table where we select only unique records.

- Check for non-uniform data in film and customer tables:



To check for non-uniform data, we can use the DISTINCT command and run query for each column in the table. This way we can check for inconsistencies.

In case of non-uniform data, we update the table by setting the values in one format.

- Check for missing values in film and customer tables:

```
148 SELECT *
149 FROM film
150 WHERE NOT (film is NOT 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_

```
152 SELECT *
153 FROM customer
154 WHERE NOT (customer IS NOT NULL)
```

Data Output | Messages | Notifications

id	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	active integer

In case of missing values, we can impute the empty records with estimates. However, a preferred approach would be to omit the columns with high number of missing values in our analysis.

STEP 2

Summarize your data: Use SQL to calculate descriptive statistics for both the film table and the customer table.

Film Data

Query | Query History | Scratch Pad

```
91
92  SELECT
93  MIN(rental_duration) AS min_rental_duration,
94  MAX(rental_duration) AS max_rental_duration,
95  AVG(rental_duration) AS avg_rental_duration,
96
97  MIN(rental_rate) AS min_rental_rate,
98  MAX(rental_rate) AS max_rental_rate,
99  AVG(rental_rate) AS avg_rental_rate,
100
101 MIN(length) AS min_length,
102 MAX(length) AS max_length,
103 AVG(length) AS avg_lenth,
104
105 MIN(replacement_cost) AS min_replacement_cost,
106 MAX(replacement_cost) AS max_replacement_cost,
107 AVG(replacement_cost) AS avg_replacement_cost,
108
109 MODE() WITHIN GROUP (ORDER BY release_year) as modal_release_year,
110 MODE() WITHIN GROUP (ORDER BY language_id) AS modal_language_id,
111 MODE() WITHIN GROUP (ORDER BY rating) AS modal_rating
```

Data Output | Messages | Notifications

	max_length smallint	avg_lenth numeric	min_replacement_cost numeric	max_replacement_cost numeric	avg_replacement_cost numeric	modal_release_year integer	modal_language_id smallint	modal_rating mpaa_rating
1	185	115.2720000000000000	9.99	29.99	19.9840000000000000	2006	1	PG-13

**The whole query is not visible due to size*

Customer Data

```
SELECT
MIN(customer_id) AS min_customer_id,
MAX(customer_id) AS max_customer_id,

MODE() WITHIN GROUP (ORDER BY first_name) AS modal_first_name,
MODE() WITHIN GROUP (ORDER BY store_id) AS modal_store_id,
MODE() WITHIN GROUP (ORDER BY activebool) AS modal_activebool
FROM customer;

```

Data Output | Messages | Notifications

	min_customer_id integer	max_customer_id integer	modal_first_name character varying	modal_store_id smallint	modal_activebool boolean
1	1	599	Jamie	1	true

**Not all columns were run in the query*

STEP 3

Main advantage of SQL in comparison with Excel is the simplicity and efficiency when working with larger datasets.

When we have smaller datasets, profiling the data using pivot tables seems to be a preferable option.

Once I am used to writing queries and/or have templates to calculate descriptive statistics, the process is easier than Excel.