Exercise 3.6 Summarizing & Cleaning Data in SQL

Checking for Dirty data

Film Table : Stores details of each film in the inventory.

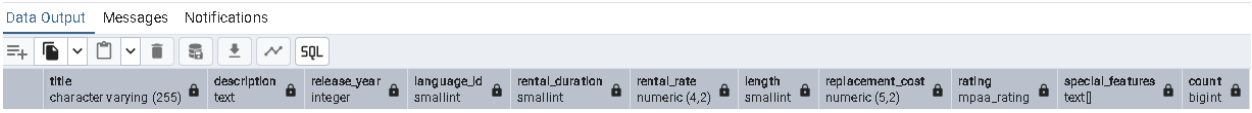
Column	Data Type	Description
film_id	integer	Unique identifier for each film
title	varchar	Title of the film
description	text	Film description
release_year	integer	Year the film was released
language_id	integer	Foreign key referencing the language table
rental_duration	smallint	Rental duration in days
rental_rate	numeric	Cost of renting the film
length	smallint	Duration of the film in minutes
replacement_cost	numeric	Cost of replacing the film
rating	user-defined	Film rating
special_features	array	Additional features of the film
last_update	timestamp	Record's last modification date

Query for checking Duplicate Records in the Film Table :

```
SELECT  title,
description,
```

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

Output :



There are no duplicate records in the Film Table.

Query for checking Non-uniform Records in the Film Table :

1. Using the Distinct Command :

```
SELECT  DISTINCT
film_id
title,
description,
release_year,
language_id,
rental_duration,
rental_rate,
length,
replacement_cost,
rating,
last_update,
special_features,
fulltext
FROM film;
```

Output :

Data Output | Messages | Notifications

Showing rows: 1 to 1000 | Page No: 1 of 1

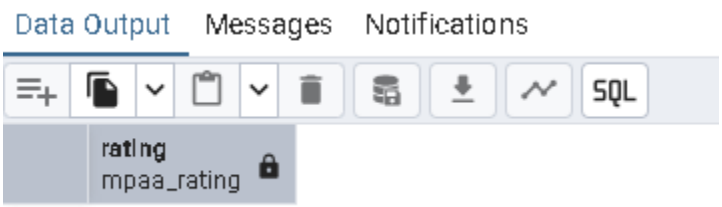
	title integer	description text	release_year integer	language_id smallint	rental_duration smallint	rental_rate numeric (4,2)	length smallint	replacement_cost numeric (5,2)
1	1	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	2006	1	6	0.99	86	20.99
2	2	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China	2006	1	3	4.99	48	12.99
3	3	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory	2006	1	7	2.99	50	18.99
4	4	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank	2006	1	5	2.99	117	26.99
5	5	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico	2006	1	6	2.99	130	22.99

There are no non-uniform records in the Film Table.

2. Identifying invalid ratings (another way to check for non-uniform records)

```
SELECT  DISTINCT  rating
FROM film
WHERE rating NOT IN ('G', 'PG', 'PG-13', 'R', 'NC-17');
```

Output :



Data Output | Messages | Notifications

SQL

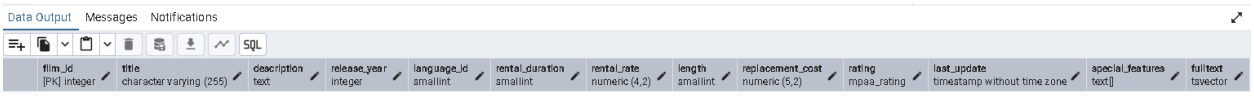
	rating mpaa_rating

There are no invalid ratings in the Film Table.

Query for checking missing values in the Film Table :

```
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 special_features IS NULL
OR last_update IS NULL;
```

Output :



There are no NULL values in any column of the Film table.

Customer Table : Stores customer details.

