Julio 6, 2022

Data Immersion
Databases & SQL for Analysts
3.6: Summarizing & Cleaning Data in SQL
David Guillen Aroche

1. **Check for and clean dirty data for the film table and the customer table.**

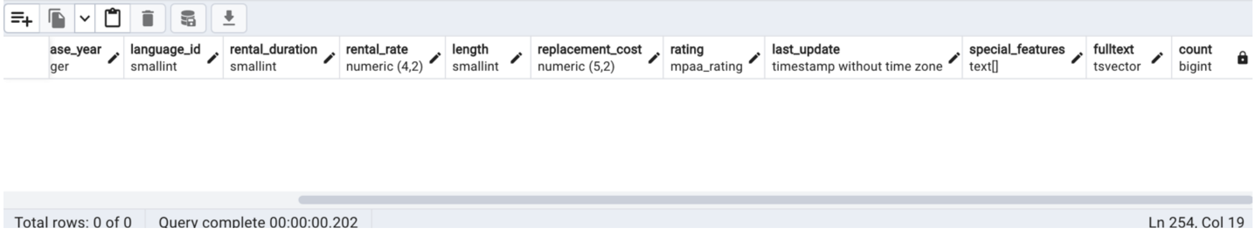
- **Duplicates.**

***Film Table***

Query:

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

Result:
No duplicates were found in film table, see screenshot below.



| ase_year<br>ger | language_id<br>smallint | rental_duration<br>smallint | rental_rate<br>numeric (4,2) | length<br>smallint | replacement_cost<br>numeric (5,2) | rating<br>mpaa_rating | last_update<br>timestamp without time zone | special_features<br>text[] | fulltext<br>tsvector | count<br>bigint |
|---|---|---|---|---|---|---|---|---|---|---|

Total rows: 0 of 0   Query complete 00:00:00.202   Ln 254, Col 19

In case of finding duplicate values, I would have created a view with unique records with the following syntax to make each row unique:

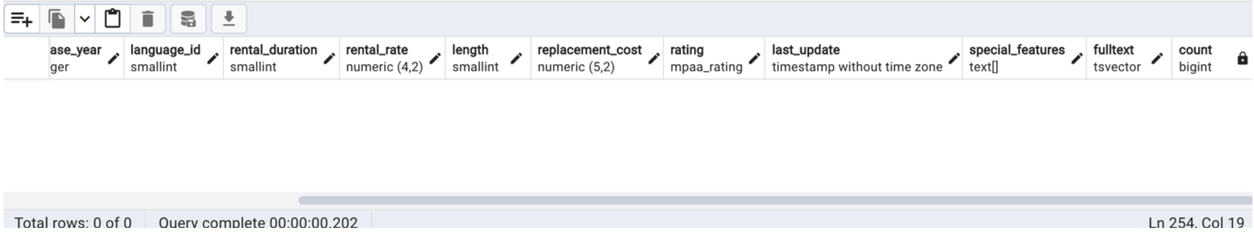
```
CREATE VIEW viewname AS
SELECT col1,
       col2,
       col3 ...
FROM tablename
GROUP BY col1,
         col2,
         col3, ... ;
```

***Customer Table***

Query:

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

Result:
No duplicates were found in customers table, see screenshot below.



| ase_year | language_id | rental_duration | rental_rate | length | replacement_cost | rating | last_update | special_features | fulltext | count |
|---|---|---|---|---|---|---|---|---|---|---|
| ger | smallint | smallint | numeric (4,2) | smallint | numeric (5,2) | mpaa_rating | timestamp without time zone | text[] | tsvector | bigint |

Total rows: 0 of 0 Query complete 00:00:00.202 Ln 254, Col 19

In case of finding duplicate values, I would have created a view with unique records with the following syntax to make each row unique:

```
CREATE VIEW viewname AS
SELECT col1,
       col2,
       col3 ...
FROM tablename
GROUP BY col1,
         col2,
         col3, ... ;
```

- **Non-uniform data.**

***Film Table***

Query (syntax):

```
SELECT DISTINCT column_name
FROM ctable_name
GROUP BY column_name
```

Results:
Non-unifrom values were searched in the table, but no irregularities were found. The total number of values for each column is as follows:

- *film_id, 1000 values*
- *title, 1000 values*
- *description, 1000 values*
- *release_year, 1 value*

- *language_id, 1 value*
- *rental_duration, 5 values*
- *rental_rate, 3 values*
- *film.length, 140 values*
- *replacement_cost, 21 values*
- *rating, 5 values*
- *last_update, 1 value*
- *special_features, 15 vlaues*
- *fulltext, 1000 values*

In case of finding a non-uniform value, first, I would have grouped the values to get familiar with them to choose a standard format, and checked on data types for each column. Then use the following syntax to update values:

```
UPDATE film
SET rating = 'G'
WHERE rating IN ('gen',
                 'g',
                 'General')
```

***Customer Table***

Query (syntax):

