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).

## Checking for Duplicate Data

### Film table

Query  Query History  Scratch Pad

```
SELECT film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext,
       COUNT(*)
FROM film
GROUP BY film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext
HAVING COUNT(*)>1
```

Data Output  Messages  Notifications

| film_id [PK] integer | title character varying (255) | 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_update timestamp without time zone | special_features text[] | fulltext tsvector | count bigint |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

### Customer Table

Query  Query History

```
SELECT customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       activebool,
       create_date,
       last_update,
       active,
       COUNT(*)
FROM customer
GROUP BY customer_id,
       store_id,
       first_name,
       last_name,
       email,
       address_id,
       activebool,
       create_date,
       last_update,
       active
HAVING COUNT(*)>1
```

Data Output  Messages  Notifications

| customer_id [PK] integer | store_id smallint | first_name character varying (45) | last_name character varying (45) | email character varying (50) | address_id smallint | activebool boolean | create_date date | last_update timestamp without time zone | active integer | count bigint |
|---|---|---|---|---|---|---|---|---|---|---|

**Duplicate Data**

If you find duplicate records in your database, there are two ways to fix them:

1. Create a virtual table, known as a "view," where you select only unique records.

**Option 1: Creating a View with Unique Records**

```
CREATE VIEW viewname AS
SELECT col1,
       col2,
       col3 ...
FROM tablename
GROUP BY col1,
         col2,
         col3, ... --Group by will make each row
unique
```

2. Delete the duplicate record from the table or view.

```
/* This will delete all duplicate records, keepin
g only one of its versions with the lowest key/un
ique ID value. There must be a unique column in t
he table for this to work.
unique_id is any column acting as a primary key u
nique for each row
*/
DELETE
FROM tablename
WHERE unique_id NOT IN
    (SELECT MIN(unique_id)
     FROM tablename
     GROUP BY col1,
              col2,
              col3, ...)
```

## Check for Non-Uniform Data

### Film table

Query  Query History

```
SELECT DISTINCT rating
FROM film
```

Data Output  Messages  Notifications

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

**Customer Table**

Query  Query History

```
SELECT DISTINCT first_name,
                last_name
FROM customer
ORDER BY first_name ASC,
         last_name DESC;
```

Data Output  Messages  Notifications

| | first_name character varying (45) | last_name character varying (45) |
|---|---|---|
| 152 | Diane | Collins |
| 153 | Dianne | Shelton |
| 154 | Dolores | Wagner |
| 155 | Don | Bone |
| 156 | Donald | Mahon |
| 157 | Donna | Thompson |

Total rows: 599 of 599  Query complete 00:00:00.101

**Non-Uniform Data**

To check for inconsistencies, you would use the GROUP BY and DISTINCT keywords. To fix and update the values in a column for consistency, you would need to use the SQL command below:

```
UPDATE film

SET rating = 'G'

WHERE rating IN ('gen',

                 'g',

                 'General')
```

