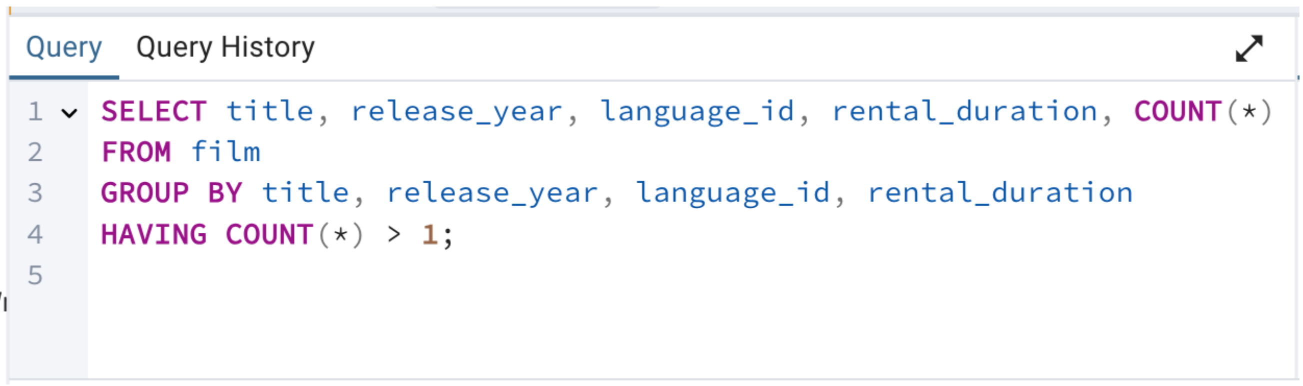
Answers 3.6

Step1)

Checking for Duplicate Data

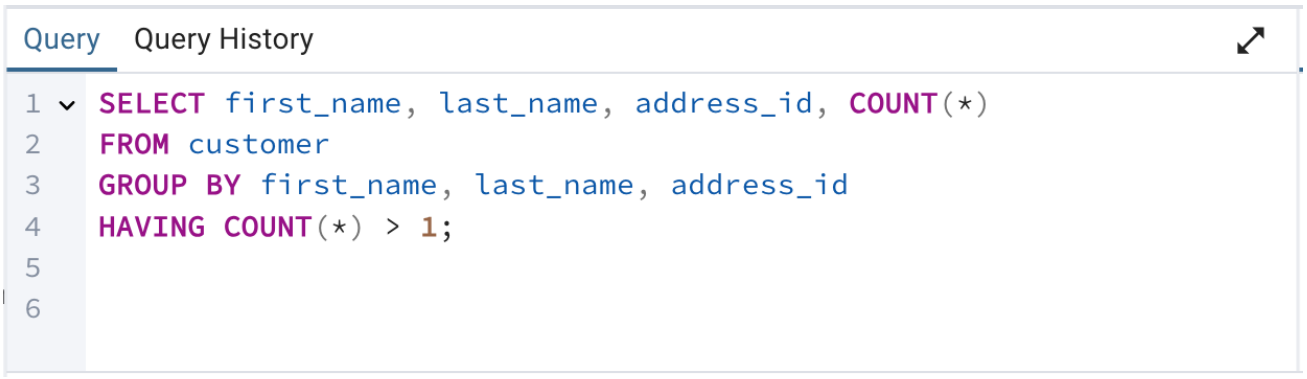
Film table



```sql
SELECT title, release_year, language_id, rental_duration, COUNT(*)
FROM film
GROUP BY title, release_year, language_id, rental_duration
HAVING COUNT(*) > 1;
```

The query to check for duplicate records in the film table returned 0 rows, indicating that there are **no duplicate records in the dataset**. Even though the data is not dirty in this case, if duplicates had been found, the standard approach would be to either remove them by **deleting duplicate rows** or **creating a view to isolate only unique records**. Creating a view is often preferred because it does not alter the original data and ensures safety for future analyses.

