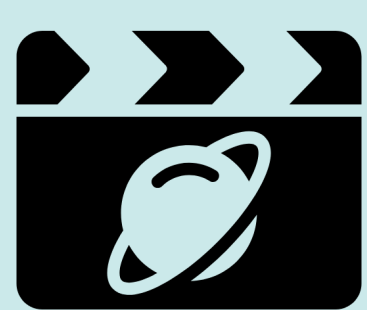
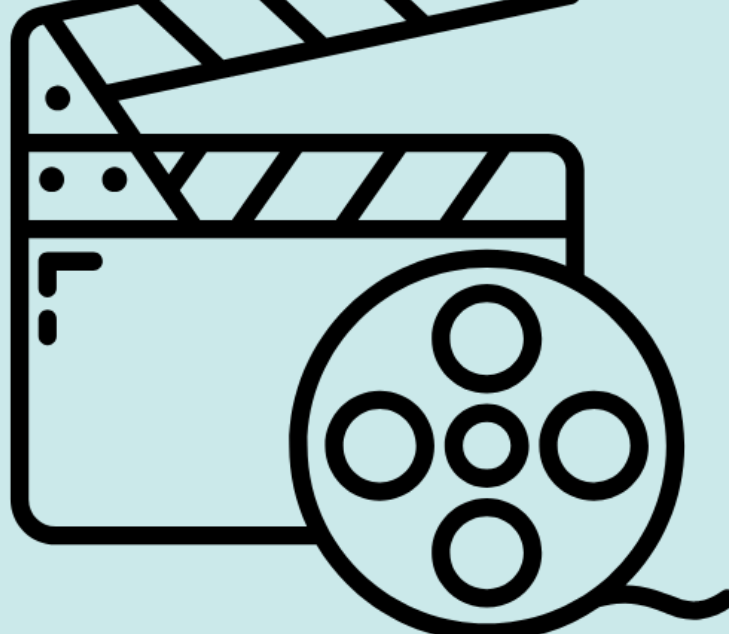


SUMMARIZING & CLEANING DATA IN SQL



Data Immersion	Achievement III		
	SQL for Data Analysts		
	Task 3.6 By Ola Gaffarova	Table of content Page 2 of 10	Made for Rockbuster Stealth

Checking if the table has non-uniform or duplicate data, or missing values

QueryQuery History

1

2

3

4

5

6

7

8

9

10

11

SELECT

title,

release_year,

language_id,

rental_duration,

COUNT(*)

FROM film

GROUP BY title,

release_year,

language_id,

rental_duration

HAVING COUNT(*) >1;

Data OutputMessagesNotifications

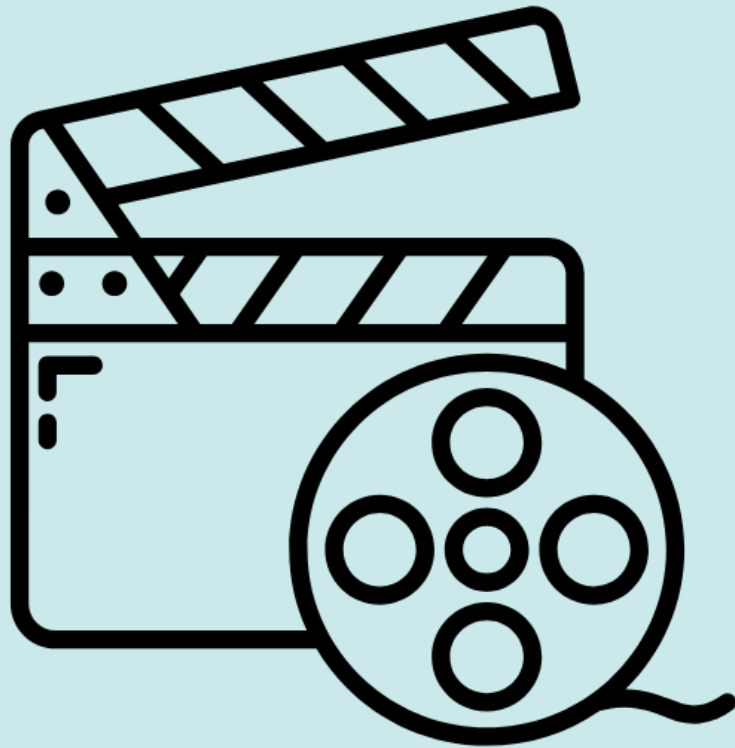
SQL

title	release_year	language_id	rental_duration	count
character varying (255)	integer	smallint	smallint	bigint

How would I clean the data?
Start with checking for duplicate values, misspellings, and incorrect data.