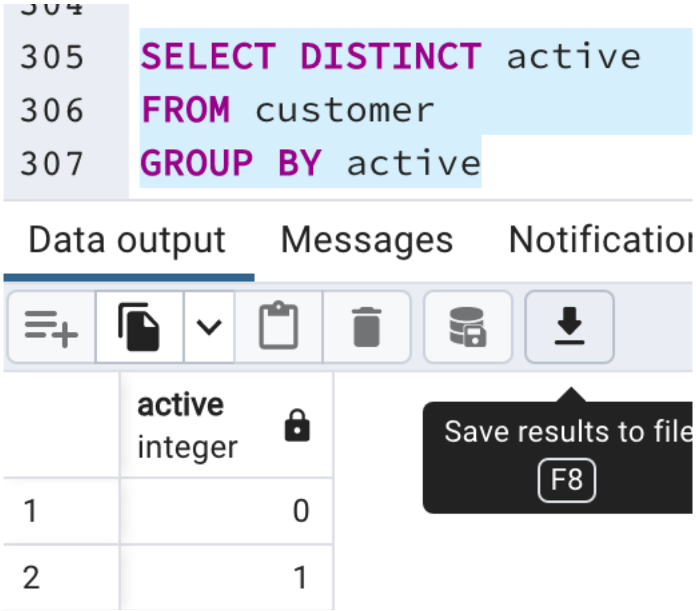
```
SELECT DISTINCT column_name
FROM ctable_name
GROUP BY column_name
```

Results:

No Non-uniform values were searched and have found no irregularities. The total number of values for each column as follows:

- *customer_id, 599 values*
- *store_id, 2 values*
- *first_name, 591 values*
- *last_name, 599 values*
- *email, 599 values*
- *address_id, 599 values*
- *activebool, 1 value*
- *create_date, 1 value*
- *active, 1 value*

e.g.:



```sql
SELECT DISTINCT active
FROM customer
GROUP BY active
```

|  | active<br>integer |
|---|---|
| 1 | 0 |
| 2 | 1 |

In case of finding a non-uniform value, first, I would have grouped the values to get familiar with them to choose a standard format, and checked on data types for each column. Then use the following syntax to update values:

```sql
UPDATE film

SET rating = 'G'

WHERE rating IN ('gen',

                 'g',

                 'General')
```

- **Missing values**

***Film Table***

While looking for **Non-uniform** values, no missing values were found.

In case of have found missing values, I would have checked on the proportion of the missing data for each column to determine if would have been valid to include them in the analysis or not, and adapt the analysis to mitigate the impact as much as possible.

***Customer Table***

While looking for **Non-uniform** values, no missing values were found.

In case of have found missing values, I would have checked on the proportion of the missing data for each column to determine if would have been valid to include them in the analysis or not, and adapt the analysis to mitigate the impact as much as possible.

## 2. Summarize Data.

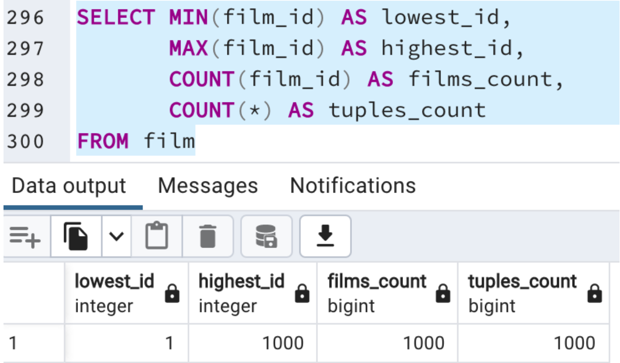
***Film Table***

Query (Syntax) Numeric Columns:

```
SELECT MIN(column_name) AS column_alias,
    MAX(column_name) AS column_alias,
    AVG(column_name) AS column_alias,
    COUNT(column_name) AS column_alias,
    COUNT(*) AS column_alias
FROM table_name
```

*--film_id.*

The average was not included for this query as it does not really say much.

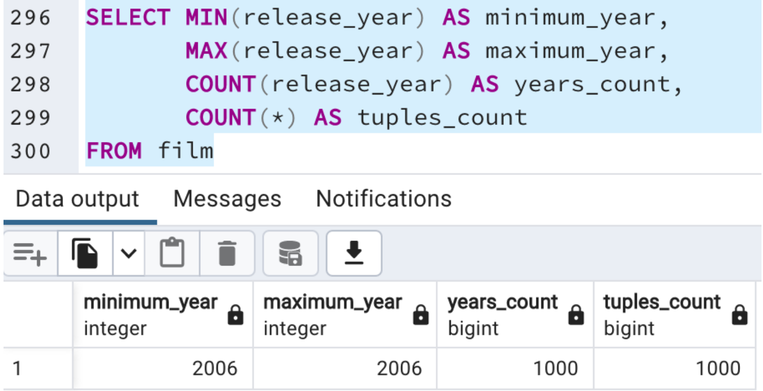


```
SELECT MIN(film_id) AS lowest_id,
       MAX(film_id) AS highest_id,
       COUNT(film_id) AS films_count,
       COUNT(*) AS tuples_count
FROM film
```

| | lowest_id<br>integer | highest_id<br>integer | films_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|
| 1 | 1 | 1000 | 1000 | 1000 |

*--release_year*

The average **years count** was not included since there is only one value for this column.

