**Summarizing and Cleaning Data in SQL**

Rockbuster's database engineers have loaded some new data into the database, and your manager has asked you to clean and profile it.

1. **Check for and clean dirty data:**

**To check for duplicate data:**

```
--Shows only those records that duplicate(based on columns selected)

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

I'd use this query. If there would be any duplicates, some results would show up. To fix it, I would:

1. Create a virtual table, known as a "view," where I could select only unique records, like this:

```
CREATE VIEW viewname AS unique_films
SELECT title,
       release_year,
       language_id,
       rental_duration
FROM film
GROUP BY title,
          release_year,
          language_id,
          rental_duration --Group by will make each row unique
```

2.Delete the duplicate record from the table or view, like this:

```
 /* This will delete all duplicate records, keeping only one of its versions with the lowest key/unique ID value. There must be a unique column in the table for this to work.
 unique_id is any column acting as a primary key unique for each row
 */
 DELETE
 FROM film
```

```
WHERE film_id NOT IN (
     SELECT MIN(film_id)
     FROM film
     GROUP BY title,
             release_year,
             language_id,
             rental_duration
);
```

**To check for non-uniform data:**

```
SELECT DISTINCT language_id FROM film;
SELECT DISTINCT release_year FROM film;
SELECT DISTINCT rental_duration FROM film;
SELECT DISTINCT title FROM film;
--I would proceed like this for all headers.
```

In case any non-uniform data would show up, lets say in ratings, I would do this:

```
UPDATE film
SET rating = 'G'
WHERE rating IN ('gen',
          'g',
          'General')
```

**To check for missing data:**

```
SELECT *
FROM film
WHERE title IS NULL
  OR release_year IS NULL
  OR language_id IS NULL
  OR rental_duration IS NULL;
--I’d proceed like this for all columns/headers.
```

If any values would have showed up, I would omit them by doing this, imagining that rental_duration had a lot of missing values:

```
SELECT title,
          release_year,
          language_id
--rental_duration ignored in select because it has a lot of missing values
FROM film
```

If possible, I would impute values like this:

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

UPDATE film
SET = AVG(rental_duration)
WHERE rental_duration IS NULL
```

2. **Summarize data:** For numerical columns, this means finding the minimum, maximum, and average values. For non-numerical columns, calculate the mode value.

Firstly, to retrieve the names of the columns and respective data types **for table "film":**

```
SELECT
   column_name,
   data_type
FROM
   information_schema.columns
WHERE
   table_name = 'film'
ORDER BY
   ordinal_position;
```

Then, with those results, to give this query to pgAdmin4 for the summarizing of the data table:

```
-- Enable tablefunc extension if needed
-- CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT
  -- Total row count
  COUNT(*) AS total_rows,

  -- film_id
  COUNT(film_id) AS count_film_id,
  MIN(film_id) AS min_film_id,
  MAX(film_id) AS max_film_id,
  MODE() WITHIN GROUP (ORDER BY film_id) AS mode_film_id,

  -- release_year
  COUNT(release_year) AS count_release_year,
  MIN(release_year) AS min_release_year,
  MAX(release_year) AS max_release_year,
  AVG(release_year) AS avg_release_year,
  MODE() WITHIN GROUP (ORDER BY release_year) AS mode_release_year,
  -- language_id
  COUNT(language_id) AS count_language_id,
  MIN(language_id) AS min_language_id,
  MAX(language_id) AS max_language_id,
  MODE() WITHIN GROUP (ORDER BY language_id) AS mode_language_id,

  -- rental_duration
  COUNT(rental_duration) AS count_rental_duration,
  MIN(rental_duration) AS min_rental_duration,
  MAX(rental_duration) AS max_rental_duration,
  AVG(rental_duration) AS avg_rental_duration,
  MODE() WITHIN GROUP (ORDER BY rental_duration) AS mode_rental_duration,

  -- rental_rate
  COUNT(rental_rate) AS count_rental_rate,
  MIN(rental_rate) AS min_rental_rate,
  MAX(rental_rate) AS max_rental_rate,
  AVG(rental_rate) AS avg_rental_rate,
  MODE() WITHIN GROUP (ORDER BY rental_rate) AS mode_rental_rate,

  -- length
  COUNT(length) AS count_length,
  MIN(length) AS min_length,
  MAX(length) AS max_length,
  AVG(length) AS avg_length,
  MODE() WITHIN GROUP (ORDER BY length) AS mode_length,

  -- replacement_cost
  COUNT(replacement_cost) AS count_replacement_cost,
  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 replacement_cost) AS
mode_replacement_cost,

  -- rating
  COUNT(rating) AS count_rating,
  MODE() WITHIN GROUP (ORDER BY rating) AS mode_rating

FROM film;
```

The non numerical columns title, description, special_features, fulltext, and last_update were not aggregated.

**For the customer table:**

```
SELECT
   column_name,
   data_type
FROM
   information_schema.columns
WHERE
   table_name = 'customer'
ORDER BY
   ordinal_position;
```

Then, with those results, to give this query to pgAdmin4 for the summarizing of the data table:

```
-- Ensure the mode function is available
-- CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT
  -- Total number of records
  COUNT(*) AS total_rows,

  -- customer_id
  COUNT(customer_id) AS count_customer_id,
  MIN(customer_id) AS min_customer_id,
  MAX(customer_id) AS max_customer_id,
  MODE() WITHIN GROUP (ORDER BY customer_id) AS mode_customer_id,

  -- store_id
  COUNT(store_id) AS count_store_id,
  MIN(store_id) AS min_store_id,
  MAX(store_id) AS max_store_id,
  MODE() WITHIN GROUP (ORDER BY store_id) AS mode_store_id,

  -- address_id
  COUNT(address_id) AS count_address_id,
  MIN(address_id) AS min_address_id,
  MAX(address_id) AS max_address_id,
  MODE() WITHIN GROUP (ORDER BY address_id) AS mode_address_id,

  -- activebool (boolean)
  COUNT(activebool) AS count_activebool,
  MODE() WITHIN GROUP (ORDER BY activebool) AS mode_activebool,

  -- active (integer; duplicate of activebool in some schemas)
  COUNT(active) AS count_active,
  MIN(active) AS min_active,
  MAX(active) AS max_active,
  AVG(active) AS avg_active,
  MODE() WITHIN GROUP (ORDER BY active) AS mode_active,

  -- create_date
  COUNT(create_date) AS count_create_date,
  MIN(create_date) AS earliest_create_date,
  MAX(create_date) AS latest_create_date,
  MODE() WITHIN GROUP (ORDER BY create_date) AS mode_create_date,

  -- last_update
  COUNT(last_update) AS count_last_update,
  MIN(last_update) AS earliest_update,
  MAX(last_update) AS latest_update

FROM customer;
```

**Notes:**
Skipped aggregations for first_name, last_name, and email, since they are too granular (often unique or low-use for summaries).
activebool is treated as categorical (mode).
active is treated numerically **and** categorically.
Timestamps and dates are included using MIN, MAX, and MODE().