Career Foundry

Data Analytics Immersion

A3.E6

Kendra Jackson

<u>Step 1:</u>

Duplicate Values:

Film Table	*Customer Table*
Query: SELECT film_id, title, release_year, language_id, COUNT(*) FROM film GROUP BY film_id, title, release_year, language_id HAVING COUNT (*) > 1; -- Returns all duplicate records where count greater than 1 film_id [PK] integer | title character varying (255) I used these columns as film ID should be unique, I included title, release_year, and language_id in case there were typos in the title name for sequels or films releases in various languages were recorded with different titles multiple times. **There are no duplicate records returned with this query.**	Query: SELECT customer_id, first_name, last_name, email, address_id,COUNT(*) FROM film GROUP BY customer_id, first_name, last_name, email, address_id HAVING COUNT (*) > 1; -- Returns all duplicate records where count greater than 1 customer_id [PK] integer | first_name character varying (45) | last_name character varying I used these columns as customer_id should be a unique identifier as well as email. First_name, last_name and address_id are all entries that when separate may be used by multiple people but used in conjunction, should create unique entries. **There are no duplicate records returned with this query.**

Cleaning approach for duplicates:

If either query had returned dupliactes, it is best practice to use CREATE VIEW rather than deleting the records. A view allows you to select only unique records.

The command to create a view is:

```
CREATE VIEW viewname AS
SELECT col1, col 2, col3…,
FROM tablename
GROUP BY col1, col2, col3,… ;
```

If permissions don't allow deleting of records or creating a view, the SELECT DISTINCT command can be used to select unique records.

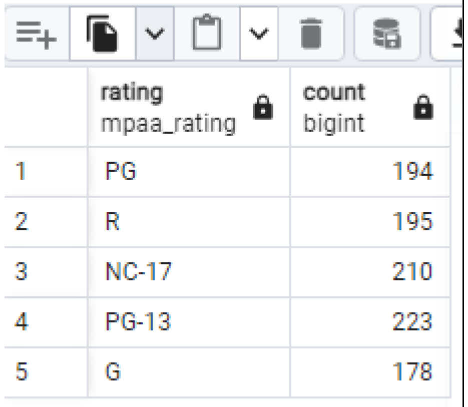
```
SELECT DISTINCT col1, col2, col3
FROM tablename:
```

Non- Uniform Data

Film Table

Query:

```
SELECT rating, COUNT(*)
FROM film
GROUP BY rating;
```



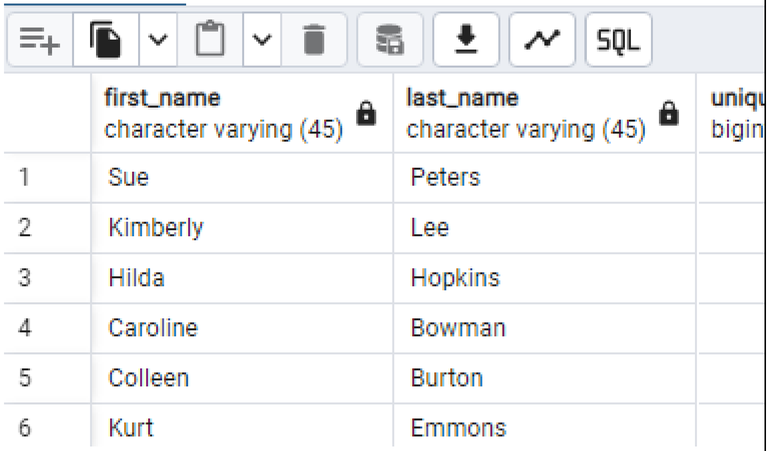
	rating mpaa_rating	count bigint
1	PG	194
2	R	195
3	NC-17	210
4	PG-13	223
5	G	178

This shows you the records for a singular column, this can be completed for all non-uniform

Customer Table

Query:

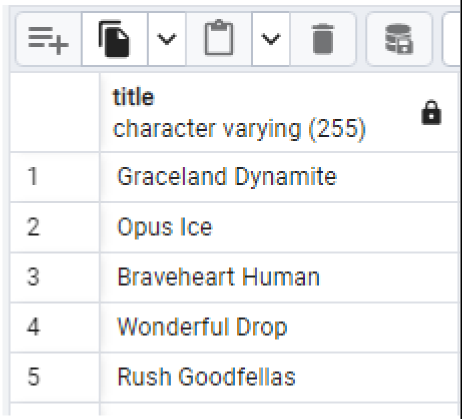
```
SELECT first_name, last_name, COUNT(*) AS
unique_customer_count
FROM customer
GROUP BY first_name, last_name
ORDER BY unique_customer_count DESC;
```



	first_name character varying (45)	last_name character varying (45)	uniq bigin
1	Sue	Peters	
2	Kimberly	Lee	
3	Hilda	Hopkins	
4	Caroline	Bowman	
5	Colleen	Burton	
6	Kurt	Emmons	

columns. If inconsistencies in formatting were found, an update would be needed.

```
SELECT title, COUNT(*) AS
title_count
FROM film
GROUP BY title
ORDER BY title_count DESC;
```



	title character varying (255)
1	Graceland Dynamite
2	Opus Ice
3	Braveheart Human
4	Wonderful Drop
5	Rush Goodfellas

This would both show duplicate records and formatting issues.

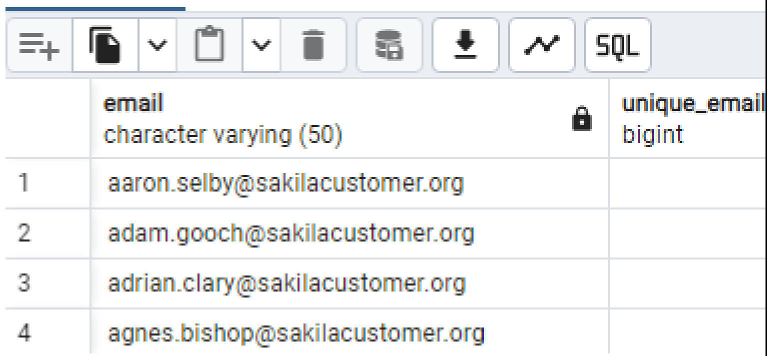
There are no duplicate records returned with this query.

Queries for invalid data types could also be checked or data that would not make sense (like a rental time of 0)

```
SELECT * FROM film WHERE
length = 0
```

Shows the count of unique names, the count column is ordered descending as if there was a duplicate it would be at the top. This also would show formatting issues.

```
SELECT email, COUNT(*) AS unique_emails
FROM customer
GROUP BY email
ORDER BY unique_emails DESC;
```



	email character varying (50)	unique_email bigint
1	aaron.selby@sakilacustomer.org	
2	adam.gooch@sakilacustomer.org	
3	adrian.clary@sakilacustomer.org	
4	agnes.bishop@sakilacustomer.org	

Shows email count. If an email had a duplicate, it would appear at the top. This would also show formatting issues.

There are no duplicate records returned with this query.

Cleaning approach for non-uniform data

If either query had returned formatting inconsistencies, the data could be updated.

```
UPDATE film
SET col1 = ‘value’
WHERE col1 IN (‘non-uniform data found’,’…’);
```

Missing Data

Film Table	*Customer Table*
Query: SELECT * FROM film WHERE film_id IS NULL OR title IS NULL OR description IS NULL OR release_year IS NULL OR language_id IS NULL OR rental_duration IS NULL OR rental_rate IS NULL OR length IS NULL OR replacement_cost IS NULL OR rating IS NULL OR last_update IS NULL OR special_features IS NULL OR fulltext IS NULL; **No empty records found.**	Query: SELECT * FROM customer WHERE customer_id IS NULL OR store_id IS NULL OR first_name IS NULL OR last_name IS NULL OR email IS NULL OR address_id IS NULL OR activebool IS NULL OR create_date IS NULL OR last_update IS NULL OR active IS NULL; **No empty records found.**
Cleaning missing data 1) When a column has too many missing values, and you cannot find the values from the data source, it is best to leave the date alone and not use it rather than deleting or replacing it as this can skew results. SELECT col1, col2, col4 FROM tablename – column 3 ignored in select because of too many missing values	