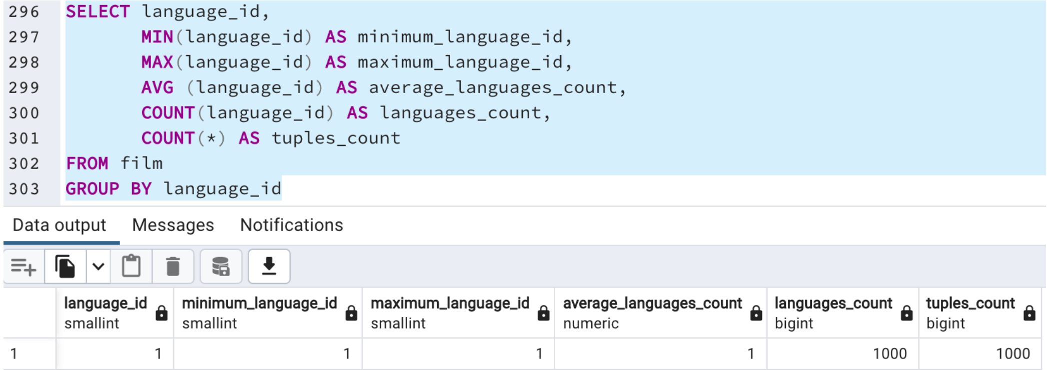


```
SELECT MIN(release_year) AS minimum_year,
       MAX(release_year) AS maximum_year,
       COUNT(release_year) AS years_count,
       COUNT(*) AS tuples_count
FROM film
```

| | minimum_year<br>integer | maximum_year<br>integer | years_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|
| 1 | 2006 | 2006 | 1000 | 1000 |

*--language_id*

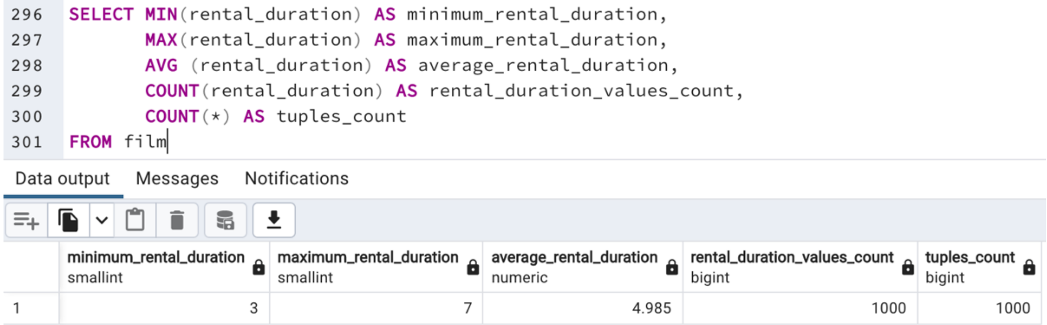
There is only one language value for the total tuples in the table.



```
SELECT language_id,
       MIN(language_id) AS minimum_language_id,
       MAX(language_id) AS maximum_language_id,
       AVG (language_id) AS average_languages_count,
       COUNT(language_id) AS languages_count,
       COUNT(*) AS tuples_count
FROM film
GROUP BY language_id
```

| | language_id | minimum_language_id | maximum_language_id | average_languages_count | languages_count | tuples_count |
|---|---|---|---|---|---|---|
| | smallint | smallint | smallint | numeric | bigint | bigint |
| 1 | 1 | 1 | 1 | 1 | 1000 | 1000 |

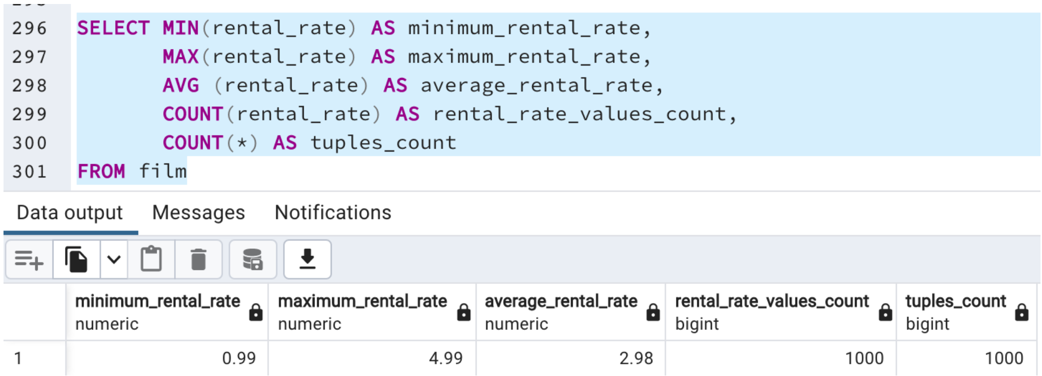
*--rental_duration*



```
SELECT MIN(rental_duration) AS minimum_rental_duration,
       MAX(rental_duration) AS maximum_rental_duration,
       AVG (rental_duration) AS average_rental_duration,
       COUNT(rental_duration) AS rental_duration_values_count,
       COUNT(*) AS tuples_count
FROM film
```

| | minimum_rental_duration | maximum_rental_duration | average_rental_duration | rental_duration_values_count | tuples_count |
|---|---|---|---|---|---|
| | smallint | smallint | numeric | bigint | bigint |
| 1 | 3 | 7 | 4.985 | 1000 | 1000 |

