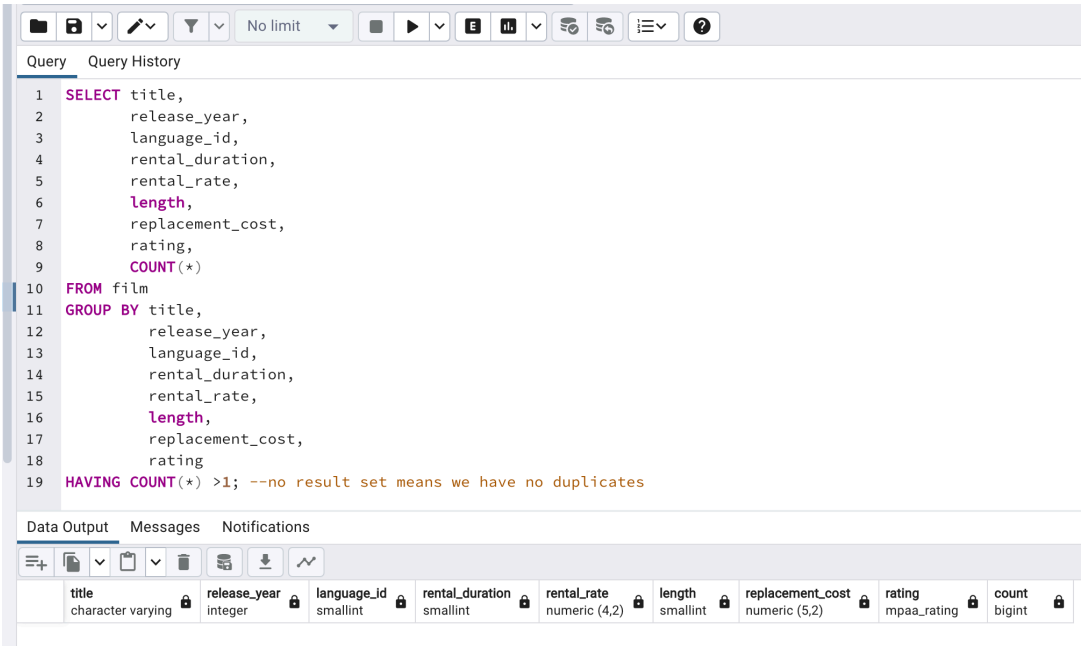
3.6: Summarizing & Cleaning Data in SQL

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).

Answer: checking for duplicate data

For Film Table:

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



For Customer Table:

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

```sql
SELECT store_id,
       first_name,
       last_name,
       email,
       COUNT(*)
FROM customer
GROUP BY store_id,
         first_name,
      last_name,
    email
HAVING COUNT(*) >1; --no result set means we have no duplicates
```

store_id smallint	first_name character varying	last_name character varying	email character varying	count bigint

Total rows: 0 of 0 Query complete 00:00:00.225

I would create View and select only unique records and delete the duplicate records. As a junior data analyst, I wouldn't be responsible for deleting the records.

Checking for non-uniform data:

For Film Table:

```sql
SELECT DISTINCT
       release_year
FROM film;
```

	release_year integer
1	2006

```sql
SELECT DISTINCT
    release_year FROM film;
```

Rockbuster/postgres@PostgreSQL 15

No limit

Query Query History

```
SELECT DISTINCT
       rating
FROM film;
```

Data Output Messages Notifications

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

```
SELECT DISTINCT
       rating
FROM film;
```

Rockbuster/postgres@Pos

Query Query History

```
SELECT DISTINCT
       rental_rate
FROM film;
```

Data Output Messages Noti

	rental_rate numeric (4,2)
1	2.99
2	4.99
3	0.99

```
SELECT DISTINCT
       rental_rate
FROM film;
```

For customer Table:

Query Query History

```
SELECT DISTINCT
       store_id
FROM customer;
```

Data Output Messages Notif

	store_id smallint
1	1
2	2

```
SELECT DISTINCT
       store_id
FROM customer;
```

Query Query History

```
SELECT DISTINCT
       active
FROM customer;
```

Data Output Messages

	active integer
1	0
2	1

```
SELECT DISTINCT
       active
FROM customer;
```

After checking with these queries, we can say that there is no non-uniform data. If there is any non-uniform data then we could perform UPDATE, SET; WHERE functions to adjust entries.

Checking for missing data:

For Film Table:

Query Query History

```
SELECT *
FROM film
WHERE rating IS NULL
```

Data Output Messages Notifications

film_id [PK] integer	title character varying	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_upda timestam

Query Query History

```
SELECT *
FROM film
WHERE rental_rate IS NULL
```

Data Output Messages Notifications

film_id [PK] integer	title character varying	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_upda timestam

No limit

Query Query History

```
SELECT *
FROM film
WHERE film_id   IS NULL
```

Data Output Messages Notifications

film_id [PK] integer	title character varying	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_upda timestam

Query Query History

```
SELECT *
FROM film
WHERE title IS NULL
```

Data Output Messages Notifications

film_id [PK] integer	title character varying	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_upd timestan

For Customer Table:

Query Query History

```
SELECT *
FROM customer
WHERE store_id IS NULL
```

Data Output Messages Notifications

customer_id [PK] integer	store_id smallint	first_name character varying	last_name character varying	email character varying	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone	active integer

Query Query History

```
SELECT *
FROM customer
WHERE first_name IS NULL
```

Data Output Messages Notifications

customer_id [PK] integer	store_id smallint	first_name character varying	last_name character varying	email character varying	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone	active integer