Column	Data Type	Description
customer_id	integer	Unique customer ID
store_id	smallint	Store where the customer is registered
first_name	varchar	Customer's first name
last_name	varchar	Customer's last name
email	varchar	Customer's email address
address_id	smallint	Foreign key referencing the address table
activebool	boolean	Indicates if the customer is active
create_date	date	Customer account creation date
last_update	timestamp	Record's last modification date
active	integer	Legacy field indicating customer activity status

Query for checking Duplicate Records in the Customer Table :

```
SELECT store_id,
first_name,
last_name,
email,
```

```
address_id,
activebool,
COUNT(*) AS duplicate_count
FROM customer
GROUP BY store_id,
 first_name,
 last_name,
 email,
 address_id,
 activebool
HAVING COUNT(*) > 1;
```

Output :

There are no Duplicate Records in the Customer Table.

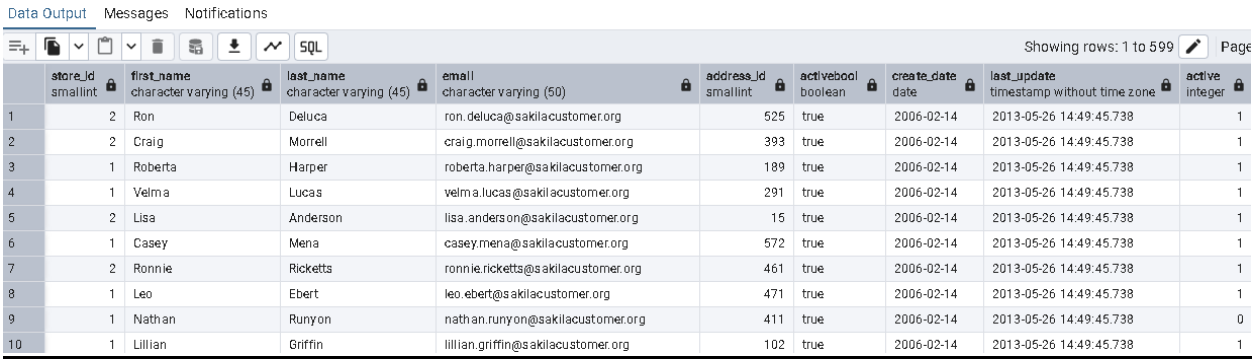
Query for checking Non-uniform Records in the Customer Table :

Using the Distinct Command

```
SELECT DISTINCT store_id,
first_name,
last_name,
email,
address_id,
```

```
activebool,
create_date,
last_update,
active
FROM customer;
```

Output :



Data Output | Messages | Notifications

Showing rows: 1 to 599 | Page

	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
1	2	Ron	Deluca	ron.deluca@sakilacustomer.org	525	true	2006-02-14	2013-05-26 14:49:45.738	1
2	2	Craig	Morrell	craig.morrell@sakilacustomer.org	393	true	2006-02-14	2013-05-26 14:49:45.738	1
3	1	Roberta	Harper	roberta.harper@sakilacustomer.org	189	true	2006-02-14	2013-05-26 14:49:45.738	1
4	1	Velma	Lucas	velma.lucas@sakilacustomer.org	291	true	2006-02-14	2013-05-26 14:49:45.738	1
5	2	Lisa	Anderson	lisa.anderson@sakilacustomer.org	15	true	2006-02-14	2013-05-26 14:49:45.738	1
6	1	Casey	Mena	casey.mena@sakilacustomer.org	572	true	2006-02-14	2013-05-26 14:49:45.738	1
7	2	Ronnie	Ricketts	ronnie.ricketts@sakilacustomer.org	461	true	2006-02-14	2013-05-26 14:49:45.738	1
8	1	Leo	Ebert	leo.ebert@sakilacustomer.org	471	true	2006-02-14	2013-05-26 14:49:45.738	1
9	1	Nathan	Runyon	nathan.runyon@sakilacustomer.org	411	true	2006-02-14	2013-05-26 14:49:45.738	0
10	1	Lillian	Griffin	lillian.griffin@sakilacustomer.org	102	true	2006-02-14	2013-05-26 14:49:45.738	1