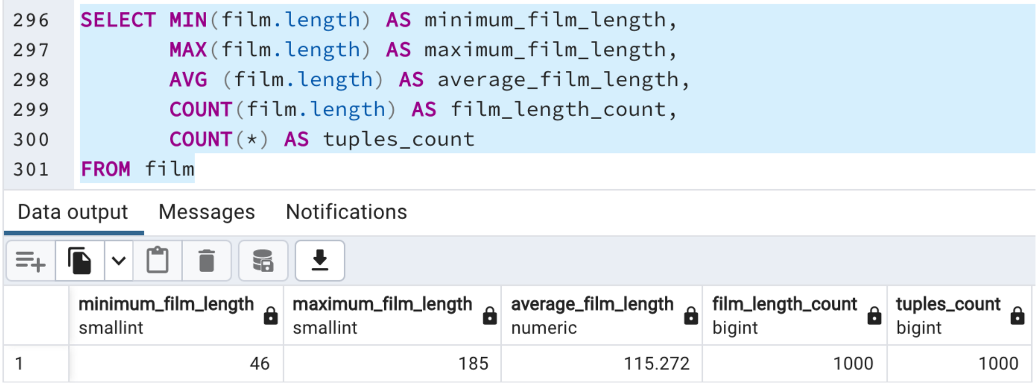
*--rental_rate*



```
SELECT MIN(rental_rate) AS minimum_rental_rate,
       MAX(rental_rate) AS maximum_rental_rate,
       AVG (rental_rate) AS average_rental_rate,
       COUNT(rental_rate) AS rental_rate_values_count,
       COUNT(*) AS tuples_count
FROM film
```

| | minimum_rental_rate | maximum_rental_rate | average_rental_rate | rental_rate_values_count | tuples_count |
|---|---|---|---|---|---|
| | numeric | numeric | numeric | bigint | bigint |
| 1 | 0.99 | 4.99 | 2.98 | 1000 | 1000 |

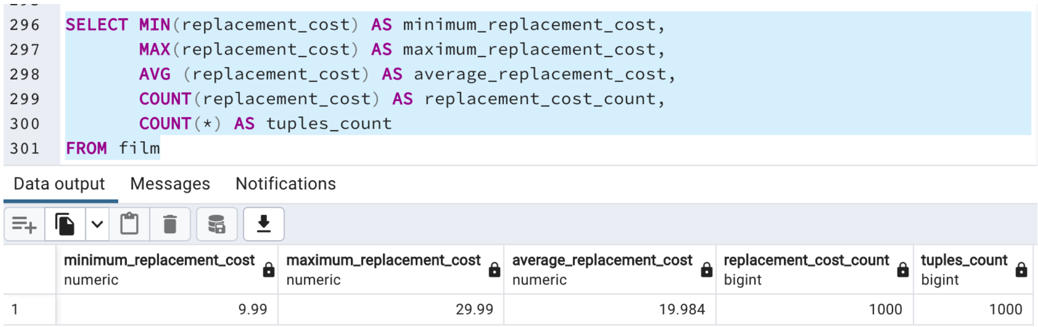
*--film.length*



```
SELECT MIN(film.length) AS minimum_film_length,
       MAX(film.length) AS maximum_film_length,
       AVG (film.length) AS average_film_length,
       COUNT(film.length) AS film_length_count,
       COUNT(*) AS tuples_count
FROM film
```

| | minimum_film_length<br>smallint | maximum_film_length<br>smallint | average_film_length<br>numeric | film_length_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|---|
| 1 | 46 | 185 | 115.272 | 1000 | 1000 |

*--replacement_cost*



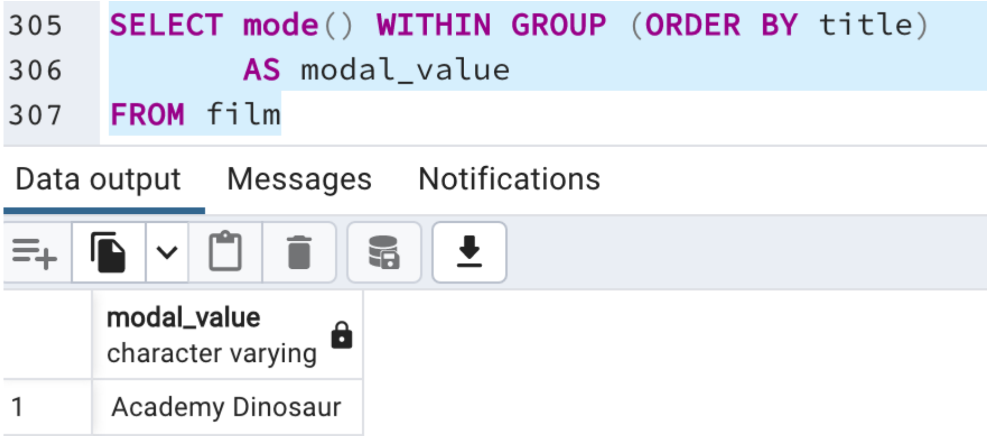
```
SELECT MIN(replacement_cost) AS minimum_replacement_cost,
       MAX(replacement_cost) AS maximum_replacement_cost,
       AVG (replacement_cost) AS average_replacement_cost,
       COUNT(replacement_cost) AS replacement_cost_count,
       COUNT(*) AS tuples_count
FROM film
```

| | minimum_replacement_cost<br>numeric | maximum_replacement_cost<br>numeric | average_replacement_cost<br>numeric | replacement_cost_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|---|
| 1 | 9.99 | 29.99 | 19.984 | 1000 | 1000 |

Query (Syntax) Non-Numeric Columns:

```
SELECT mode() WITHIN GROUP (ORDER BY column_name)
    AS modal_value
FROM tablename
```

