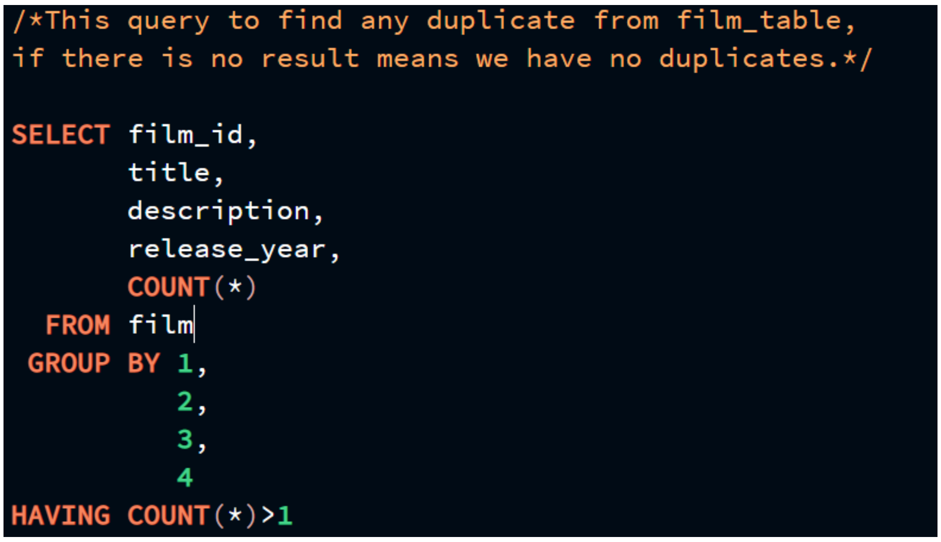
3.5: Filtering Data

ANSWERS

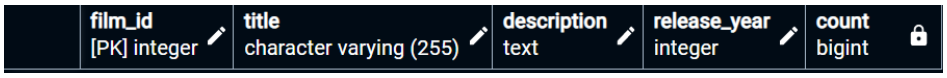
1. Dirty data checking:
   Duplicates:

   Film table



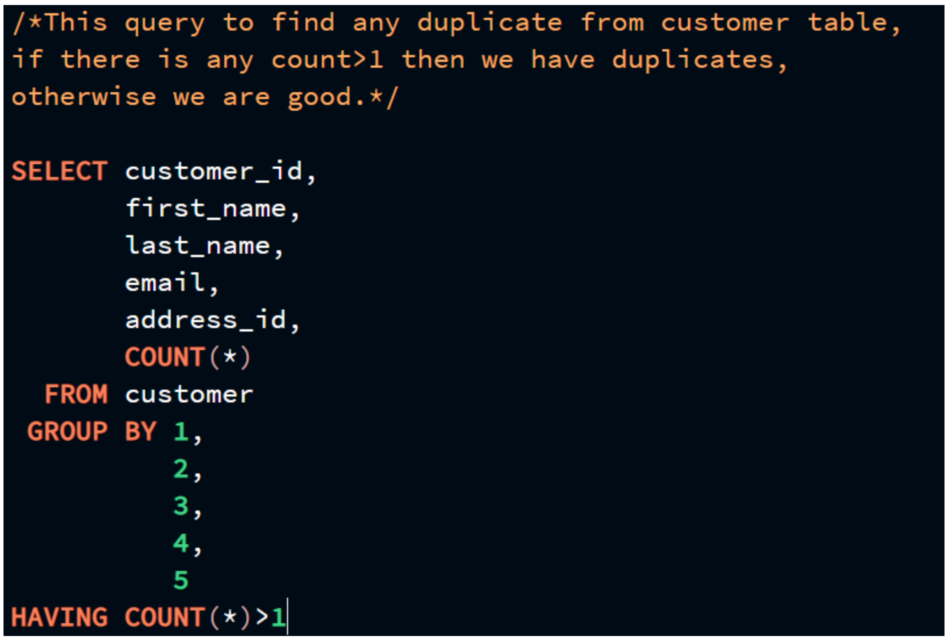
```
/*This query to find any duplicate from film_table,
if there is no result means we have no duplicates.*/

SELECT film_id,
       title,
       description,
       release_year,
       COUNT(*)
  FROM film
 GROUP BY 1,
          2,
          3,
          4
HAVING COUNT(*)>1
```