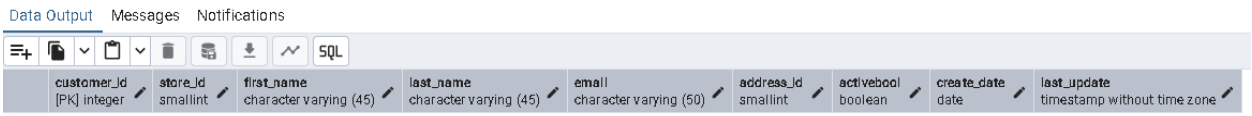
There are no non-uniform records in the Customer Table.

Query for checking missing values in the Customer Table :

```
SELECT customer_id,
store_id,
first_name,
last_name,
email,
address_id,
activebool,
create_date,
last_update
```

```
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;
```

Output :



There are no Null Values in the Customer Table.

Cleaning Duplicate Data

1. Create a Virtual Table (View) : We can create a view to filter out duplicates by selecting only unique records. This ensures the duplicates are excluded without altering the original table data.

2. Delete Duplicate Records : We can delete the duplicates from the table by retaining only one instance of each record. This involves identifying duplicates and using a DELETE statement with appropriate filtering conditions.

3. Prevention Strategies: We can implement database constraints, such as UNIQUE, to avoid duplicates in the future. This ensures data integrity and reduces the need for periodic cleanup.

Cleaning Non-Uniform Data

1. Identify inconsistencies : We will review the data to find variations in formats or values, such as differences in case (e.g., 'G' vs 'g') or invalid entries.
2. Standardize values : Use UPDATE queries to replace non-uniform data with standardized values based on a predefined set of valid options.
3. Use validation rules : Implement constraints or data validation checks to prevent non-uniform data from being entered in the future.

Cleaning Missing Records

1. Assess criticality : We will determine if the missing records are critical to the analysis or business processes.
2. Fill or delete : For important fields, fill missing values using default values, averages or calculated estimates. If the records are not critical remove them using DELETE.
3. Prevent future issues : Add constraints like NOT NULL or input validation at the database or application level to ensure completeness in future entries.

Summarizing the Data

To calculate the minimum, maximum and average values for all numerical columns and to calculate the mode value for all non-numerical columns.

Film Table

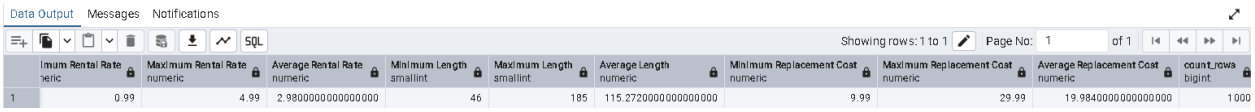
Numeric Variables : release_year, rental_duration, rental_rate, length, replacement_cost

Query :

```
SELECT MIN(release_year) AS "Minimum Release Year",
MAX(release_year) AS "Maximum Release Year",
AVG(release_year) AS "Average Release Year",
MIN(rental_duration) AS "Minimum Rental Duration",
MAX(rental_duration) AS "Maximum Rental Duration",
```

```
AVG(rental_duration) AS "Average Rental Duration",
 MIN(rental_rate) AS "Minimum Rental Rate",
 MAX(rental_rate) AS "Maximum Rental Rate",
 AVG(rental_rate) AS "Average Rental Rate",
 MIN(length) AS "Minimum Length",
 MAX(length) AS "Maximum Length",
 AVG(length) AS "Average Length",
 MIN(replacement_cost) AS "Minimum Replacement Cost",
 MAX(replacement_cost) AS "Maximum Replacement Cost",
 AVG(replacement_cost) AS "Average Replacement Cost",
 COUNT(*) AS count_rows
 FROM film;
```