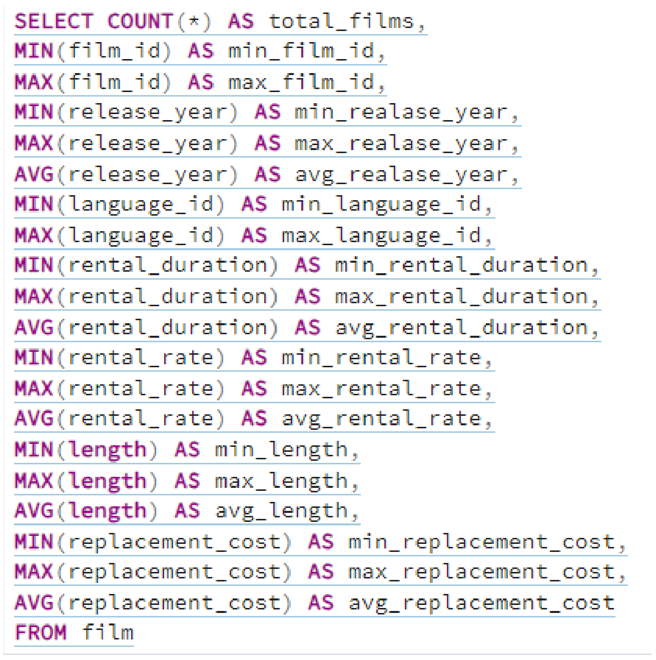
2) If there are a few missing values, you can update them with an imputed average

UPDATE tablemane SET = AVG(col1) WHERE col1 IS NULL

<u>Step 2:</u>

Film Table

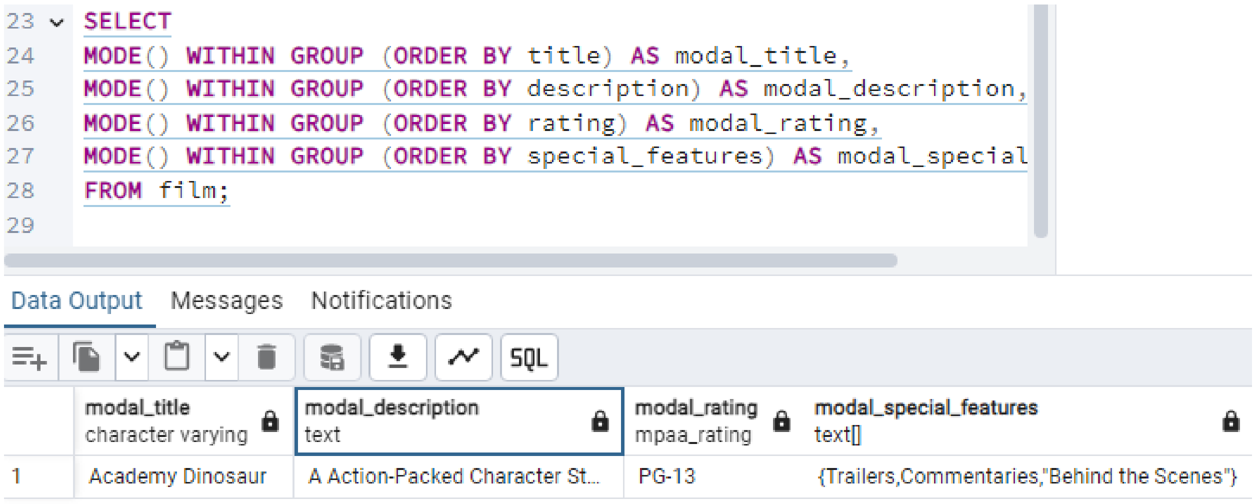
Numerical Values:



```
SELECT COUNT(*) AS total_films,
MIN(film_id) AS min_film_id,
MAX(film_id) AS max_film_id,
MIN(release_year) AS min_realase_year,
MAX(release_year) AS max_realase_year,
AVG(release_year) AS avg_realase_year,
MIN(language_id) AS min_language_id,
MAX(language_id) AS max_language_id,
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,
MIN(length) AS min_length,
MAX(length) AS max_length,
AVG(length) AS avg_length,
MIN(replacement_cost) AS min_replacement_cost,
MAX(replacement_cost) AS max_replacement_cost,
AVG(replacement_cost) AS avg_replacement_cost
FROM film
```

total_films	min_film_ic	max_film_i	min_realas	max_realas	avg_realase	min_langua
1000	1	1000	2006	2006	2006	1