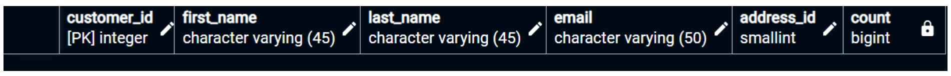
	film_id [PK] integer	title character varying (255)	description text	release_year integer	count bigint

Customer table



```
/*This query to find any duplicate from customer table,
if there is any count>1 then we have duplicates,
otherwise we are good.*/

SELECT customer_id,
       first_name,
       last_name,
       email,
       address_id,
       COUNT(*)
  FROM customer
 GROUP BY 1,
          2,
          3,
          4,
          5
HAVING COUNT(*)>1
```

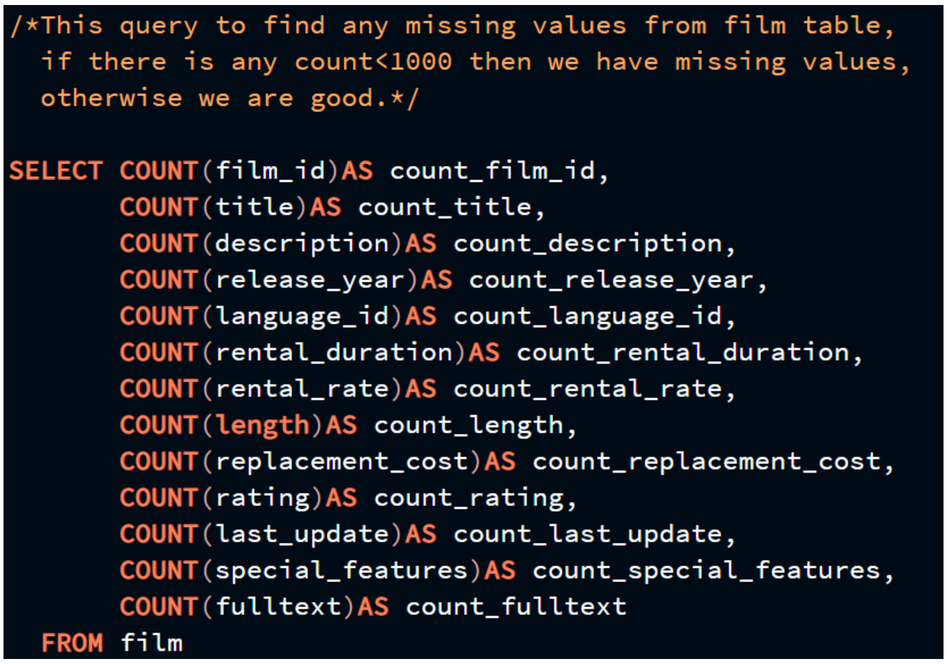


	customer_id [PK] integer	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	count bigint

- No duplicates find from these two tables. If there are some occurs, I would like to use create view method to have all unique records instead of deleting from table method when we met too many duplicates.