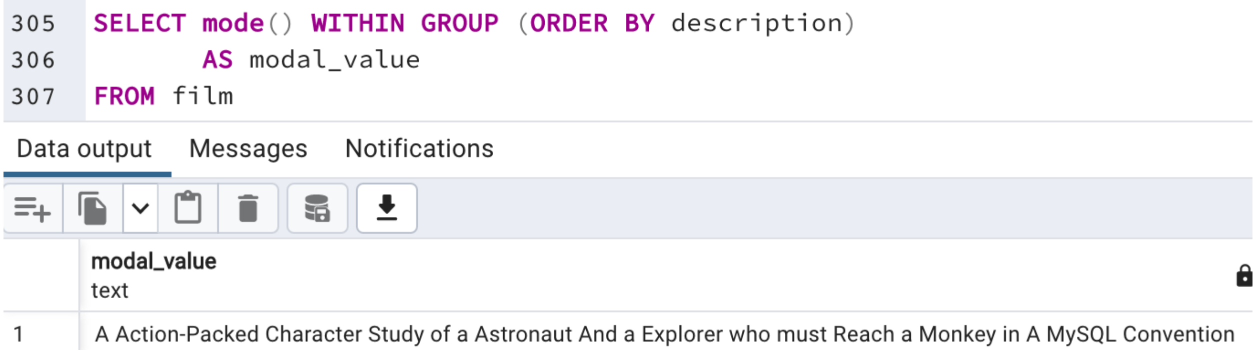
*--title*



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

| | modal_value<br>character varying |
|---|---|
| 1 | Academy Dinosaur |

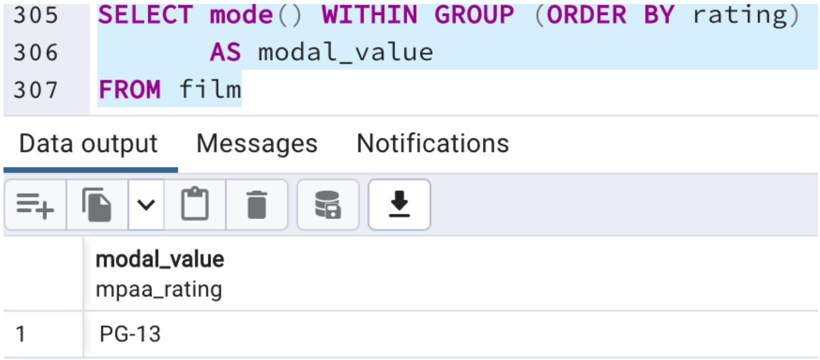
*--description*



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

| | modal_value<br>text |
|---|---|
| 1 | A Action-Packed Character Study of a Astronaut And a Explorer who must Reach a Monkey in A MySQL Convention |

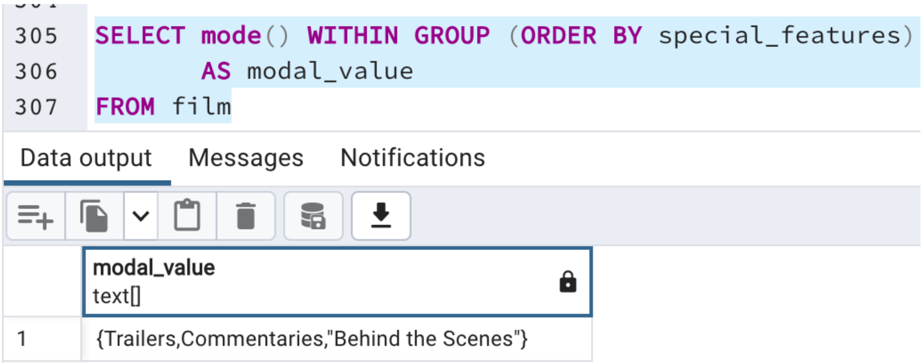
*--rating*



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

| | modal_value<br>mpaa_rating |
|---|---|
| 1 | PG-13 |

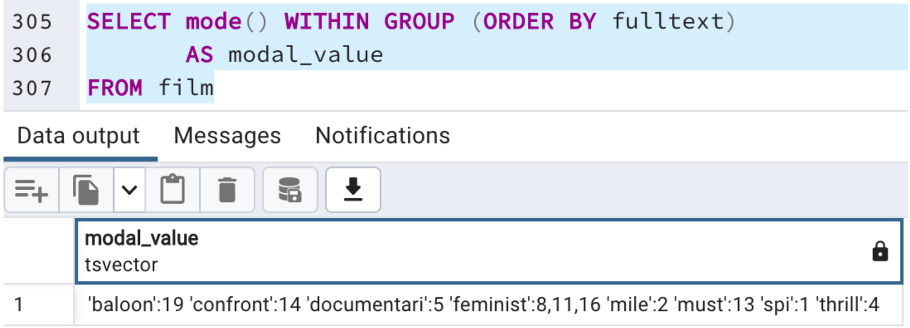
*--special_features*



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

| | modal_value<br>text[] |
|---|---|
| 1 | {Trailers,Commentaries,"Behind the Scenes"} |

*--fulltext*



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

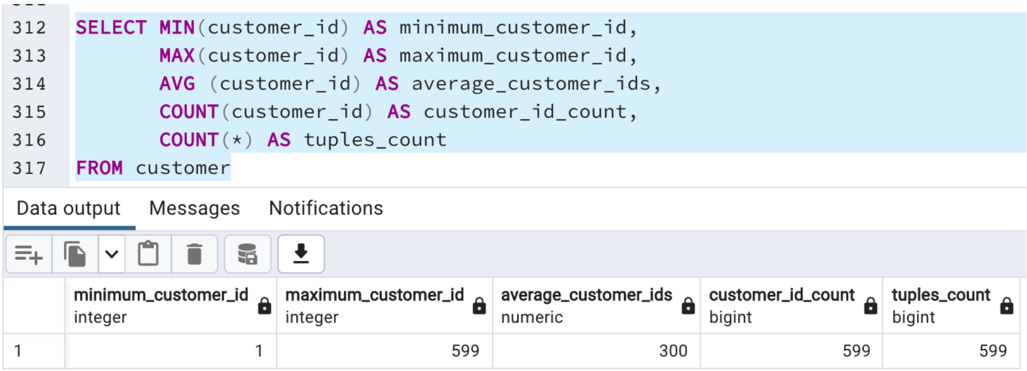
| | modal_value<br>tsvector |
|---|---|
| 1 | 'baloon':19 'confront':14 'documentari':5 'feminist':8,11,16 'mile':2 'must':13 'spi':1 'thrill':4 |

