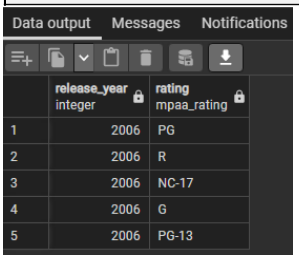
3.6 Summarizing & Cleaning Data in SQL

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. Create a new "Answers 3.6" document and copy-paste your queries into it. Next to each query write 2 to 3 sentences explaining how you would clean the data (even if the data is not dirty).
   - Non-uniform data

```
-- Checking non-uniform data
SELECT DISTINCT release_year,
                rating
FROM film;
```



Data output | Messages | Notifications

	release_year integer	rating mpaa_rating
1	2006	PG
2	2006	R
3	2006	NC-17
4	2006	G
5	2006	PG-13

   - Duplicate data (and also non-uniform data)

```
-- Checking duplicate or non-uniform data
SELECT title,
    release_year,
    language_id,
    rental_duration,
    COUNT(*)
FROM film
GROUP BY title,
    release_year,
    language_id,
    rental_duration
HAVING COUNT(*) > 1;
-- Duplicate or non-uniform data is not found within film table

SELECT store_id,
    first_name,
    last_name,
    email,
    active,
    COUNT(*)
FROM customer
GROUP BY store_id,
    first_name,
    last_name,
    email,
    active
HAVING COUNT(*) > 1;
-- Duplicate or non-uniform data is not found within customer table
```

Non-uniform and duplicate data could be assessed together. The values of non-uniform could then be updated accordingly. While, duplicate values could be either deleted or hidden by creating a virtual table using VIEW. Alternatively, by using GROUP BY or DISTINCT query to select unique records.

- Missing values

```
-- -- Checking missing values
SELECT   COUNT(title),
         COUNT(description),
         COUNT(release_year),
         COUNT(language_id),
         COUNT(rental_duration),
         COUNT(rental_rate),
         COUNT(length),
         COUNT(replacement_cost),
         COUNT(rating),
         COUNT(special_features)
FROM film;
```

Data output | Messages | Notifications

	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_special_features bigint
1	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000

```
-- -- Checking missing values
SELECT   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(active) AS count_active
FROM customer;
```

Data output | Messages | Notifications

	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_active bigint
1	599	599	599	599	599	599	599	599

After identifying the proportion and characteristics of the missing values, there are two options to deal with missing values:

- In case column has high percentage of the missing values (based on my quick research, it is greater than 50%), then ignore the entire column by omitting from the query.
- Imputing the missing value if column shows low percentage of the 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.
   - Numerical columns include:
     - Film table: film_id, rental_duration, rental_rate, length and replacement_cost
     - Customer table: customer_id, store_id and address_id

```
-- Descriptive statistics for numerical columns of film table: Checking min, max and avg values
SELECT   MIN(film_id) AS min_film_id,
         MIN(rental_duration) AS min_rental_duration,
         MIN(rental_rate) AS min_rental_rate,
         MIN(length) AS min_length_film,
         MIN(replacement_cost) AS min_replacement_cost
FROM film;

SELECT   MAX(film_id) AS max_film_id,
         MAX(rental_duration) AS max_rental_duration,
         MAX(rental_rate) AS max_rental_rate,
         MAX(length) AS max_length_film,
         MAX(replacement_cost) AS max_replacement_cost
FROM film;

SELECT   AVG(film_id) AS avg_film_id,
         AVG(rental_duration) AS avg_rental_duration,
         AVG(rental_rate) AS avg_rental_rate,
         AVG(length) AS avg_length_film,
         AVG(replacement_cost) AS avg_replacement_cost
FROM film;
```

min_film_id	min_rental_duration	min_rental_rate	min_length_film	min_replacement_cost
1	3	0.99	46	9.99
max_film_id	max_rental_duration	max_rental_rate	max_length_film	max_replacement_cost
1000	7	4.99	185	29.99
avg_film_id	avg_rental_duration	avg_rental_rate	avg_length_film	avg_replacement_cost
500.5	4.985	2.98	115.272	19.984

```
-- Descriptive statistics for for numerical columns of customer table: Checking min, max and avg values
SELECT MIN(customer_id) AS min_customer_id,
       MIN(store_id ) AS min_store_id,
       MIN(address_id) AS min_address_id
FROM customer;

SELECT MAX(customer_id) AS max_customer_id,
       MAX(store_id ) AS max_store_id,
       MAX(address_id) AS max_address_id
FROM customer;

SELECT AVG(customer_id) AS avg_customer_id,
       AVG(store_id ) AS avg_store_id,
       AVG(address_id) AS avg_address_id
FROM customer;
```

min_customer_id	min_store_id	min_address_id
1	1	5
max_customer_id	max_store_id	max_address_id
599	2	605
avg_customer_id	avg_store_id	avg_address_id
300	1.455	304.724

- Non-numerical columns include:
  - Film table: title, description, release_year, language_id, rating, special_features
  - Customer table: first_name, last_name, email, activebool, active

```
-- Descriptive statistics for non-numerical columns: Checking mode
SELECT   mode() WITHIN GROUP (ORDER BY title) AS title_value,
         mode() WITHIN GROUP (ORDER BY description) AS description_value,
         mode() WITHIN GROUP (ORDER BY release_year) AS release_year_value,
         mode() WITHIN GROUP (ORDER BY rating) AS rating_value,
         mode() WITHIN GROUP (ORDER BY special_features) AS special_features
FROM film;

SELECT   mode() WITHIN GROUP (ORDER BY first_name) AS first_name_value,
         mode() WITHIN GROUP (ORDER BY last_name) AS last_name_value,
         mode() WITHIN GROUP (ORDER BY email) AS email_value,
         mode() WITHIN GROUP (ORDER BY activebool) AS activebool_value
FROM customer;
```

title_value	description_value	release_year_value	rating_value	special_features
Academy Dinosaur	A Action-Packed Character Study of a Astronaut And a Explorer who must Reach a Monkey in A MySQL Convention	2006	PG-13	{Trailers,Commentaries,"Behind the Scenes"}

first_name_value	last_name_value	email_value	activebool_value
Jamie	Abney	aaron.selby@sakilacustomer.org	TRUE

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.**

   It depends on the data size. Excel is sufficient and might also faster to handle small data with couple of tables such as profiling and visualizing data distribution.
   In comparison, SQL is more suitable and faster to handle bigger and complicated data with many interconnected tables. It would also be even faster when SQL could be coupled with automation.