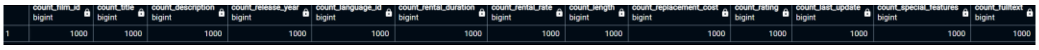
Missing Values

film table



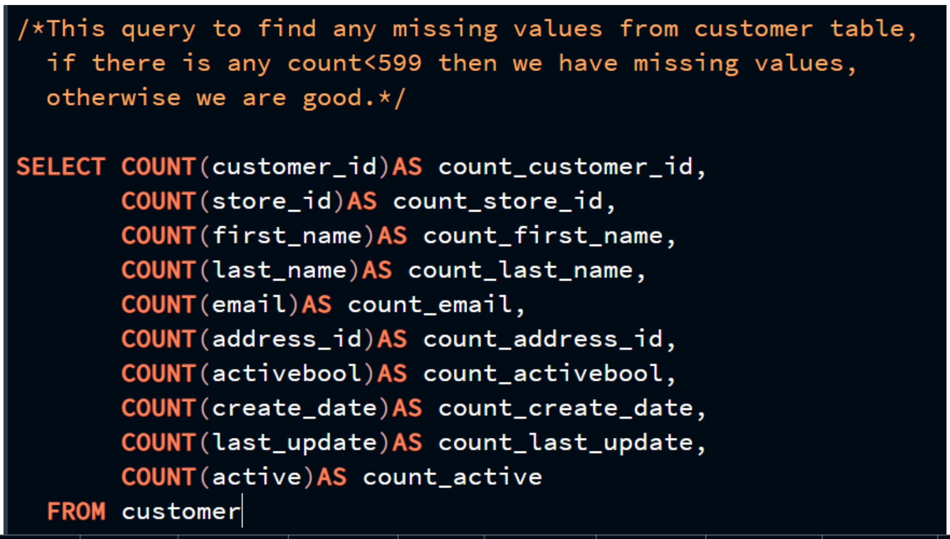
```
/*This query to find any missing values from film table,
  if there is any count<1000 then we have missing values,
  otherwise we are good.*/

SELECT COUNT(film_id)AS count_film_id,
       COUNT(title)AS count_title,
       COUNT(description)AS count_description,
       COUNT(release_year)AS count_release_year,
       COUNT(language_id)AS count_language_id,
       COUNT(rental_duration)AS count_rental_duration,
       COUNT(rental_rate)AS count_rental_rate,
       COUNT(length)AS count_length,
       COUNT(replacement_cost)AS count_replacement_cost,
       COUNT(rating)AS count_rating,
       COUNT(last_update)AS count_last_update,
       COUNT(special_features)AS count_special_features,
       COUNT(fulltext)AS count_fulltext
  FROM film
```



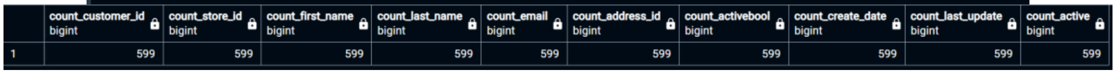
	count_film_id bigint	count_title bigint	count_description bigint	count_release_year bigint	count_language_id bigint	count_rental_duration bigint	count_rental_rate bigint	count_length bigint	count_replacement_cost bigint	count_rating bigint	count_last_update bigint	count_special_features bigint	count_fulltext bigint
1	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000

Customer table



```
/*This query to find any missing values from customer table,
  if there is any count<599 then we have missing values,
  otherwise we are good.*/

SELECT COUNT(customer_id)AS count_customer_id,
       COUNT(store_id)AS count_store_id,
       COUNT(first_name)AS count_first_name,
       COUNT(last_name)AS count_last_name,
       COUNT(email)AS count_email,
       COUNT(address_id)AS count_address_id,
       COUNT(activebool)AS count_activebool,
       COUNT(create_date)AS count_create_date,
       COUNT(last_update)AS count_last_update,
       COUNT(active)AS count_active
  FROM customer
```