### ***Customer Table***

Query (Syntax) Numeric Columns:

```
SELECT MIN(column_name) AS column_alias,
       MAX(column_name) AS column_alias,
       AVG(column_name) AS column_alias,
       COUNT(column_name) AS column_alias,
       COUNT(*) AS column_alias
FROM table_name
```

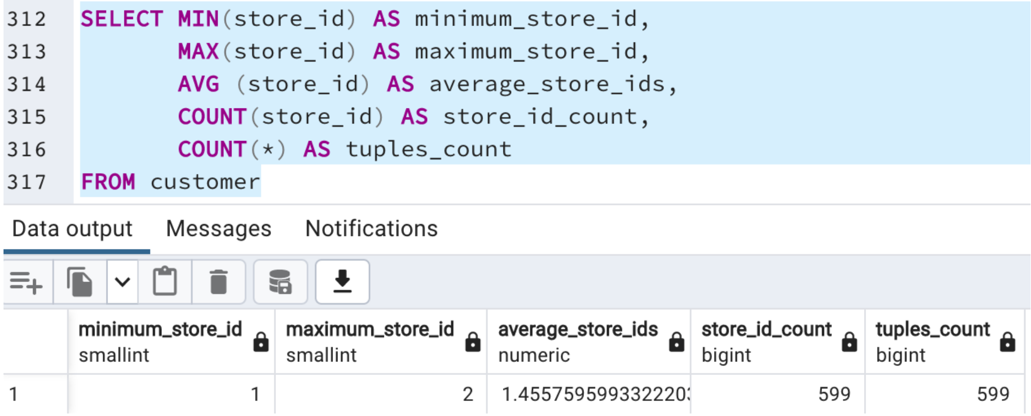
*--customer_id*



```
SELECT MIN(customer_id) AS minimum_customer_id,
       MAX(customer_id) AS maximum_customer_id,
       AVG (customer_id) AS average_customer_ids,
       COUNT(customer_id) AS customer_id_count,
       COUNT(*) AS tuples_count
FROM customer
```

| | minimum_customer_id<br>integer | maximum_customer_id<br>integer | average_customer_ids<br>numeric | customer_id_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|---|
| 1 | 1 | 599 | 300 | 599 | 599 |

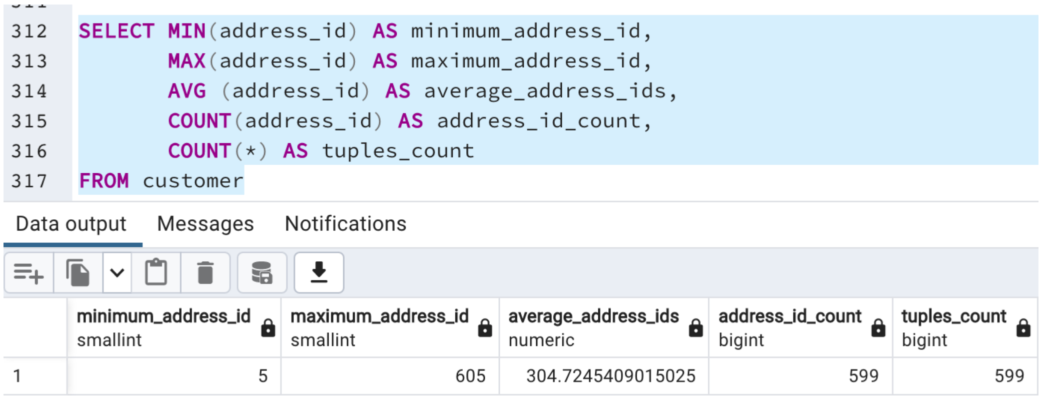
*--storage_id*



```
SELECT MIN(store_id) AS minimum_store_id,
       MAX(store_id) AS maximum_store_id,
       AVG (store_id) AS average_store_ids,
       COUNT(store_id) AS store_id_count,
       COUNT(*) AS tuples_count
FROM customer
```

| | minimum_store_id<br>smallint | maximum_store_id<br>smallint | average_store_ids<br>numeric | store_id_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 1.455759599332220 | 599 | 599 |

