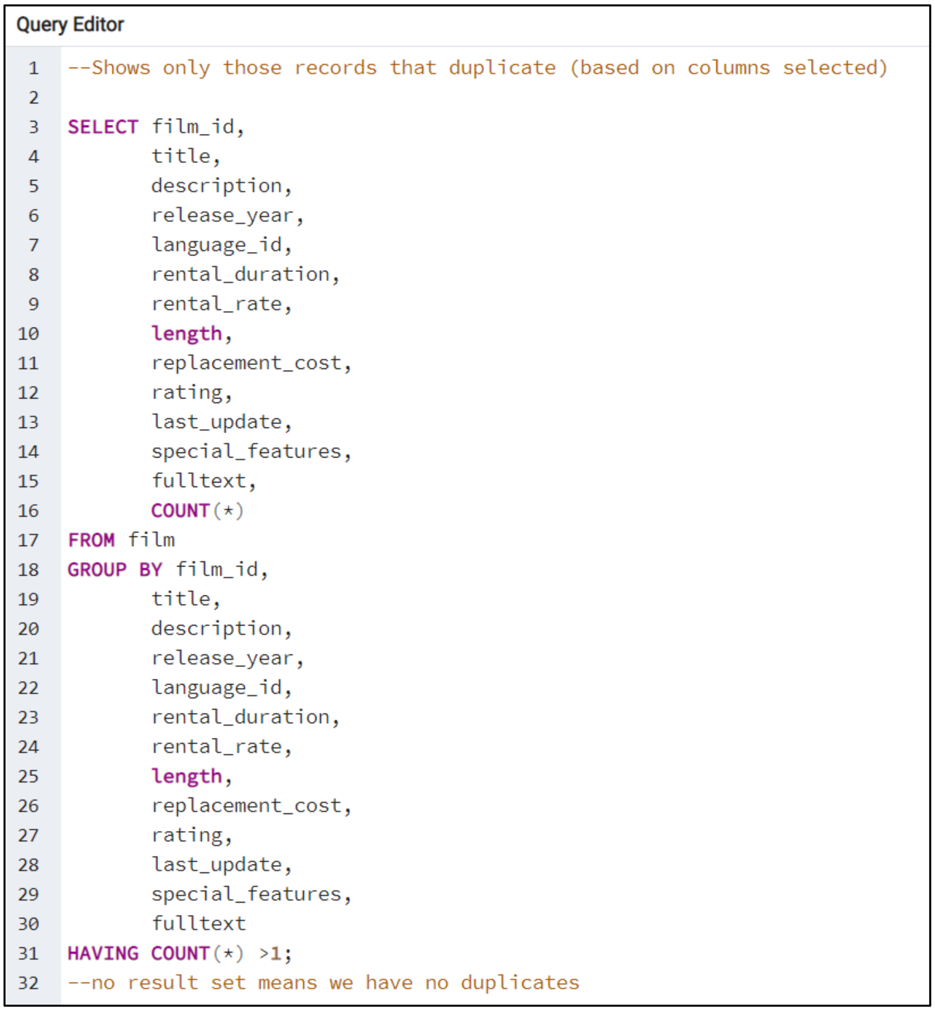
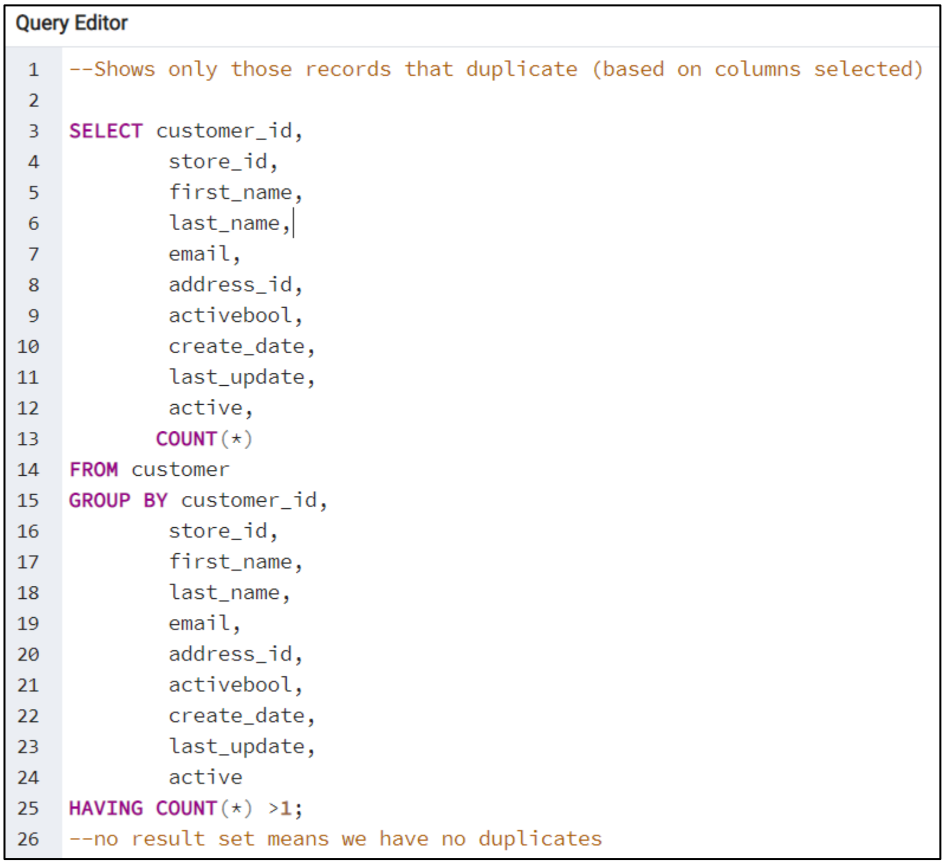
1. Check for and clean dirty data:
   - Check for duplicates
     - Film Table