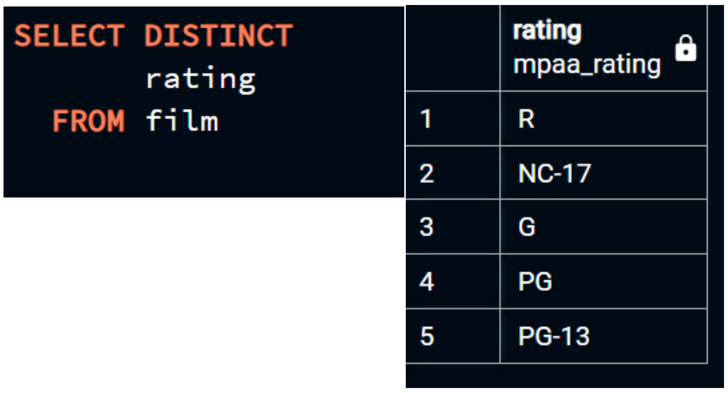


	count_customer_id bigint	count_store_id bigint	count_first_name bigint	count_last_name bigint	count_email bigint	count_address_id bigint	count_activebool bigint	count_create_date bigint	count_last_update bigint	count_active bigint
1	599	599	599	599	599	599	599	599	599	599

- There is not any missing value from these two tables. If there are missing values occurs, I would like to find out the percentage of the missing values. When missing values percentage above 5%, I would leave them as normal, because fill these values will cause the dataset skewed. On the other hand, if the percentage under 5%, I will fill up them with the mean method, and in SQL we can basically write the query like below to update.
  --imputing missing values with the AVG value
  UPDATE film
  SET =AVG(length)--when length column has missing values
  WHERE length IS NULL

  Non-Uniform Data

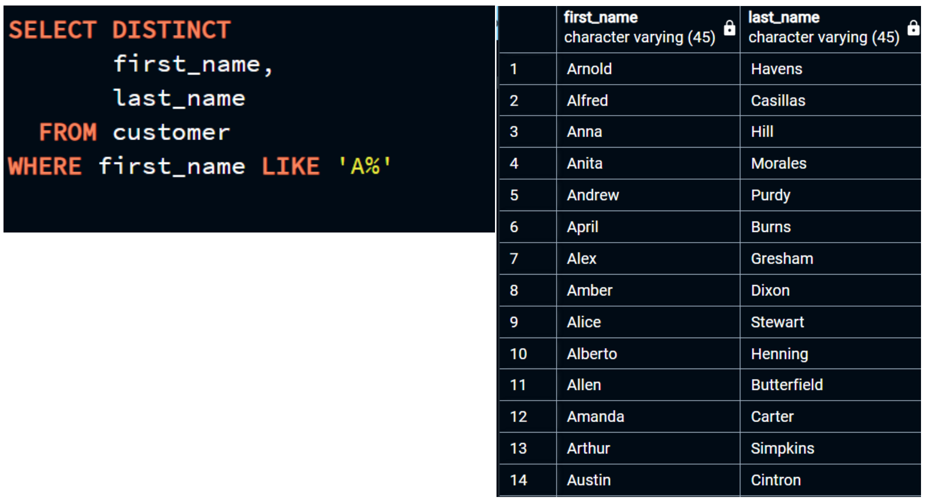
  Film table



```
SELECT DISTINCT
        rating
   FROM film
```

	rating mpaa_rating
1	R
2	NC-17
3	G
4	PG
5	PG-13

Customer table



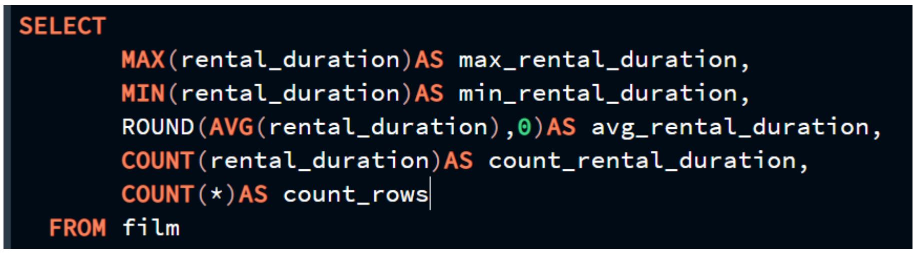
```
SELECT DISTINCT
        first_name,
        last_name
  FROM customer
WHERE first_name LIKE 'A%'
```

	first_name character varying (45)	last_name character varying (45)
1	Arnold	Havens
2	Alfred	Casillas
3	Anna	Hill
4	Anita	Morales
5	Andrew	Purdy
6	April	Burns
7	Alex	Gresham
8	Amber	Dixon
9	Alice	Stewart
10	Alberto	Henning
11	Allen	Butterfield
12	Amanda	Carter
13	Arthur	Simpkins
14	Austin	Cintron