Customer Table

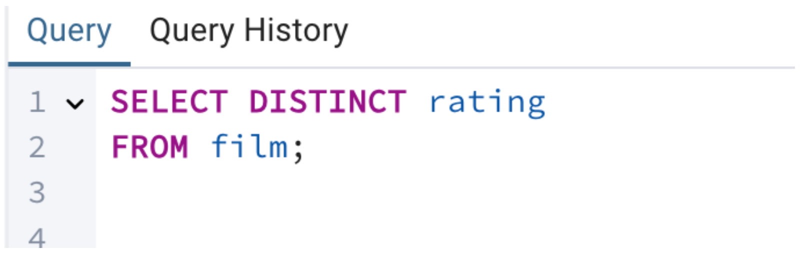


```sql
SELECT first_name, last_name, address_id, COUNT(*)
FROM customer
GROUP BY first_name, last_name, address_id
HAVING COUNT(*) > 1;
```

The query to check for duplicate records in the customer table returned 0 rows, indicating that there are no duplicate records in the dataset. Although no duplicates exist, if duplicates were present, I would clean the data by either deleting the duplicate rows or creating a view to isolate only unique records based on first_name, last_name, and address_id. Using a view is the safer approach, as it preserves the original data and ensures that only unique records are used for further analysis.

Non-Uniform Data

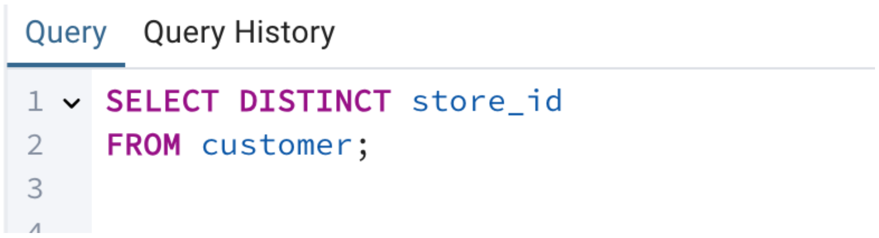
Film table



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

The query output indicates that the rating column in the film table has the following unique values: PG-13, NC-17, R, G, and PG. These values appear to be consistent and fall within the expected valid set, and, the rating column does not have any non-uniform or invalid data.

Customer Table

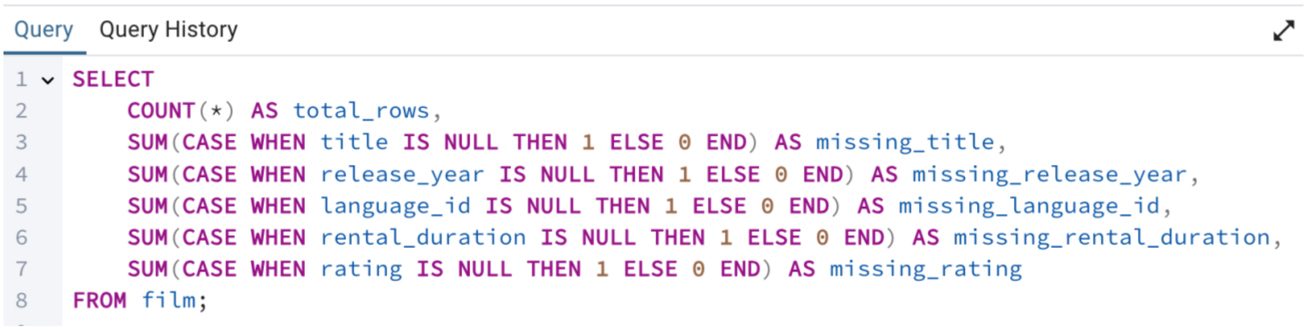


```sql
SELECT DISTINCT store_id
FROM customer;
```

The query to check for non-uniform data in the store_id column of the customer table returned only two distinct values: 1 and 2. This result indicates that the data in the store_id column is consistent and does not contain any non-uniform values.