*--address_id*

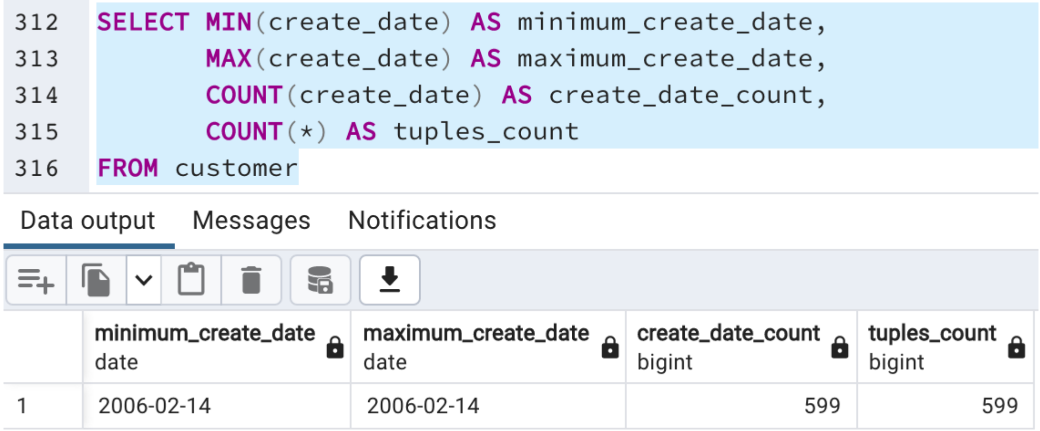


```
SELECT MIN(address_id) AS minimum_address_id,
       MAX(address_id) AS maximum_address_id,
       AVG (address_id) AS average_address_ids,
       COUNT(address_id) AS address_id_count,
       COUNT(*) AS tuples_count
FROM customer
```

| | minimum_address_id<br>smallint | maximum_address_id<br>smallint | average_address_ids<br>numeric | address_id_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|---|
| 1 | 5 | 605 | 304.7245409015025 | 599 | 599 |

*--create_date*

Not possible to get average for dates.

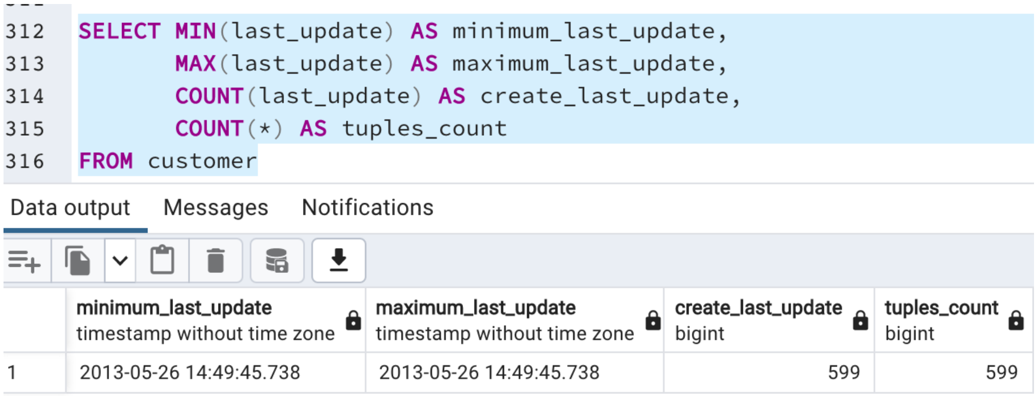


```
SELECT MIN(create_date) AS minimum_create_date,
       MAX(create_date) AS maximum_create_date,
       COUNT(create_date) AS create_date_count,
       COUNT(*) AS tuples_count
FROM customer
```

| | minimum_create_date<br>date | maximum_create_date<br>date | create_date_count<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|
| 1 | 2006-02-14 | 2006-02-14 | 599 | 599 |

*--last_update*

Not possible to get averages fordates.

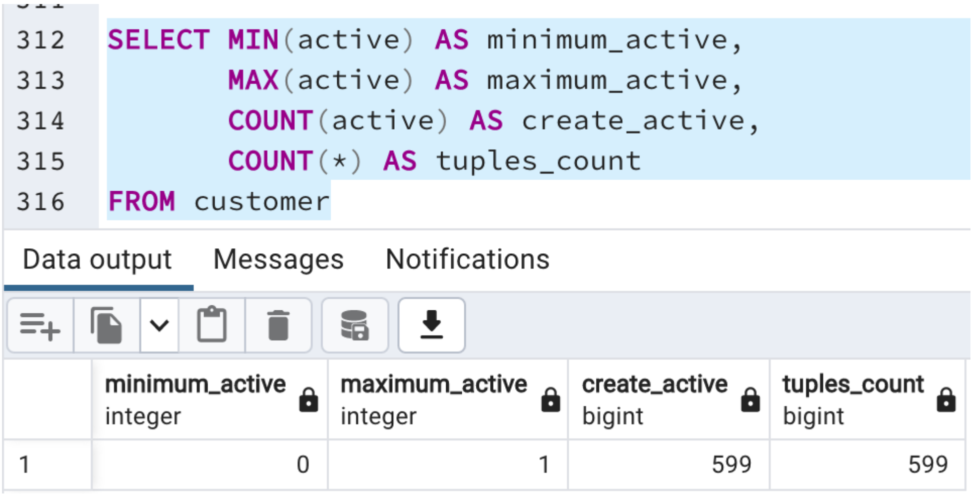


```
SELECT MIN(last_update) AS minimum_last_update,
       MAX(last_update) AS maximum_last_update,
       COUNT(last_update) AS create_last_update,
       COUNT(*) AS tuples_count
FROM customer
```

| | minimum_last_update<br>timestamp without time zone | maximum_last_update<br>timestamp without time zone | create_last_update<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|
| 1 | 2013-05-26 14:49:45.738 | 2013-05-26 14:49:45.738 | 599 | 599 |

*--active*

Averages was not calculated for this columns as does not make sense to get it; only to values.



