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.

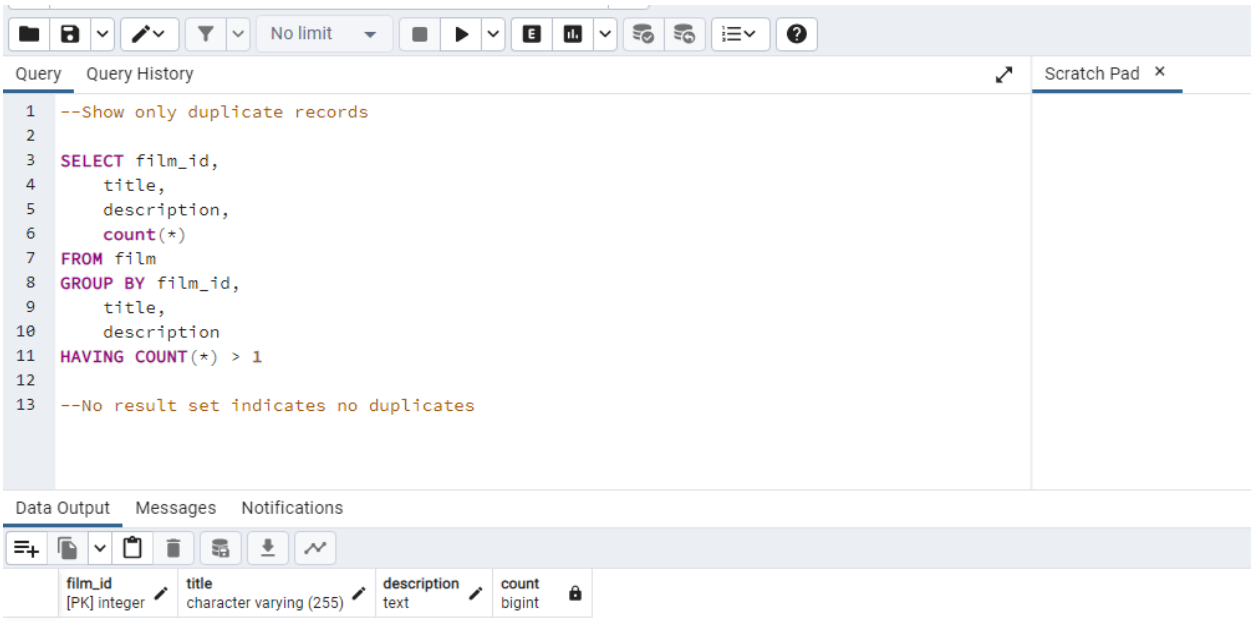
Checking for duplicate records (this could also be done with the DISTINCT queries shown for the non-uniform answer below):

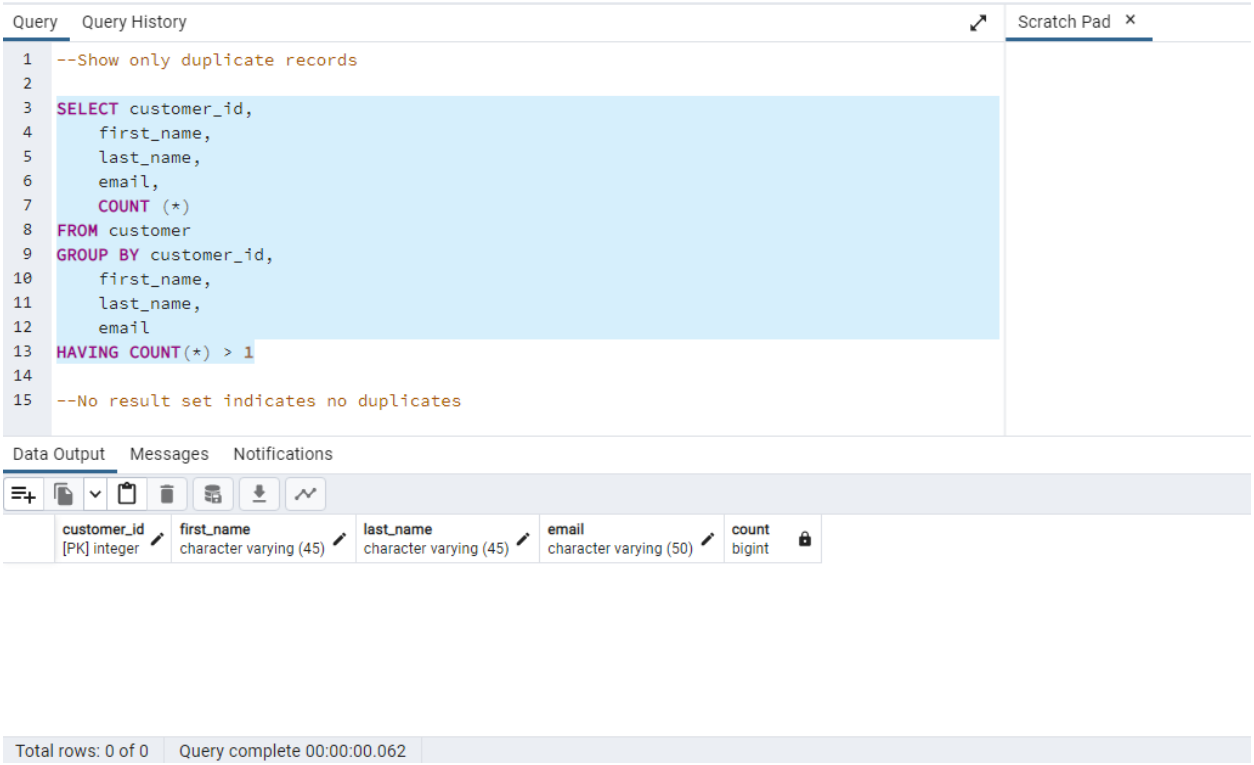
Query | Query History | Scratch Pad

```
1  --Show only unique records in film table
2
3  SELECT DISTINCT film_id,
4      title,
5      description
6  FROM film
7
8
```