If I find duplicate records in the database I would:

- Create a virtual table, known as a view, that selects only unique records.
CREATE VIEW film_no_duplicates AS
SELECT
title, release_year, language_id, length, rating
FROM film
GROUP BY
film_id, title, release_year, language_id, length, rating
- Delete the duplicate records directly from the view (but not delete from the server).



Query

Query History

1

2

3

4

5

6

7

SELECT DISTINCT

title,

release_year,

language_id,

length,

rating

FROM film

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

Showing rows: 1 to 1000

	title character varying (255)	release_year integer	language_id smallint	length smallint	rating mpaa_rating
1	Show Lord	2006	1	167	PG-13
2	Interview Liaisons	2006	1	59	R
3	Instinct Airport	2006	1	116	PG
4	Insider Arizona	2006	1	78	NC-17
5	Oleander Clue	2006	1	161	PG
6	Side Ark	2006	1	52	G
7	Downhill Enough	2006	1	47	G
8	Drop Waterfront	2006	1	178	R
9	Amelie Hellfighters	2006	1	79	R
10	Ishtar Rocketeer	2006	1	79	R
11	Greedy Roots	2006	1	166	R
12	Mask Peach	2006	1	123	NC-17
13	Berets Agent	2006	1	77	PG-13
14	Fire Wolves	2006	1	173	R
15	Gone Trouble	2006	1	84	R
16	Caribbean Liberty	2006	1	92	NC-17







Total rows: 1000

Query complete 00:00:00.205

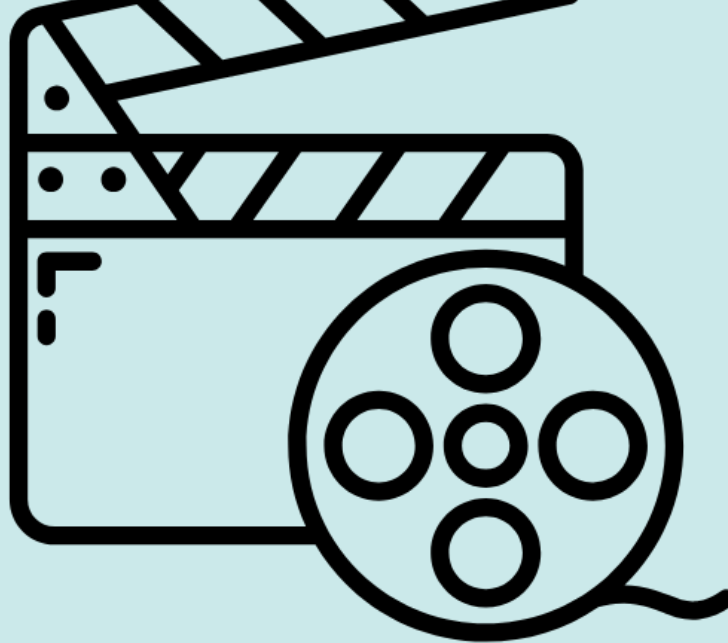
If I find inconsistencies in the data, I would review the source of the issue. In many cases, correcting inconsistent values is not possible unless I have access to the original or correct source data. Once the correct values are confirmed, I can update the records using the UPDATE command.

```
UPDATE table_name
SET field_name = 'Standard_Value'
WHERE field_name IN ('Variation1', 'Variation2', 'Variation3');
```




missing_length 	missing_replacement_cost 	missing_rating 	missing_last_update 	missing_special_features 	missing_fulltext 
bigint	bigint	bigint	bigint	bigint	bigint
0	0	0	0	0	0

```
CASE
  WHEN column_name IS NULL THEN 1
  ELSE 0
END
```


Data Immersion	Achievement III		
	SQL for Data Analysts		
	Task 3.6 By Ola Gaffarova	Table of content Page 5 of 10	Made for Rockbuster Stealth

One more option to check missing values
COUNT(column) automatically excludes NULLs.
COUNT(*) - COUNT(column) gives us the number of NULLs directly.