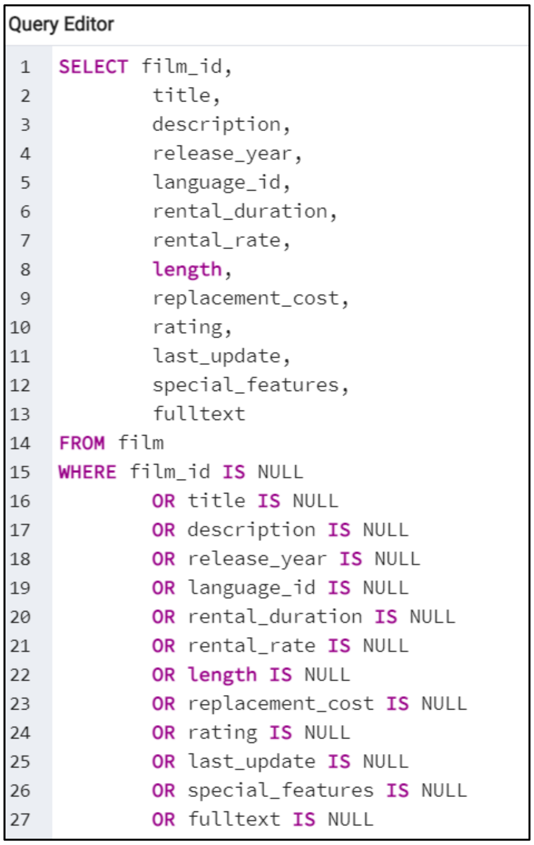
```
--Shows only those records that duplicate (based on columns selected)

SELECT film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext,
       COUNT(*)
FROM film
GROUP BY film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext
HAVING COUNT(*) >1;
--no result set means we have no duplicates
```