Non Numerical Values

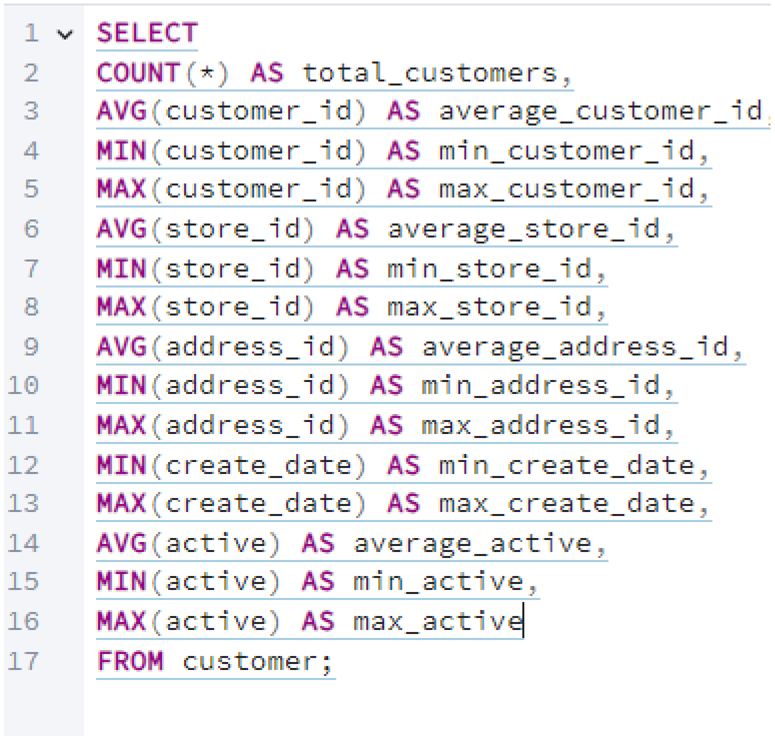


```
SELECT
MODE() WITHIN GROUP (ORDER BY title) AS modal_title,
MODE() WITHIN GROUP (ORDER BY description) AS modal_description,
MODE() WITHIN GROUP (ORDER BY rating) AS modal_rating,
MODE() WITHIN GROUP (ORDER BY special_features) AS modal_special
FROM film;
```

	modal_title character varying	modal_description text	modal_rating mpaa_rating	modal_special_features text[]
1	Academy Dinosaur	A Action-Packed Character St...	PG-13	{Trailers,Commentaries,"Behind the Scenes"}