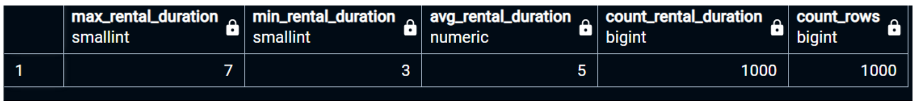
- If there is any non-uniform appears in first_name, last_name without upper-case for the first letter, we can find out using above query randomly by alphabet ‘A-Z’.

2. SUMMARIZE DATA

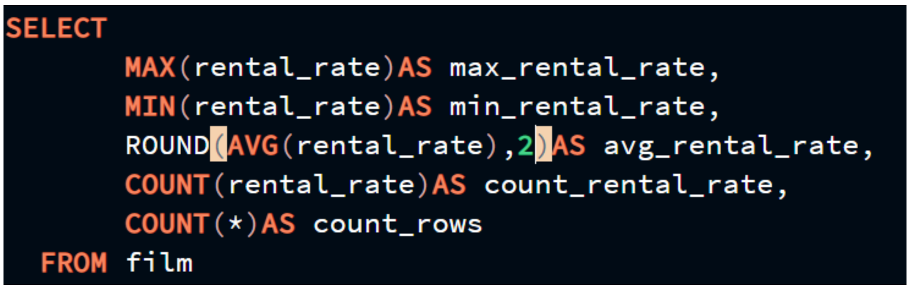
Film table



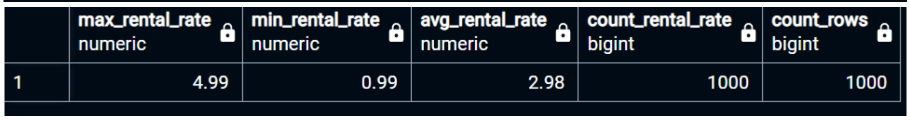
```sql
SELECT
        MAX(rental_duration)AS max_rental_duration,
        MIN(rental_duration)AS min_rental_duration,
        ROUND(AVG(rental_duration),0)AS avg_rental_duration,
        COUNT(rental_duration)AS count_rental_duration,
        COUNT(*)AS count_rows
  FROM film
```



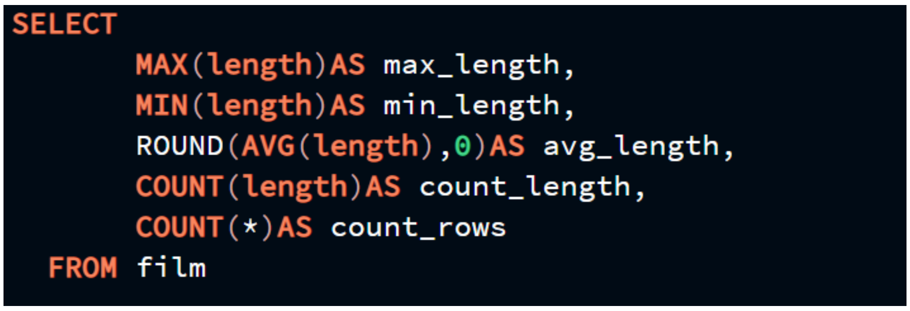
	max_rental_duration smallint	min_rental_duration smallint	avg_rental_duration numeric	count_rental_duration bigint	count_rows bigint
1	7	3	5	1000	1000