     - Customer Table



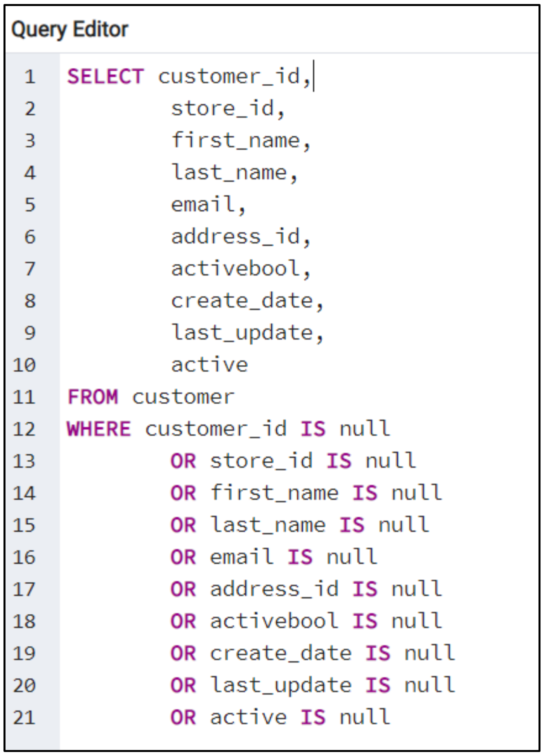
```
--Shows only those records that duplicate (based on columns selected)

SELECT customer_id,
        store_id,
        first_name,
        last_name,
        email,
        address_id,
        activebool,
        create_date,
        last_update,
        active,
       COUNT(*)
FROM customer
GROUP BY customer_id,
        store_id,
        first_name,
        last_name,
        email,
        address_id,
        activebool,
        create_date,
        last_update,
        active
HAVING COUNT(*) >1;
--no result set means we have no duplicates
```

    - If duplicate data did exist, I would create a new view of the table to show only unique records. Then I would use that view to continue my analysis. If there were a lot of duplicates, I would also notify the data engineer so they could make sure data migration and everything on their end is correct.
- Check for non-uniform data
    - Film Table Queries
        - SELECT DISTINCT title
          FROM film
        - SELECT DISTINCT release_year
          FROM film
        - SELECT DISTINCT language_id
          FROM film
        - SELECT DISTINCT rental_rate
          FROM film
        - SELECT DISTINCT replacement_cost
          FROM film
        - SELECT DISTINCT rating
          FROM film
    - Customer Table Queries
        - SELECT DISTINCT email
          FROM customer
        - SELECT DISTINCT address_id
          FROM customer
        - SELECT DISTINCT active
          FROM customer
    - If there was inconsistent data, I would update it using the UPDATE and SET commands.
- Check for missing data
    - Film Table



```
SELECT film_id,
        title,
        description,
        release_year,
        language_id,
        rental_duration,
        rental_rate,
        length,
        replacement_cost,
        rating,
        last_update,
        special_features,
        fulltext
FROM film
WHERE film_id IS NULL
        OR title IS NULL
        OR description IS NULL
        OR release_year IS NULL
        OR language_id IS NULL
        OR rental_duration IS NULL
        OR rental_rate IS NULL
        OR length IS NULL
        OR replacement_cost IS NULL
        OR rating IS NULL
        OR last_update IS NULL
        OR special_features IS NULL
        OR fulltext IS NULL
```