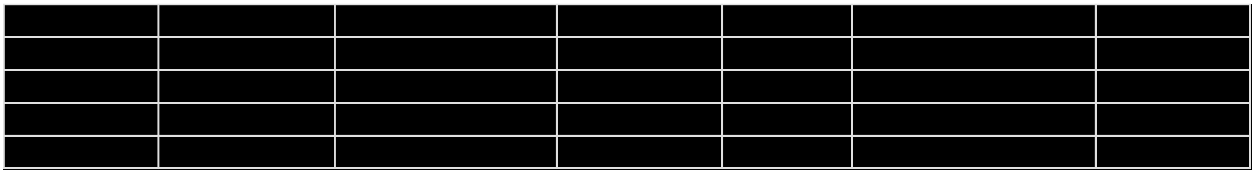
Output :



Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	Imum Rental Rate numeric	Maximum Rental Rate numeric	Average Rental Rate numeric	Minimum Length smallint	Maximum Length smallint	Average Length numeric	Minimum Replacement Cost numeric	Maximum Replacement Cost numeric	Average Replacement Cost numeric	count_rows bigint
1	0.99	4.99	2.9800000000000000	46	185	115.2720000000000000	9.99	29.99	19.9840000000000000	1000

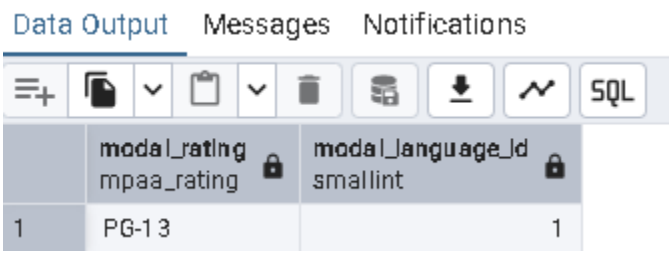


Non-Numerical Variables : language_id, rating

Query :

```
SELECT
MODE() WITHIN GROUP (ORDER BY rating) AS modal_rating,
MODE() WITHIN GROUP (ORDER BY language_id) AS
modal_language_id
FROM film;
```

Output :



Customer Table

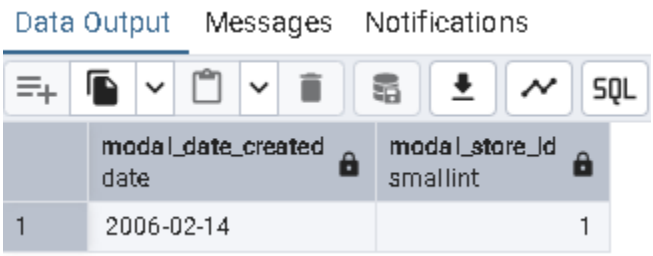
Numerical Variables : There are no numerical variables in this table.

Non-numerical Variables : create_date, store_id

Query :

```
SELECT
MODE() WITHIN GROUP (ORDER BY create_date) AS modal_date_created,
MODE() WITHIN GROUP (ORDER BY store_id) AS modal_store_id
FROM customer;
```

Output :



	modal_date_created date	modal_store_id smallint
1	2006-02-14	1

Reflection :

I personally enjoy using Excel because I've spent some time with it and find its interface simple and intuitive. It's perfect for quick tasks like creating pivot tables, visualizing data or spotting trends in smaller datasets. However, SQL is also a powerful tool and while it may feel challenging at first, once you get the hang of it, it becomes straightforward and highly efficient.

SQL is particularly valuable for handling larger datasets and performing complex queries such as grouping, filtering and joining tables. It processes data at remarkable speed, making it ideal for scalable and detailed data profiling. While Excel is great for

simplicity and quick visual insights, SQL is essential for tasks that demand power and precision. Both tools have their strengths, but mastering SQL unlocks a new level of capability for professional data analysis.