Query Query History

```
SELECT *
FROM customer
WHERE email IS NULL
```

Data Output Messages Notifications

customer_id [PK] integer	store_id smallint	first_name character varying	last_name character varying	email character varying	address_id smallint	activebool boolean	create_date date	last_update timestamp without time zone	active integer

I did not find any missing data in any column. If I would find any missing data then I would perform these two options:

1. Impute missing values with average value by calculating average function and then would perform UPDATE function to set the average value.
2. Ignore columns with high number of 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.

```
SELECT film_id, MAX(rental_duration) AS
maximum_rental_duration, MIN(rental_duration) AS
minimum_rental_duration,  AVG(rental_duration) AS
average_rental_duration, MAX(rental_rate) AS
maximum_rental_rate,   MIN(rental_rate) AS minimum_rental_rate,
AVG(rental_rate) AS average_rental_rate, MAX(length) AS
maximum_movie_length,  MIN(length) AS minimum_movie_length,
AVG(length) AS average_movie_length, MAX(replacement_cost)
AS maximum_replacement_cost, MIN(replacement_cost) AS
minimum_replacement_cost, AVG(replacement_cost) AS
average_replacement_cost

FROM film GROUP BY film_id, rental_duration, rental_rate, length,
replacement_cost
```

Data Output Messages Notifications

	maximum_rental_duration smallint	minimum_rental_duration smallint	average_rental_duration numeric	count_rental_duration bigint	maximum_rental_rate numeric	minimum_rental_rate numeric	average_rental_rate numeric
1	7	3	4.9850000000000000	1000	4.99	0.99	2.9800000000000000

Data Output Messages Notifications

	count_rental_rate bigint	maximum_movie_length smallint	minimum_movie_length smallint	average_movie_length numeric	count_length bigint	maximum_replacement_cost numeric	minimum_replacement_cost numeric
1	1000	185	46	115.2720000000000000	1000	29.99	9.99

Data Output Messages Notifications

	h	average_movie_length numeric	count_length bigint	maximum_replacement_cost numeric	minimum_replacement_cost numeric	average_replacement_cost numeric	count_replacement_cost bigint	count_rows bigint
1	46	115.2720000000000000	1000	29.99	9.99	19.9840000000000000	1000	1000

Query Query History

```sql
-- Mode for non-numeric columns.
SELECT  MODE() WITHIN GROUP (ORDER BY title) AS title_mode_value,
        MODE() WITHIN GROUP (ORDER BY description) AS description_mode_value,
        MODE() WITHIN GROUP (ORDER BY last_update) AS last_update_mode_value,
        MODE() WITHIN GROUP (ORDER BY release_year) AS release_year_mode_value,
        MODE() WITHIN GROUP (ORDER BY rating) AS rating_mode_value,
        MODE() WITHIN GROUP (ORDER BY fulltext) AS full_text_mode_value
        FROM film
```

Data Output Messages Notifications

	title_mode_value character varying	description_mode_value text	last_update_mode_value timestamp without time zone	release_year_mode_value integer
1	Academy Dinosaur	A Action-Packed Character Study of a Astronaut And a Explorer who must Reach a Monkey in A MySQL Convent...	2013-05-26 14:50:58.951	2006

Data Output Messages Notifications

		last_update_mode_value timestamp without time zone	release_year_mode_value integer	rating_mode_value mpaa_rating	full_text_mode_value tsvector
1	n A MySQL Convent...	2013-05-26 14:50:58.951	2006	PG-13	'baloon':19 'confront':14 'documentari':5 'feminist':8,11,16 'mile':2 'must':13 'spi':1 'thri...

Query Query History

```sql
-- Mode for non-numeric columns.
SELECT  MODE() WITHIN GROUP (ORDER BY first_name) AS firstname_mode_value,
        MODE() WITHIN GROUP (ORDER BY last_name) AS lastname_mode_value,
        MODE() WITHIN GROUP (ORDER BY last_update) AS last_update_mode_value,
        MODE() WITHIN GROUP (ORDER BY create_date) AS createdate_mode_value,
        MODE() WITHIN GROUP (ORDER BY active) AS active_mode_value

        FROM customer
```

Data Output Messages Notifications

	firstname_mode_value character varying	lastname_mode_value character varying	last_update_mode_value timestamp without time zone	createdate_mode_value date	active_mode_value integer
1	Jamie	Abney	2013-05-26 14:49:45.738	2006-02-14	1

3. Reflect on your work: At this stage, we can not say much more about both the languages. They both have their own pros and cons. Some Excel to SQL points are there :

- SQL is not actually harder than Excel, it's just a little different. Once we learned the language, acts much more like "talking" as opposed to simply managing or manipulating data given its syntax.
- It can take minutes in SQL to do what it takes nearly an hour to do in Excel.
- Excel is a great program for simplicity and flexibility. SQL databases are excellent choices for storage, manipulation and analysis of large amount of data.

- Excel is extremely intuitive. We can see the data and calculations at a glance. The formulas are simple for anyone to understand. It's inexpensive and there's a free version.
- Excel is prone to human error. It's easy to type over a critical formula.
- Data is much safer in SQL data storage than in Excel, since it's more difficult for a user to delete data by mistake.
- Collaboration isn't a strong point for Excel, since it's easy for other users to delete critical information or formulas.
- Excel can technically handle one million rows, but that's before the pivot tables, multiple tabs, and functions you're probably using.
- SQL is much faster than Excel. SQL also separates analysis from data.