- Customer Table

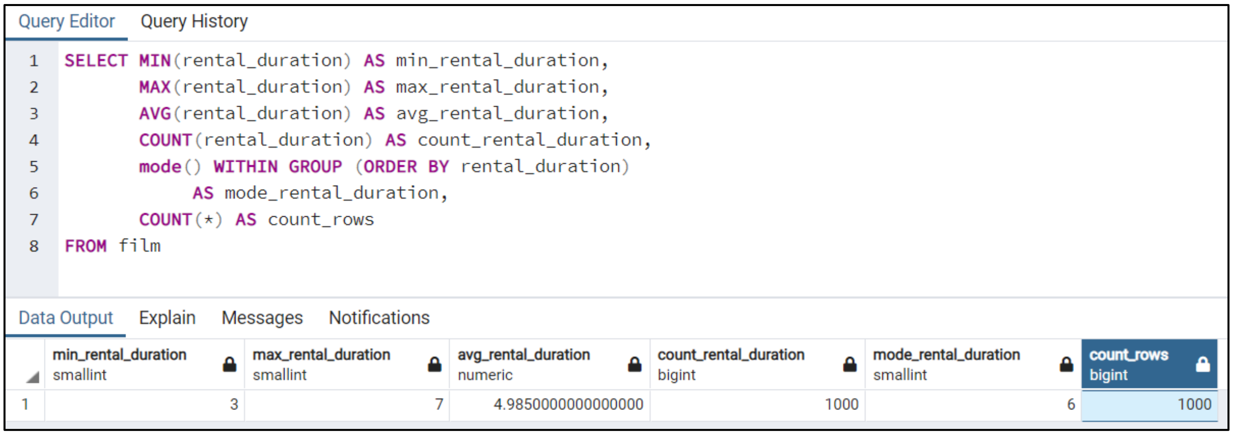


```sql
SELECT customer_id,
        store_id,
        first_name,
        last_name,
        email,
        address_id,
        activebool,
        create_date,
        last_update,
        active
FROM customer
WHERE customer_id IS null
        OR store_id IS null
        OR first_name IS null
        OR last_name IS null
        OR email IS null
        OR address_id IS null
        OR activebool IS null
        OR create_date IS null
        OR last_update IS null
        OR active IS null
```

- If there was missing data in a column that I don’t need for analysis, I would ignore the column entirely. If a small amount of data were missing from a column with numeric values that could be averaged, I would impute the average or mean in place of the NULL values and note it in my analysis.

2. Summarize data:

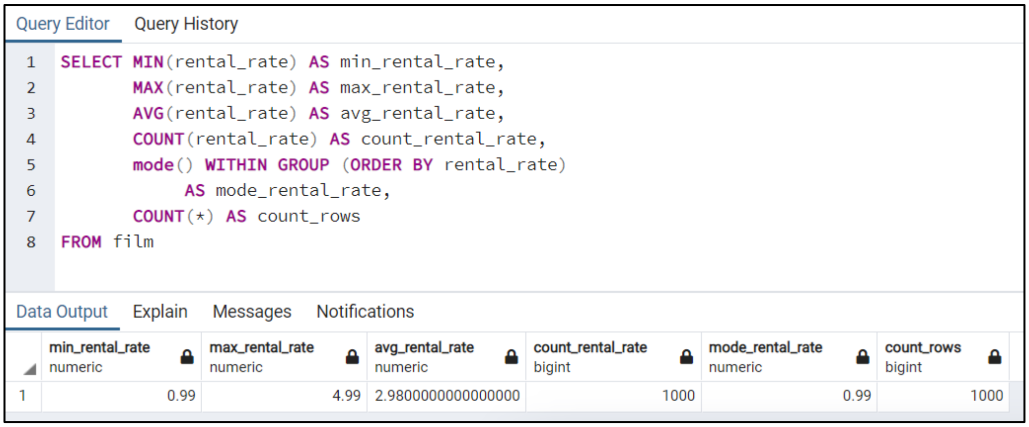
- Film Table
  - Rental Duration



```sql
SELECT MIN(rental_duration) AS min_rental_duration,
       MAX(rental_duration) AS max_rental_duration,
       AVG(rental_duration) AS avg_rental_duration,
       COUNT(rental_duration) AS count_rental_duration,
       mode() WITHIN GROUP (ORDER BY rental_duration)
           AS mode_rental_duration,
       COUNT(*) AS count_rows
FROM film
```

	min_rental_duration smallint	max_rental_duration smallint	avg_rental_duration numeric	count_rental_duration bigint	mode_rental_duration smallint	count_rows bigint
1	3	7	4.9850000000000000	1000	6	1000