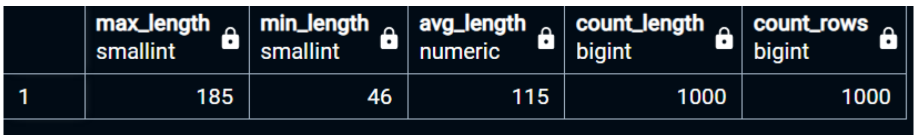
```sql
SELECT
        MAX(rental_rate)AS max_rental_rate,
        MIN(rental_rate)AS min_rental_rate,
        ROUND(AVG(rental_rate),2)AS avg_rental_rate,
        COUNT(rental_rate)AS count_rental_rate,
        COUNT(*)AS count_rows
  FROM film
```



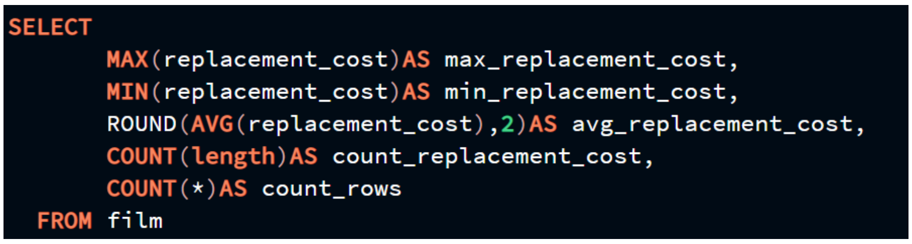
	max_rental_rate numeric	min_rental_rate numeric	avg_rental_rate numeric	count_rental_rate bigint	count_rows bigint
1	4.99	0.99	2.98	1000	1000



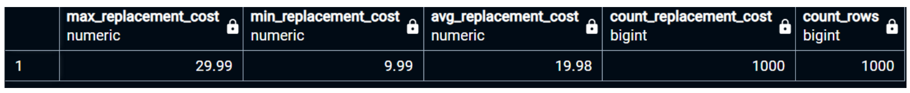
```sql
SELECT
        MAX(length)AS max_length,
        MIN(length)AS min_length,
        ROUND(AVG(length),0)AS avg_length,
        COUNT(length)AS count_length,
        COUNT(*)AS count_rows
  FROM film
```



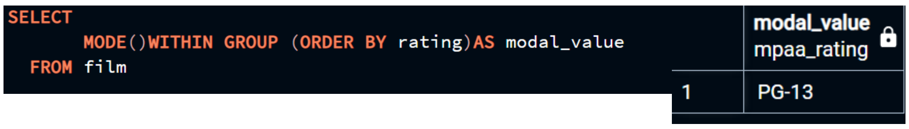
	max_length smallint	min_length smallint	avg_length numeric	count_length bigint	count_rows bigint
1	185	46	115	1000	1000



```sql
SELECT
        MAX(replacement_cost)AS max_replacement_cost,
        MIN(replacement_cost)AS min_replacement_cost,
        ROUND(AVG(replacement_cost),2)AS avg_replacement_cost,
        COUNT(length)AS count_replacement_cost,
        COUNT(*)AS count_rows
  FROM film
```



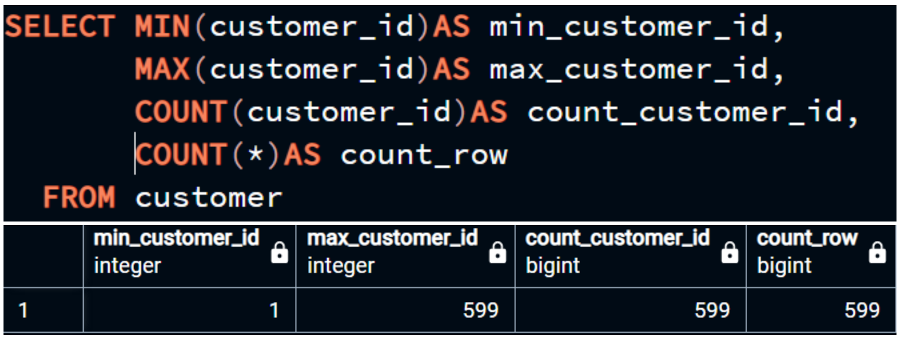
	max_replacement_cost numeric	min_replacement_cost numeric	avg_replacement_cost numeric	count_replacement_cost bigint	count_rows bigint
1	29.99	9.99	19.98	1000	1000