Customer Table

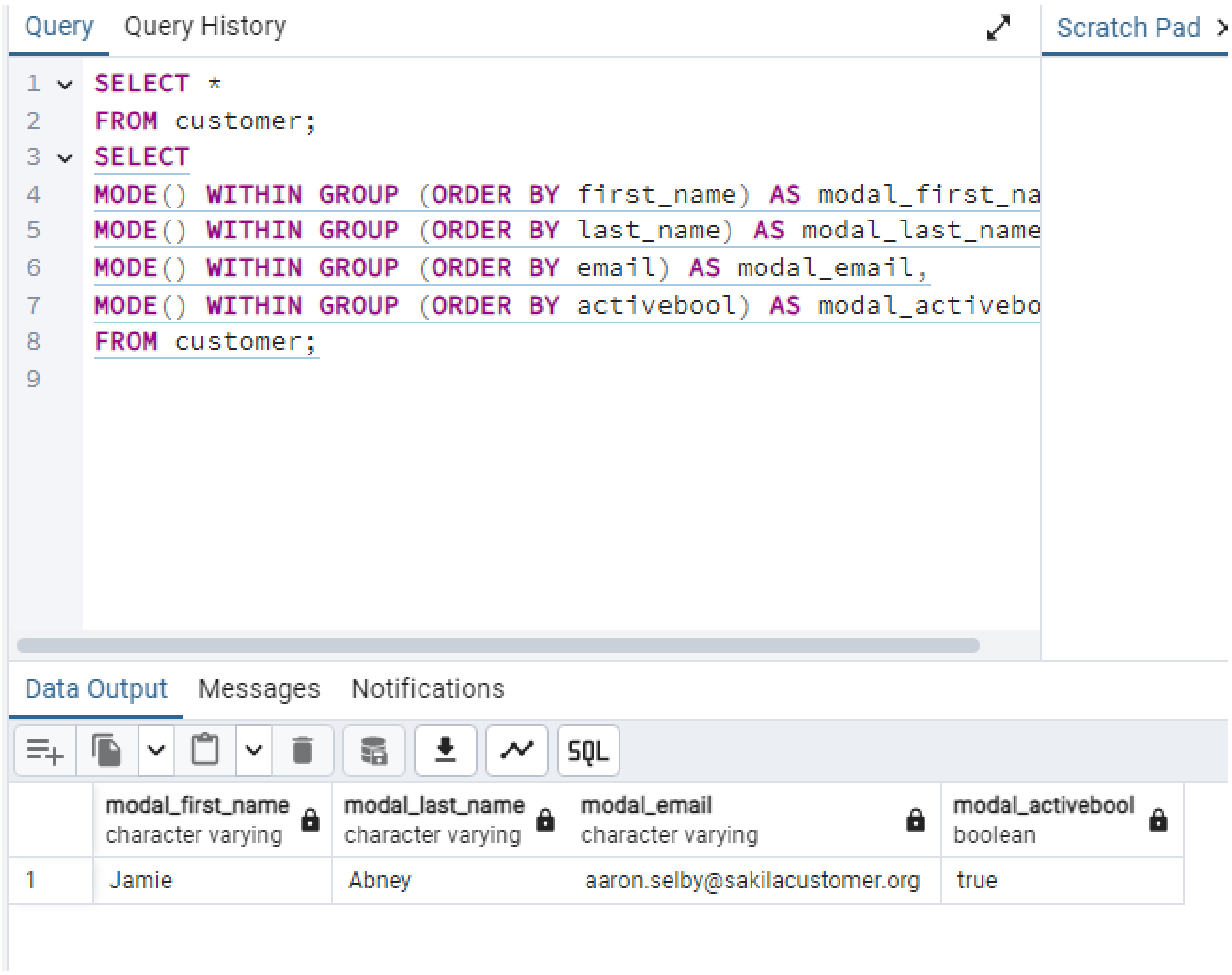
Numerical Values



```
SELECT
COUNT(*) AS total_customers,
AVG(customer_id) AS average_customer_id
MIN(customer_id) AS min_customer_id,
MAX(customer_id) AS max_customer_id,
AVG(store_id) AS average_store_id,
MIN(store_id) AS min_store_id,
MAX(store_id) AS max_store_id,
AVG(address_id) AS average_address_id,
MIN(address_id) AS min_address_id,
MAX(address_id) AS max_address_id,
MIN(create_date) AS min_create_date,
MAX(create_date) AS max_create_date,
AVG(active) AS average_active,
MIN(active) AS min_active,
MAX(active) AS max_active
FROM customer;
```

total_custo	average_cu	min_custor	max_custoı	average_stı	min_store_	max_store_
599	300	1	599	1.45576	1	2

Non Numerical Values



Query | Query History | Scratch Pad

```
SELECT *
FROM customer;
SELECT
MODE() WITHIN GROUP (ORDER BY first_name) AS modal_first_na
MODE() WITHIN GROUP (ORDER BY last_name) AS modal_last_name
MODE() WITHIN GROUP (ORDER BY email) AS modal_email,
MODE() WITHIN GROUP (ORDER BY activebool) AS modal_activebo
FROM customer;
```

Data Output | Messages | Notifications

	modal_first_name character varying	modal_last_name character varying	modal_email character varying	modal_activebool boolean
1	Jamie	Abney	aaron.selby@sakilacustomer.org	true

<u>Step 3</u>

I believe SQL is more effective for data profiling. I think with the correct query, you can find out a lot of information practically instantly. In Excel, it may take longer to reach the same result. I also feel like much more complicated questions in SQL can be answered easier. That being said, you have to teach yourself the language of SQL and it can be frustrating at first to learn all the syntax. SQL allows you to extract a column and analyze it separately from the main table whereas in excel, you could do that but it would mean copy and pasting the entire row. SQL also handles larger datasets much easier. Cleaning datasets in Excel seems to be a much more labour-intensive process. SQL does lack the capacity to perform visual analysis on datasets.