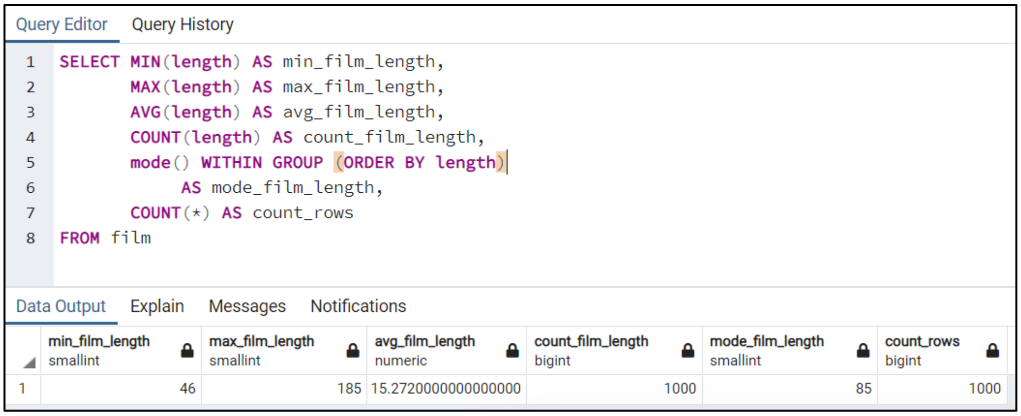
  - Rental Rate



```sql
SELECT MIN(rental_rate) AS min_rental_rate,
       MAX(rental_rate) AS max_rental_rate,
       AVG(rental_rate) AS avg_rental_rate,
       COUNT(rental_rate) AS count_rental_rate,
       mode() WITHIN GROUP (ORDER BY rental_rate)
           AS mode_rental_rate,
       COUNT(*) AS count_rows
FROM film
```

	min_rental_rate numeric	max_rental_rate numeric	avg_rental_rate numeric	count_rental_rate bigint	mode_rental_rate numeric	count_rows bigint
1	0.99	4.99	2.9800000000000000	1000	0.99	1000

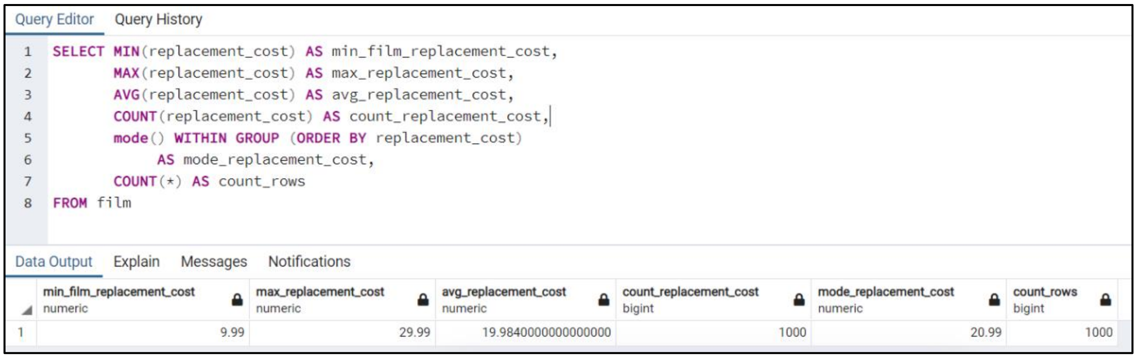
- Length



```
SELECT MIN(length) AS min_film_length,
       MAX(length) AS max_film_length,
       AVG(length) AS avg_film_length,
       COUNT(length) AS count_film_length,
       mode() WITHIN GROUP (ORDER BY length)
           AS mode_film_length,
       COUNT(*) AS count_rows
FROM film
```

	min_film_length smallint	max_film_length smallint	avg_film_length numeric	count_film_length bigint	mode_film_length smallint	count_rows bigint
1	46	185	15.2720000000000000	1000	85	1000