## Check for Missing Data

### Film table

Query  Query History  Scratch Pad

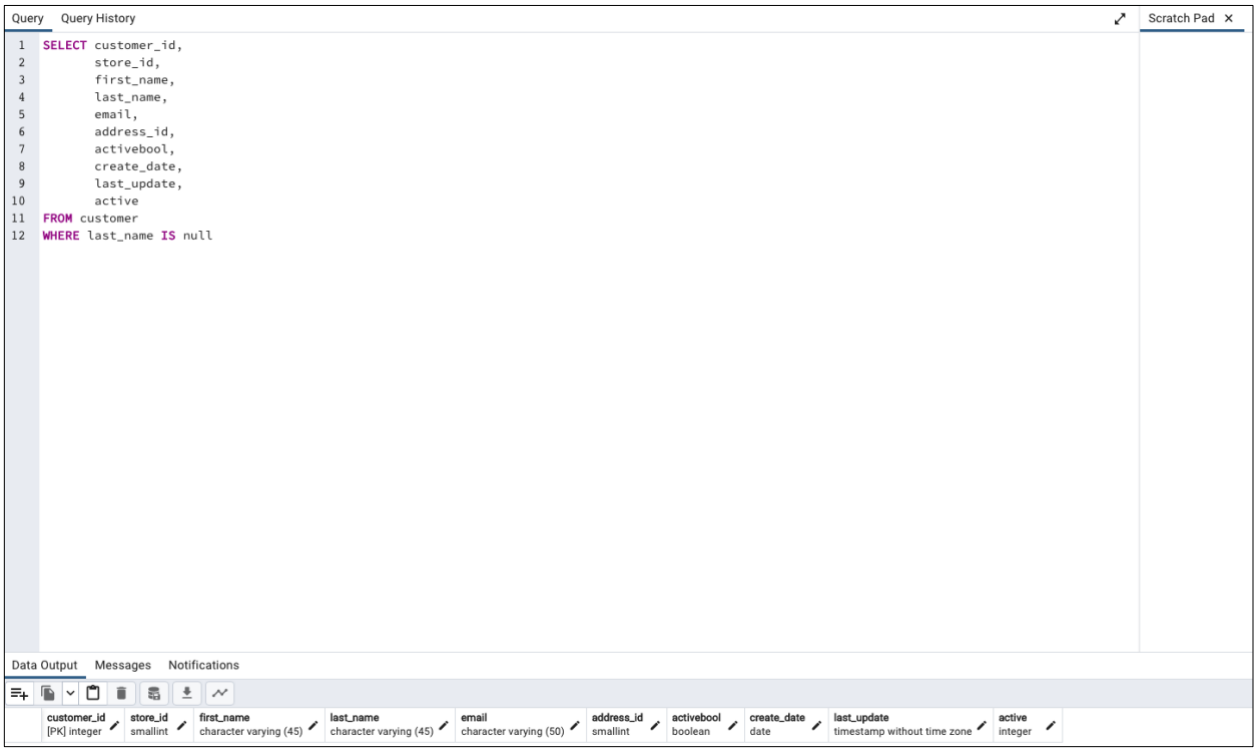
```
SELECT film_id,
       title,
       description,
       release_year,
       language_id,
       rental_duration,
       rental_rate,
       length,
       replacement_cost,
       rating,
       last_update,
       special_features,
       fulltext
FROM film
WHERE rating IS null
```

Data Output  Messages  Notifications

| film_id [PK] integer | title character varying (255) | 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_update timestamp without time zone | special_features text[] | fulltext tsvector |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

You can run the same query for each individual column to determine any missing values. However, there are no missing values in the film table.

## Customer Table




## Missing Data

To avoid missing values or minimize their impact on an analysis. The first approach is to simply ignore columns with a high percentage of missing values. To do so in SQL, you simply omit whichever column you want to ignore from your SELECT statement.

```
SELECT col1,

    col2,

    col4… --col3 ignored in select because it has a lot of missing values

FROM tablename
```

The second approach is to impute the missing values using statistical methods.

```
--imputing missing values with the AVG value



UPDATE tablename

SET = AVG(col1)
```

```
WHERE col1 IS NULL
```

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.

**Minimum, maximum, and average values for film table**

**Numerical columns (film_id, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, last_update)**

Query  Query History

```sql
SELECT MIN(film_id) AS min_film_id,
       MAX(film_id) AS max_film_id,
       AVG(film_id) AS avg_film_id,
       COUNT(film_id) AS count_film_id_values,
       MIN(release_year) AS min_release_year,
       MAX(release_year) AS max_release_year,
       AVG(release_year) AS avg_release_year,
       COUNT(release_year) AS count_release_year_values,
       MIN(language_id) AS min_language_id,
       MAX(language_id) AS max_language_id,
       AVG(language_id) AS avg_language_id,
       COUNT(language_id) AS count_language_id_values,
       MIN(rental_duration) AS min_rental_duration,
       MAX(rental_duration) AS max_rental_duration,
       AVG(rental_duration) AS avg_rental_duration,
       COUNT(rental_duration) AS count_rental_duration_values,
       MIN(rental_rate) AS min_rental_rate,
       MAX(rental_rate) AS max_rental_rate,
       AVG(rental_rate) AS avg_rental_rate,
       COUNT(rental_rate) AS count_rental_rate_values,
       MIN(length) AS min_length,
       MAX(length) AS max_length,
       AVG(length) AS avg_length,
       COUNT(length) AS count_length_values,
       MIN(replacement_cost) AS min_replacement_cost,
       MAX(replacement_cost) AS max_replacement_cost,
       AVG(replacement_cost) AS avg_replacement_cost,
       COUNT(replacement_cost) AS count_replacement_cost_values,
       MIN(last_update) AS min_last_update,
       MAX(last_update) AS max_last_update,
       COUNT(last_update) AS count_last_update_values,
       COUNT(*) AS count_rows
FROM film;
```