Query

Query History

2

COUNT(*) AS total_records,

3

COUNT(*) - COUNT(film_id) AS missing_film_id,

4

COUNT(*) - COUNT(title) AS missing_title,

5

COUNT(*) - COUNT(description) AS missing_description,

6

COUNT(*) - COUNT(release_year) AS missing_release_year,

7

COUNT(*) - COUNT(language_id) AS missing_language_id,

8

COUNT(*) - COUNT(rental_duration) AS missing_rental_duration,

9

COUNT(*) - COUNT(rental_rate) AS missing_rental_rate,

10

COUNT(*) - COUNT(length) AS missing_length,

11

COUNT(*) - COUNT(replacement_cost) AS missing_replacement_cost,

12

COUNT(*) - COUNT(rating) AS missing_rating,

13

COUNT(*) - COUNT(last_update) AS missing_last_update,

14

COUNT(*) - COUNT(special_features) AS missing_special_features,

15

COUNT(*) - COUNT(fulltext) AS missing_fulltext

16

FROM film;

17

Data Output

Messages

Notifications

≡

📄

▼

📋

▼

🗑️

📥

⬇️

📶

SQL

Showing rows: 1 to 1

Page No: 1 of 1

	total_records bigint	missing_film_id bigint	missing_title bigint	missing_description bigint	missing_release_year bigint	missing_language_id bigint	missing_rental_duration bigint
1	1000	0	0	0	0	0	0

missing_rental_rate bigint	missing_length bigint	missing_replacement_cost bigint	missing_rating bigint	missing_last_update bigint	missing_special_features bigint	missing_fulltext bigint
0	0	0	0	0	0	0

When dealing with missing values, there are two common approaches:
Ignoring Columns with Many Missing Values:
If a column has a large percentage of missing data, one practical solution is to exclude that column from your analysis. In SQL, this means simply omitting that column from your SELECT statement. It's also good practice to add a comment explaining why you are ignoring a column. It's also good practice to add a comment explaining why you are ignoring a column.

```
SELECT col1,  
       col2,  
       col4 -- col3 is excluded due to many missing values  
FROM tablename;
```

Imputing Missing Values:
Instead of excluding data, you can fill in missing values using statistical imputation methods. One common technique is to replace missing values with the column's average. This can be done in SQL with an UPDATE statement like this:

```
-- Impute missing values in col1 with the column's average  
UPDATE tablename  
SET col1 = (SELECT AVG(col1) FROM tablename)  
WHERE col1 IS NULL;
```




Summarizing the data

Query

Query History

```

1 SELECT COUNT (*) AS count_rows,
2 MIN (release_year) AS min_release_year,
3 MAX (release_year) AS max_release_year,
4 COUNT (release_year) AS count_release_year,
5 MIN (rental_duration) AS minimum_rental_duration,
6 MAX (rental_duration) AS maximum_rental_duration,
7 AVG (rental_duration) AS average_rental_duration,
8 COUNT (rental_duration) AS count_rental_duration,
9 MIN(rental_rate) AS minimum_rental_rate,
10 MAX (rental_rate) AS maximum_rental_rate,
11 AVG (rental_rate) AS average_rental_rate,
12 COUNT (rental_rate) AS count_rental_rate,
13 MIN (length) AS minimum_fiml_length,
14 MAX (length) AS maximum_film_length,
15 AVG (length) AS average_film_length,
16 COUNT (length) AS count_film_length,
17 MIN (replacement_cost) AS minimum_replacement_cost,
18 MAX (replacement_cost) AS maximum_replacement_cost,
19 AVG (replacement_cost) AS average_replacement_cost,
20 COUNT (replacement_cost) AS count_replacement_cost
21 FROM film;