Missing Values

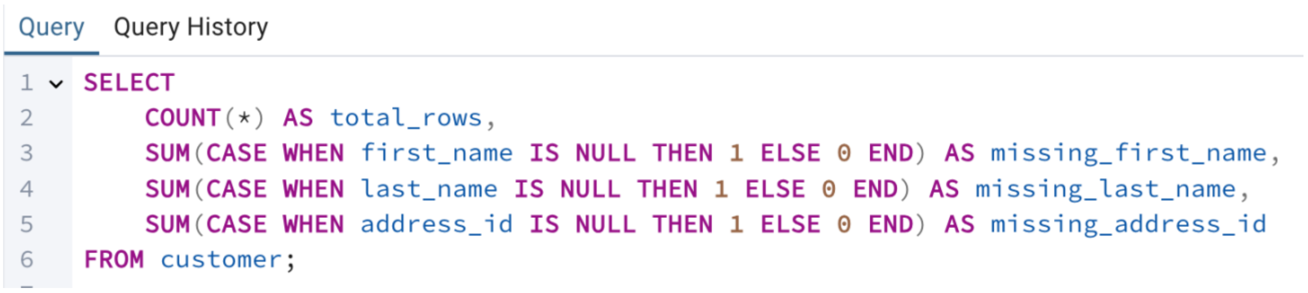
Film table



```sql
SELECT
    COUNT(*) AS total_rows,
    SUM(CASE WHEN title IS NULL THEN 1 ELSE 0 END) AS missing_title,
    SUM(CASE WHEN release_year IS NULL THEN 1 ELSE 0 END) AS missing_release_year,
    SUM(CASE WHEN language_id IS NULL THEN 1 ELSE 0 END) AS missing_language_id,
    SUM(CASE WHEN rental_duration IS NULL THEN 1 ELSE 0 END) AS missing_rental_duration,
    SUM(CASE WHEN rating IS NULL THEN 1 ELSE 0 END) AS missing_rating
FROM film;
```

The query results show that there are no missing values in the key columns of the film table (title, release_year, language_id, rental_duration, rating). If missing values had been present, I would clean the data by either imputing them with logical defaults (replacing missing rental_duration with the average duration) or excluding rows with excessive missing values from the analysis. Since no missing values exist, no cleaning is required for this table.

Customer Table



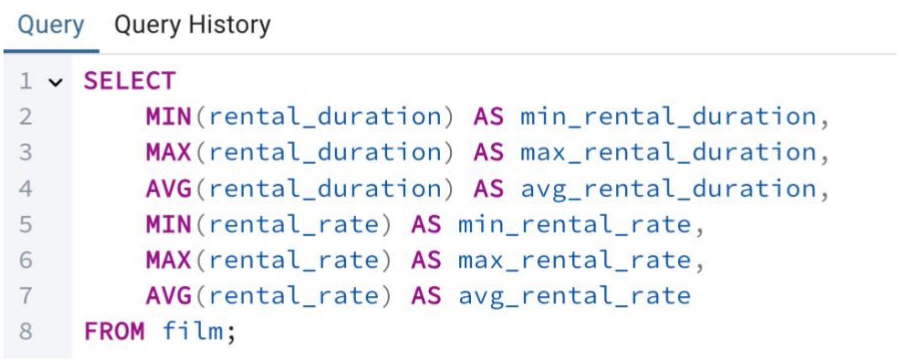
```
SELECT
    COUNT(*) AS total_rows,
    SUM(CASE WHEN first_name IS NULL THEN 1 ELSE 0 END) AS missing_first_name,
    SUM(CASE WHEN last_name IS NULL THEN 1 ELSE 0 END) AS missing_last_name,
    SUM(CASE WHEN address_id IS NULL THEN 1 ELSE 0 END) AS missing_address_id
FROM customer;
```

The query results indicate that there are no missing values in the first_name, last_name, or address_id columns of the customer table. If missing values were present, I would clean the data by imputing logical defaults (e.g., assigning placeholder values for names or a default address ID) or excluding rows with excessive missing data from analysis. Since no missing data exists, no further cleaning is required for this table.