Data Output  Messages  Notifications

| | min_film_id integer | max_film_id integer | avg_film_id numeric | count_film_id_values bigint | min_release_year integer | max_release_year integer | avg_release_year numeric | count_release_year_values bigint | min_language_id smallint | max_lar smallint |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1000 | 500.5000000000000000 | 1000 | 2006 | 2006 | 2006.0000000000000000 | 1000 | 1 | |

**Mode values for film table**

**Non-numerical columns (title, description, rating, special_features, fulltext)**

Query | Query History | Scratch Pad

```sql
SELECT mode() WITHIN GROUP (ORDER BY title)
       AS modal_value,
       mode() WITHIN GROUP (ORDER BY description)
       AS modal_value,
       mode() WITHIN GROUP (ORDER BY rating)
       AS modal_value,
       mode() WITHIN GROUP (ORDER BY special_features)
       AS modal_value,
       mode() WITHIN GROUP (ORDER BY fulltext)
       AS modal_value
FROM film;
```

Data Output | Messages | Notifications

| | modal_value (character varying) | modal_value (text) | modal_value (mpaa_rating) | modal_value (text[]) | modal_value (tsvector) |
|---|---|---|---|---|---|
| 1 | Academy Dinosaur | A Action-Packed Character Study of a Astronaut And a Explorer who must Reach a Monkey in A MySQL Convent... | PG-13 | {Trailers,Commentaries,"Behind the Scenes"} | 'baloon':19 'confront':14 'documentari':5 'feminist':8,11,16 'mile':2 'must':13 'spi':1 'thri... |

**Minimum, maximum, and average values for customer table**

**Numerical columns (customer_id, store_id, address_id, create_date, last_update, active)**

Query  Query History  Scratch Pad

```sql
SELECT MIN(customer_id) AS min_customer_id,
       MAX(customer_id) AS max_customer_id,
       AVG(customer_id) AS avg_customer_id,
       COUNT(customer_id) AS count_customer_id_values,
       MIN(store_id) AS min_store_id,
       MAX(store_id) AS max_store_id,
       AVG(store_id) AS avg_store_id,
       COUNT(store_id) AS count_store_id_values,
       MIN(address_id) AS min_address_id,
       MAX(address_id) AS max_address_id,
       AVG(address_id) AS avg_address_id,
       COUNT(address_id) AS count_address_id_values,
       MIN(create_date) AS min_create_date,
       MAX(create_date) AS max_create_date,
       COUNT(create_date) AS count_create_date_values,
       MIN(last_update) AS min_last_update,
       MAX(last_update) AS max_last_update,
       COUNT(last_update) AS avg_last_update_values,
       MIN(active) AS min_active,
       MAX(active) AS max_active,
       AVG(active) AS avg_active,
       COUNT(active) AS count_active_values,
       COUNT(*) AS count_rows
FROM customer;
```

Data Output  Messages  Notifications

| | min_customer_id integer | max_customer_id integer | avg_customer_id numeric | count_customer_id_values bigint | min_store_id smallint | max_store_id smallint | avg_store_id numeric | count_store_id_values bigint | min_address_id smallint | max_address_id smallint | avg_address_id numeric |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 599 | 300.0000000000000000 | 599 | 1 | 2 | 1.4557595993322204 | 599 | 5 | 605 | 304.7245409015025042 |

**Mode values for customer table**

**Non-numerical columns (first_name, last_name, email, activebool)**

Query  Query History

```sql
SELECT mode() WITHIN GROUP (ORDER BY first_name)
       AS modal_value,
       mode() WITHIN GROUP (ORDER BY last_name)
       AS modal_value,
       mode() WITHIN GROUP (ORDER BY email)
       AS modal_value,
       mode() WITHIN GROUP (ORDER BY activebool)
       AS modal_value
FROM customer;
```

Data Output  Messages  Notifications

| | modal_value character varying | modal_value character varying | modal_value character varying | modal_value boolean |
|---|---|---|---|---|
| 1 | 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.

   SQL is the language used to interact with relational databases. It can work with large volumes of data and operates fast to get results. A well-written SQL query can fetch results from a few million rows within a minute. By using SQL, you can write a query once and reuse it again. As the data increases in your database, you are not required to change much in your query to accommodate similar results.