```

Data Output

Messages

Notifications

+

📄

▼

📋

▼

🗑️

🗑️

📥

📥

📥

SQL

Showing rows: 1 to 1

Page No: 1 of 1

⏪

⏴

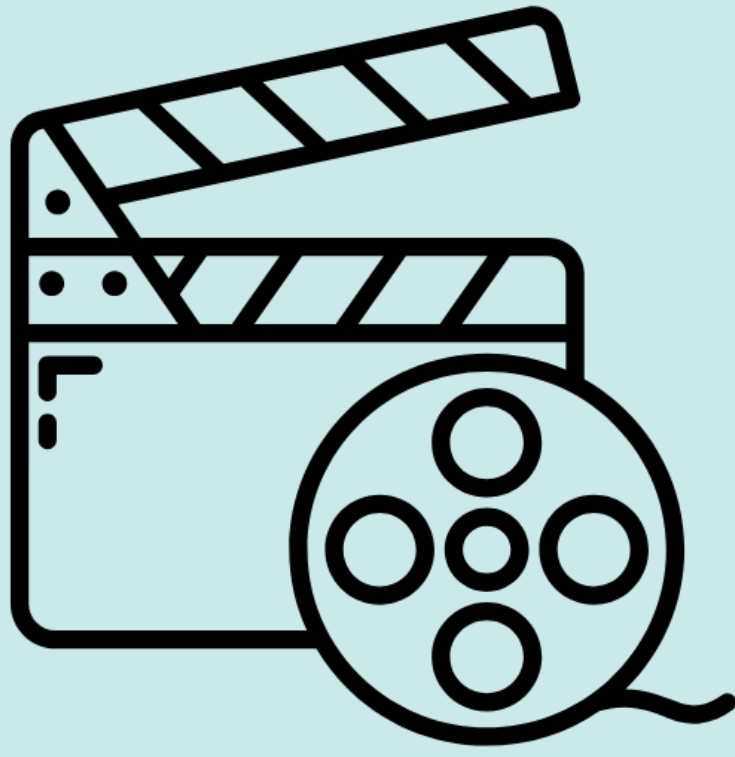
⏵

⏩

	count_rows bigint	min_release_year integer	max_release_year integer	count_release_year bigint	minimum_rental_duration smallint	maximum_rental_duration smallint	average_rental_duration numeric
1	1000	2006	2006	1000	3	7	4.9850000000000000

count_rental_duration bigint	minimum_rental_rate numeric	maximum_rental_rate numeric	average_rental_rate numeric	count_rental_rate bigint	minimum_fiml_length smallint	maximum_film_length smallint
1000	0.99	4.99	2.9800000000000000	1000	46	185

average_film_length numeric	count_film_length bigint	minimum_replacement_cost numeric	maximum_replacement_cost numeric	average_replacement_cost numeric	count_replacement_cost bigint
115.2720000000000000	1000	9.99	29.99	19.9840000000000000	1000



Checking if the table has non-uniform or duplicate data, or missing values

Query

Query History

1

2

3

4

5

6

7

8

SELECT

customer_id, store_id, first_name, last_name,

email,address_id, activebool, create_date,last_update,

active

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

≡+

▼

▼

SQL

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

CUSTOMER TABLE

Query

Query History

1

2

3

4

5

6

SELECT DISTINCT

customer_id, store_id, first_name,

last_name,

email,address_id, activebool,

create_date,last_update,

active

FROM

customer;