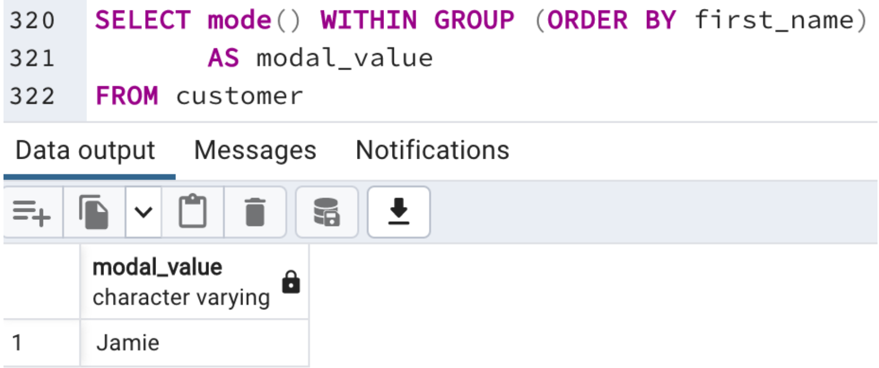
```
SELECT MIN(active) AS minimum_active,
       MAX(active) AS maximum_active,
       COUNT(active) AS create_active,
       COUNT(*) AS tuples_count
FROM customer
```

| | minimum_active<br>integer | maximum_active<br>integer | create_active<br>bigint | tuples_count<br>bigint |
|---|---|---|---|---|
| 1 | 0 | 1 | 599 | 599 |

Query (Syntax) Non-Numeric Columns:

```
SELECT mode() WITHIN GROUP (ORDER BY column_name)
    AS modal_value
FROM tablename
```

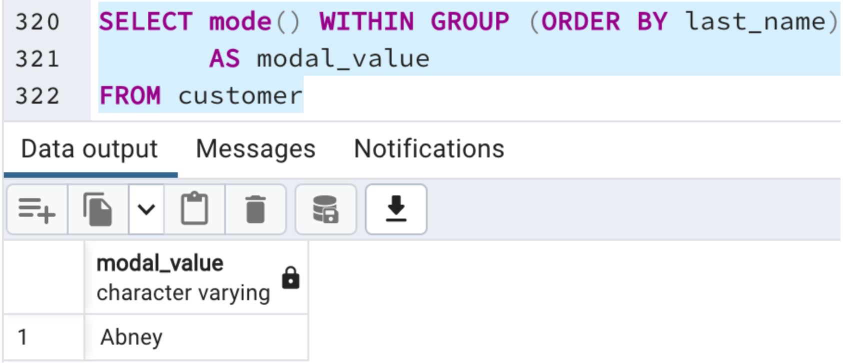
*--first_name*



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

| | modal_value<br>character varying |
|---|---|
| 1 | Jamie |

*--last_name*

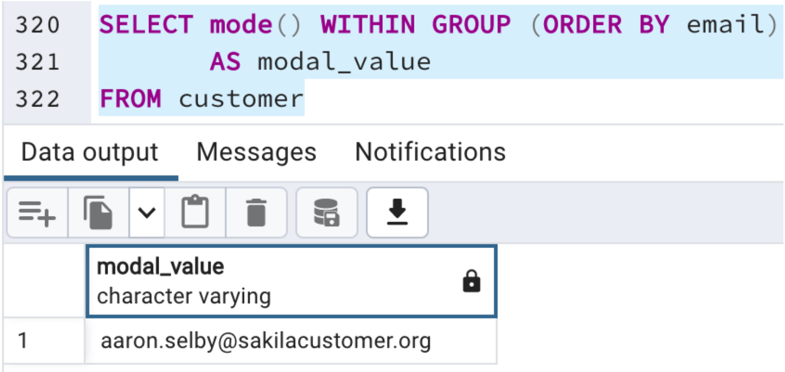


```
SELECT mode() WITHIN GROUP (ORDER BY last_name)
        AS modal_value
FROM customer
```

| | modal_value<br>character varying |
|---|---|
| 1 | Abney |

*--email*

**Please consider there are 599, and 599 different emails. This results is for one of those 599 different emails.**



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

```
  modal_value
  character varying
1 aaron.selby@sakilacustomer.org
```

3. **Which tool (Excel or SQL) is more efficient for data profiling and why? Consider functions, ease of use, and speed.**

In general, SQL is way more effective at providing data types, data integrity, and calculating descriptive statistics of the data. Personally, I find the difference essentially in the capacity that both have in terms of processing data, and how they show the data; SQL's results are succinct, and is considerably quicker coding a query rather than making the data profile from scratch in Excel—though making a data profile in Excel is slower and has less data processing potence, it also is more intuitive once you know what you are doing, and there are more resources around to troubleshoot in excel than in SQL.

*Functions.*

I believe SQL is more effective at providing results; the used functions are basically the same as in Excel, but the one difference would be knowing how to code queries with everything at once in SQL, rather than making a longer process in Excel for each one of the functions—leaving aside the data processing capacity. The downside of SQL is that you must know how to query, you have to have the technical knowledge to do so, otherwise it might be difficult to impossible to retrieve the data the way you want it.

*Ease of Use.*

It's easier to use Excel even though the data integrity analysis represents more time. The issue with Excel is that it has less capacity to process data. On the other hand, once you have (at least the basic) technical knowledge to code queries, it's easier to use SQL, again, because of how succinct the process and the results are

*Speed.*

SQL is by far the fastest way to make a data integrity analysis. It's faster to code a query, even if it's a long and/or complext one (once you have the knowledge).

In conclusion, SQL is better to make a data integrity analysis, but not the easiest.