Data Output | Messages | Notifications

	film_id [PK] integer	title character varying (255)	description text
1	503	Kramer Chocolate	A Amazing Yarn of a Robot And a Pastry Chef who must Redeem a Mad Scientist in The Outback
2	182	Control Anthem	A Fateful Documentary of a Robot And a Student who must Battle a Cat in A Monastery
3	158	Clones Pinocchio	A Amazing Drama of a Car And a Robot who must Pursue a Dentist in New Orleans
4	643	Orient Closer	A Astounding Epistle of a Technical Writer And a Teacher who must Fight a Squirrel in The Sahara Desert
5	375	Grail Frankenstein	A Unbelieveable Saga of a Teacher And a Monkey who must Fight a Girl in An Abandoned Mine Shaft
6	35	Arachnophobia Rollercoaster	A Action-Packed Reflection of a Pastry Chef And a Composer who must Discover a Mad Scientist in The First M



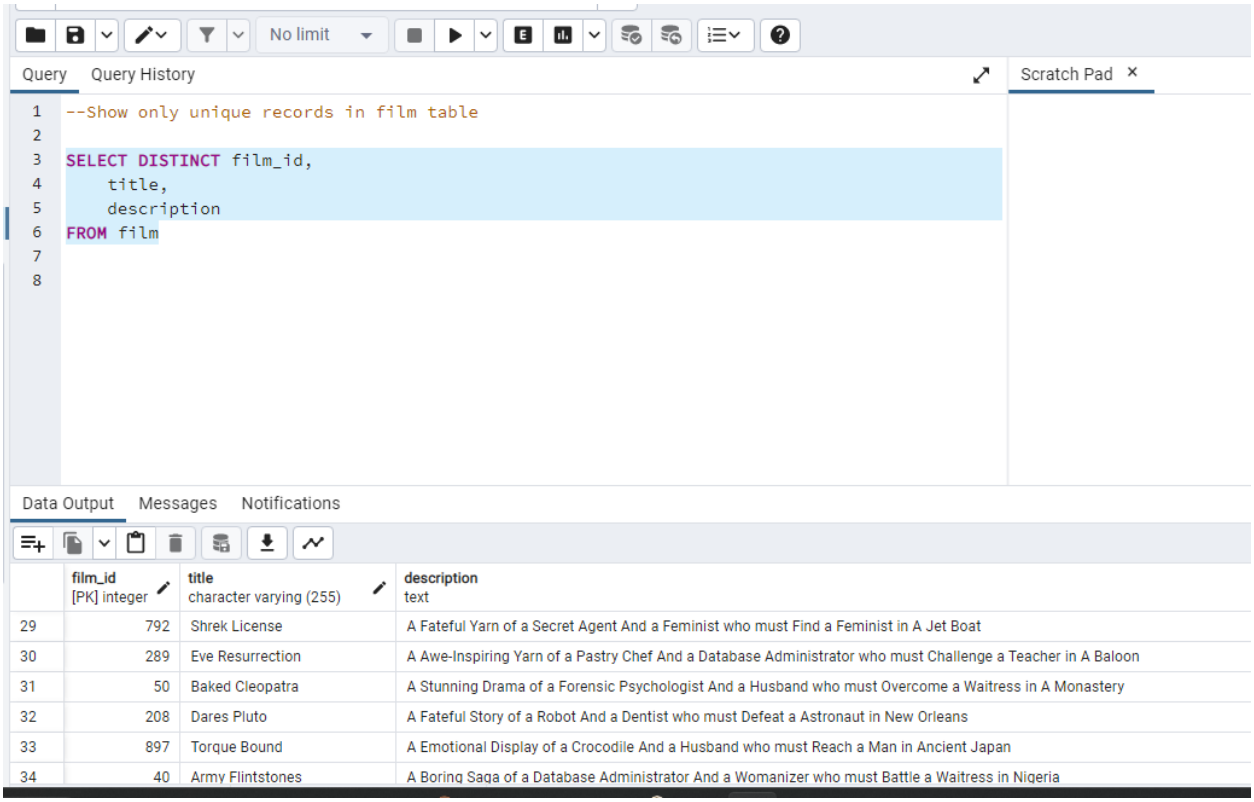


For duplicate data, I would create a View of Unique records for each of the tables to use for analysis. I could query those views without altering the main tables in the database.

```sql
--Show only unique records in customer table

CREATE VIEW customer_unique AS
SELECT customer_id,
    first_name,
    last_name,
    email
FROM customer
GROUP BY customer_id,
    first_name,
    last_name,
    email
```

Checking for non-uniform records: I show the DISTINCT queries here to differentiate them from the duplicate queries above. However, quickly identifying differences in the ways data had been entered would be more easily accomplished with the GROUP BY queries shown above.



Query | Query History | Scratch Pad

```sql
--Show only unique records in customer table

SELECT DISTINCT customer_id,
    first_name,
    last_name,
    email
FROM customer
```

Data Output | Messages | Notifications

	customer_id [PK] integer	first_name character varying (45)	last_name character varying (45)	email character varying (50)
1	114	Grace	Ellis	grace.ellis@sakilacustomer.org
2	374	Jeremy	Hurtado	jeremy.hurtado@sakilacustomer.org
3	172	Bernice	Willis	bernice.willis@sakilacustomer.org
4	556	Armando	Gruber	armando.gruber@sakilacustomer.org
5	420	Jacob	Lance	jacob.lance@sakilacustomer.org
6	354	Justin	Ngo	justin.ngo@sakilacustomer.org

If I identified non-uniform entries with a GROUP BY query, I could use an UPDATE query to SET one uniform value.

Checking for missing records: I could use COUNT queries to identify columns that have missing values.