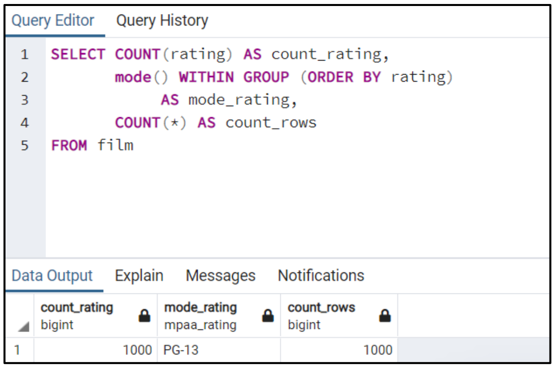
- Replacement Cost



```
SELECT MIN(replacement_cost) AS min_film_replacement_cost,
       MAX(replacement_cost) AS max_replacement_cost,
       AVG(replacement_cost) AS avg_replacement_cost,
       COUNT(replacement_cost) AS count_replacement_cost,
       mode() WITHIN GROUP (ORDER BY replacement_cost)
           AS mode_replacement_cost,
       COUNT(*) AS count_rows
FROM film
```

	min_film_replacement_cost numeric	max_replacement_cost numeric	avg_replacement_cost numeric	count_replacement_cost bigint	mode_replacement_cost numeric	count_rows bigint
1	9.99	29.99	19.9840000000000000	1000	20.99	1000

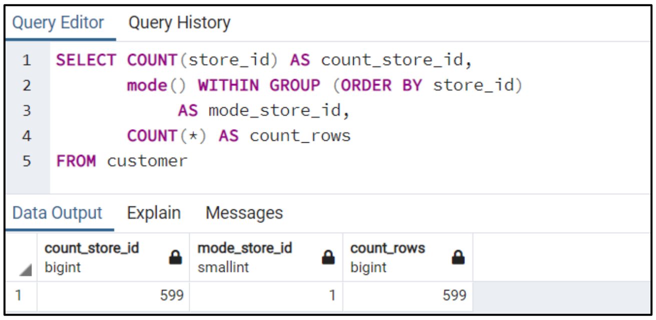
- Rating



```
SELECT COUNT(rating) AS count_rating,
       mode() WITHIN GROUP (ORDER BY rating)
           AS mode_rating,
       COUNT(*) AS count_rows
FROM film
```

	count_rating bigint	mode_rating mpaa_rating	count_rows bigint
1	1000	PG-13	1000

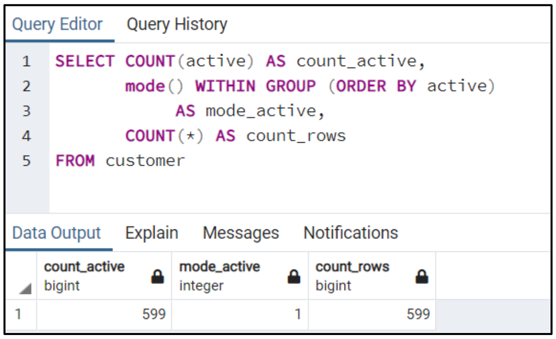
- Customer Table
  - Store ID



```
SELECT COUNT(store_id) AS count_store_id,
       mode() WITHIN GROUP (ORDER BY store_id)
           AS mode_store_id,
       COUNT(*) AS count_rows
FROM customer
```

	count_store_id bigint	mode_store_id smallint	count_rows bigint
1	599	1	599

  - Active



```
SELECT COUNT(active) AS count_active,
       mode() WITHIN GROUP (ORDER BY active)
           AS mode_active,
       COUNT(*) AS count_rows
FROM customer
```

	count_active bigint	mode_active integer	count_rows bigint
1	599	1	599

3. SQL is more efficient than Excel for data profiling because SQL performs the summary calculations in one query and only returns the information requested. Excel in comparison, requires you to have an entire data set in hand, then perform additional steps and calculations to get data summary information.