Data Output

Messages

Notifications

≡+

▼

▼

SQL

Showing rows: 1 to 599

Page No

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

1

357

1

Keith

Rico

keith.rico@sakilacustomer.org

362

true

2

171

2

Dolores

Wagner

dolores.wagner@sakilacustomer.org

175

true

3

139

1

Amber

Dixon

amber.dixon@sakilacustomer.org

143

true

4

471

1

Dean

Sauer

dean.sauer@sakilacustomer.org

476

true

5

594

1

Eduardo

Hiatt

eduardo.hiatt@sakilacustomer.org

600

true

6

401

2

Tony

Carranza

tony.carranza@sakilacustomer.org

406

true

7

157

2

Darlene

Rose

darlene.rose@sakilacustomer.org

161

true

8

154

2

Michele

Grant

michele.grant@sakilacustomer.org

158

true

9

530

2

Darryl

Ashcraft

darryl.ashcraft@sakilacustomer.org

536

true

10

493

1

Brent

Harkins

brent.harkins@sakilacustomer.org

498

true

11

542

2

Lonnie

Tirado

lonnie.tirado@sakilacustomer.org

548

true

12

566

1

Casey

Mena

casey.mena@sakilacustomer.org

572

true

13

186

2

Holly

Fox

holly.fox@sakilacustomer.org

190

true

14

128

1

Marjorie

Tucker

marjorie.tucker@sakilacustomer.org

132

true

15

466

1

Leo

Ebert

leo.ebert@sakilacustomer.org

471

true

16

494

2

Ramon

Choate

ramon.choate@sakilacustomer.org

499

true

17

178

2

Marion

Snyder

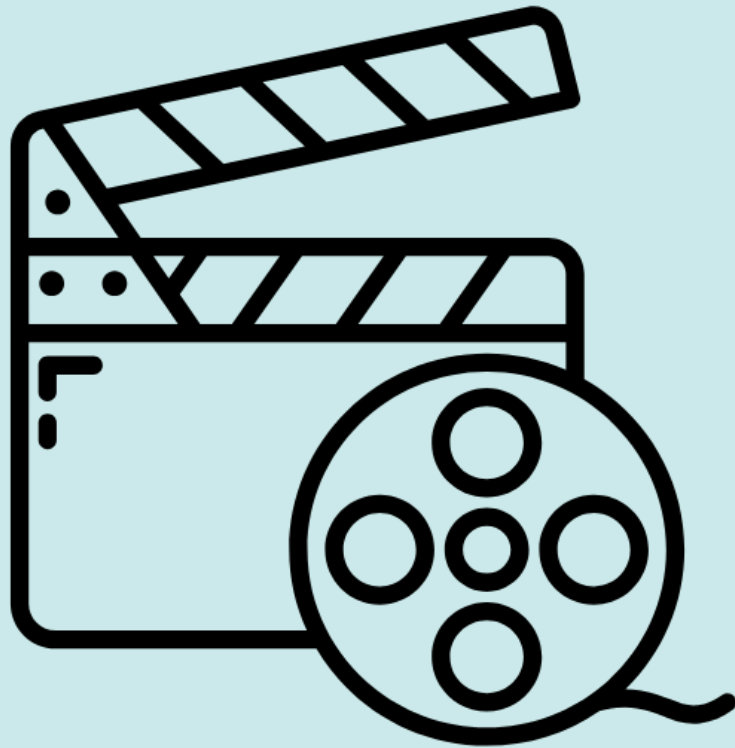
marion.snyder@sakilacustomer.org

182

true

Total rows: 599

Query complete 00:00:00.126



Query

Query History

1

2

3

4

5

6

7

8

9

10

11

12

SELECT

SUM(CASE WHEN customer_id IS NULL THEN 1 ELSE 0 END) AS Missing_customer_id,

SUM(CASE WHEN store_id IS NULL THEN 1 ELSE 0 END) AS Missing_store_id,

SUM(CASE WHEN first_name IS NULL THEN 1 ELSE 0 END) AS Missing_first_name,

SUM(CASE WHEN last_name IS NULL THEN 1 ELSE 0 END) AS Missing_last_name,

SUM(CASE WHEN email IS NULL THEN 1 ELSE 0 END) AS Missing_email,

SUM(CASE WHEN address_id IS NULL THEN 1 ELSE 0 END) AS Missing_address_id,

SUM(CASE WHEN activebool IS NULL THEN 1 ELSE 0 END) AS Missing_activebool,

SUM(CASE WHEN create_date IS NULL THEN 1 ELSE 0 END) AS Missing_create_date,

SUM(CASE WHEN last_update IS NULL THEN 1 ELSE 0 END) AS Missing_last_update,

SUM(CASE WHEN active IS NULL THEN 1 ELSE 0 END) AS Missing_active

FROM customer;

Data Output

Messages

Notifications

Showing rows: 1 to 1

Page No: 1

of 1

missing_customer_id	missing_store_id	missing_first_name	missing_last_name	missing_email	missing_address_id	missing_activebool	missing_create_date
bigint	bigint	bigint	bigint	bigint	bigint	bigint	bigint
0	0	0	0	0	0	0	0

missing_create_date	missing_last_update	missing_active
bigint	bigint	bigint
0	0	0

If we want to see which specific rows have missing data:

Query

Query History

1

2

3

4

5

6

7

8

9

10

11

12

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;