Query | Query History | Scratch Pad

```sql
--Identifying columns with missing values

SELECT COUNT(film_id) AS film_id_count,
    COUNT(title) AS title_count,
    COUNT(description) AS description_count,
    COUNT(release_year) AS release_year_count,
    COUNT(language_id) AS language_id_count,
    COUNT(rental_duration) AS rental_duration_count,
    COUNT(rental_rate) AS rental_rate_count
FROM film
```

Data Output | Messages | Notifications

	film_id_count bigint	title_count bigint	description_count bigint	release_year_count bigint	language_id_count bigint	rental_duration_count bigint	rental_rate_count bigint
1	1000	1000	1000	1000	1000	1000	1000

If there were many missing values in a column, I would eliminate that column from my analysis. If there were only a few missing values, I would impute values for the missing records.

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.

Query | Query History | Scratch Pad

```sql
--Summarizing the data from film table

SELECT MIN(release_year) AS min_release_year,
    MAX(release_year) AS max_release_year,
    MODE() WITHIN GROUP (ORDER BY release_year) AS modal_release_year,
    MODE() WITHIN GROUP (ORDER BY language_id) AS modal_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,
    MODE() WITHIN GROUP (ORDER BY rating) AS modal_rating,
    MIN(last_update) AS min_last_update,
    MAX(last_update) AS max_last_update,
    MODE() WITHIN GROUP (ORDER BY last_update) AS modal_last_update
FROM film

```

Data Output | Messages | Notifications

Query | Query History | Scratch Pad

```sql
--Summarizing the data from customer table

SELECT MODE() WITHIN GROUP (ORDER BY store_id) AS modal_store_id,
    MODE() WITHIN GROUP (ORDER BY activebool) AS modal_activebool,
    MIN(create_date) AS min_create_date,
    MAX(create_date) AS max_create_date,
    MODE() WITHIN GROUP (ORDER BY create_date) AS modal_create_date,
    MIN(last_update) AS min_last_update,
    MAX(last_update) AS max_last_update,
    MODE() WITHIN GROUP (ORDER BY last_update) AS modal_last_update
FROM customer


```

Data Output | Messages | Notifications

	modal_store_id smallint	modal_activebool boolean	min_create_date date	max_create_date date	modal_create_date date	min_last_update timestamp without time zone	max_last_update timestamp without time zone	modal timest
1	1	true	2006-02-14	2006-02-14	2006-02-14	2013-05-26 14:49:45.738	2013-05-26 14:49:45.738	2013-