Step 2)

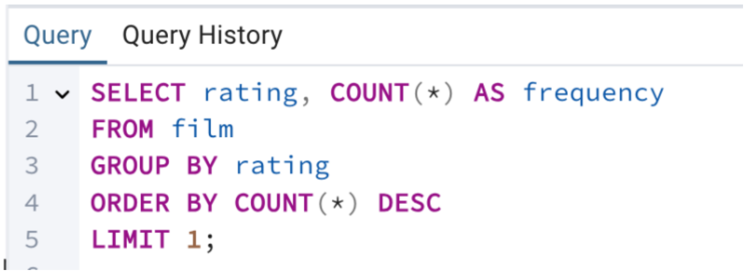
Film Table

-Numerical Columns: columns to summarize: rental_duration, rental_rate.



```
SELECT
    MIN(rental_duration) AS min_rental_duration,
    MAX(rental_duration) AS max_rental_duration,
    AVG(rental_duration) AS avg_rental_duration,
    MIN(rental_rate) AS min_rental_rate,
    MAX(rental_rate) AS max_rental_rate,
    AVG(rental_rate) AS avg_rental_rate
FROM film;
```

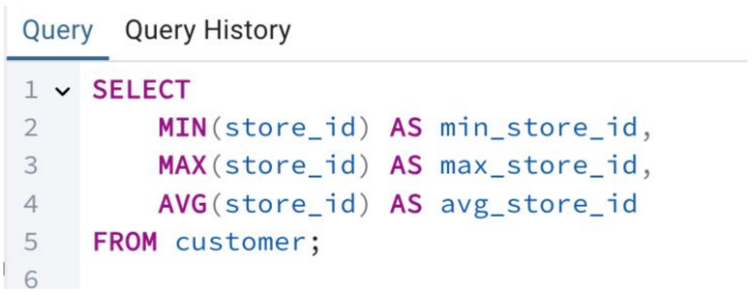
-Non-Numerical Columns: column to summarize is rating



```
SELECT rating, COUNT(*) AS frequency
FROM film
GROUP BY rating
ORDER BY COUNT(*) DESC
LIMIT 1;
```

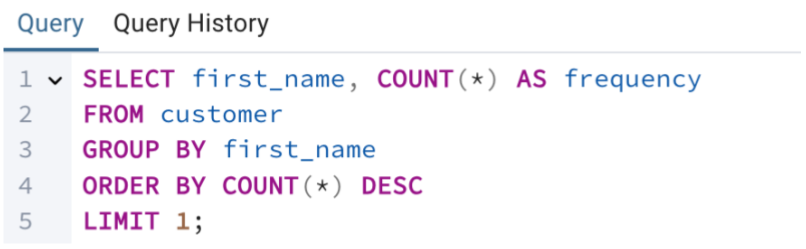
Customer Table

-Numerical Columns: column to summarize is store_id.



```
SELECT
    MIN(store_id) AS min_store_id,
    MAX(store_id) AS max_store_id,
    AVG(store_id) AS avg_store_id
FROM customer;
```

-Non-Numerical Columns: column to summarize is first_name.



```
SELECT first_name, COUNT(*) AS frequency
FROM customer
GROUP BY first_name
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Summary of descriptive statistics

Table	Column	Metric/Mode
Film	Rental Duration and Rate	Min: 3, Max: 7, Avg: 4,985 / Min: 0,99, Max: 4,99, Avg: 2.98
Film	Rating	Mode: PG-13 (223 occurrences)
Customer	Store ID	Min: 1, Max: 2, Avg: 1,456
Customer	First Name	Mode: Leslie (2 occurrences)

Step 3)

When comparing Excel and SQL for data profiling, both tools offer distinct advantages, but SQL proves to be more effective for large datasets and complex analyses. SQL excels in speed and scalability, as it is designed to handle millions of rows of data efficiently, whereas Excel becomes slow and less reliable with larger datasets. SQL offers precise, reusable queries and functions, which allow for automated and systematic data profiling. On the other hand, Excel provides a user-friendly interface with visualization capabilities, making it ideal for smaller analyses or when graphical outputs are needed. However, for structured data, faster execution, and complex operations, SQL is the superior tool, especially when working with relational databases.