Data Output

Messages

Notifications

customer_id

store_id

first_name

last_name

email

address_id

[PK] integer

smallint

character varying (45)

character varying (45)

character varying (50)

smallint

activebool

create_date

last_update

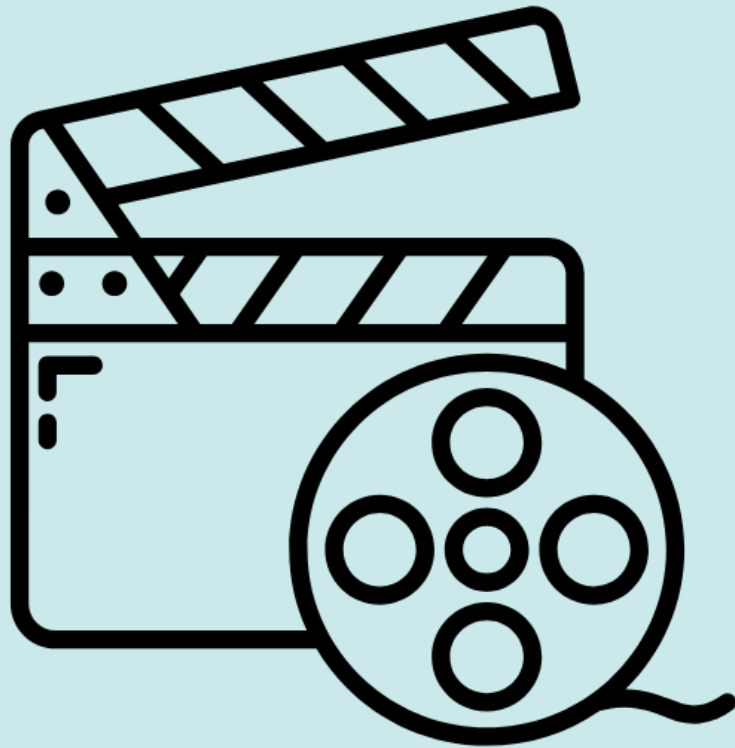
active

boolean

date

timestamp without time zone

integer



Query

Query History

1

2

3

4

5

6

7

8

9

10

11

12

13

SELECT

MIN(customer_id)

AS

Min_Customer_ID,

MAX(customer_id)

AS

Max_Customer_ID,

MIN(store_id)

AS

Min_Store_ID,

MAX(store_id)

AS

Max_Store_ID,

COUNT(store_id)

AS

Count_Store_ID,

MIN(active)

AS

Min_Active,

MAX(active)

AS

Max_Active,

COUNT(active)

AS

Count_Active,

AVG(active)

AS

Avg_Active,

MIN(last_update)

AS

Min_last_update,

MAX(last_update)

AS

max_last_update,

COUNT(last_update)

AS

Count_last_update

FROM

customer;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

Showing rows: 1 to 1

Page No: 1 of 1

	min_customer_id integer	max_customer_id integer	min_store_id smallint	max_store_id smallint	count_store_id bigint	min_active integer	max_active integer	count_active bigint
1	1	599	1	2	599	0	1	599

avg_active numeric	min_last_update timestamp without time zone	max_last_update timestamp without time zone	count_last_update bigint
0.97495826377295492487	2013-05-26 14:49:45.738	2013-05-26 14:49:45.738	599

Query

Query History

1

2

3

4

5

SELECT

MODE()

WITHIN GROUP (ORDER BY activebool)

AS

Mode_ActiveBool,

MODE()

WITHIN GROUP (ORDER BY first_name)

AS

Mode_First_Name,

MODE()

WITHIN GROUP (ORDER BY last_name)

AS

Mode_Last_Name,

MODE()

WITHIN GROUP (ORDER BY email)

AS

Mode_Email

FROM

customer;

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

Showing rows: 1 to 1

Page No: 1 of 1

	mode_activebool boolean	mode_first_name character varying	mode_last_name character varying	mode_email character varying
1	true	Jamie	Abney	aaron.selby@sakilacustomer.org

CUSTOMER TABLE

Data Immersion	Achievement III		
	SQL for Data Analysts		
	Task 3.6 By Ola Gaffarova	Table of content Page 10 of 10	Made for Rockbuster Stealth

From my experience, SQL is more effective for data profiling than Excel. While Excel is easier to use for small datasets and quick overviews, SQL is faster, more scalable, and better suited for handling large amounts of data. It also simplifies tasks like calculating statistics and checking for missing values without manual steps. Overall, SQL offers more efficiency and flexibility for data profiling.