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

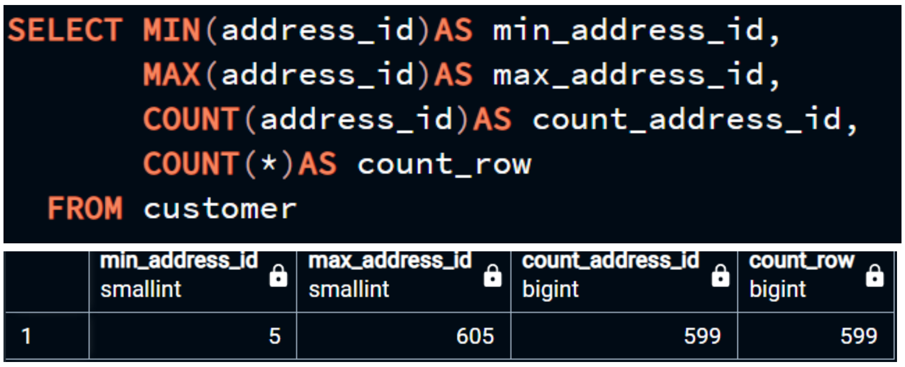
	modal_value mpaa_rating
1	PG-13

Customer table



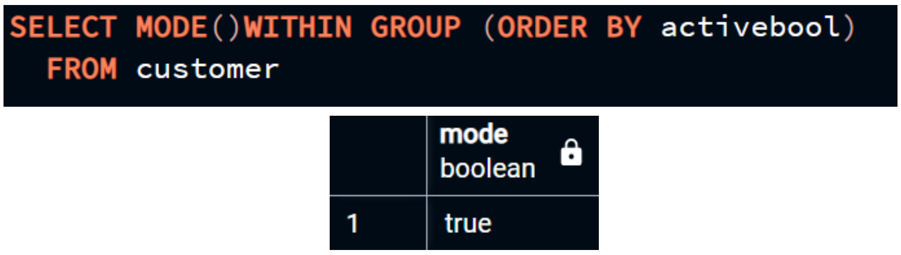
```sql
SELECT MIN(customer_id)AS min_customer_id,
       MAX(customer_id)AS max_customer_id,
       COUNT(customer_id)AS count_customer_id,
       COUNT(*)AS count_row
  FROM customer
```

	min_customer_id integer	max_customer_id integer	count_customer_id bigint	count_row bigint
1	1	599	599	599



```sql
SELECT MIN(address_id)AS min_address_id,
       MAX(address_id)AS max_address_id,
       COUNT(address_id)AS count_address_id,
       COUNT(*)AS count_row
  FROM customer
```

	min_address_id smallint	max_address_id smallint	count_address_id bigint	count_row bigint
1	5	605	599	599



```sql
SELECT MODE()WITHIN GROUP (ORDER BY activebool)
  FROM customer
```

	mode boolean
1	true

3. Using SQL to find the duplicates or missing values is easier to read than using Excel, as we write the query then get the result directly, the query shows the conclusion in the same row. In Excel, we use pivot-table to count the data-grain and then put every single column in, if there are duplicates, the count number would be 1. Generally speaking, these two tools seem work all good. However, dealing with the non—numeric values, Excel is better than SQL, like finding different formats, we only need to filter the columns.