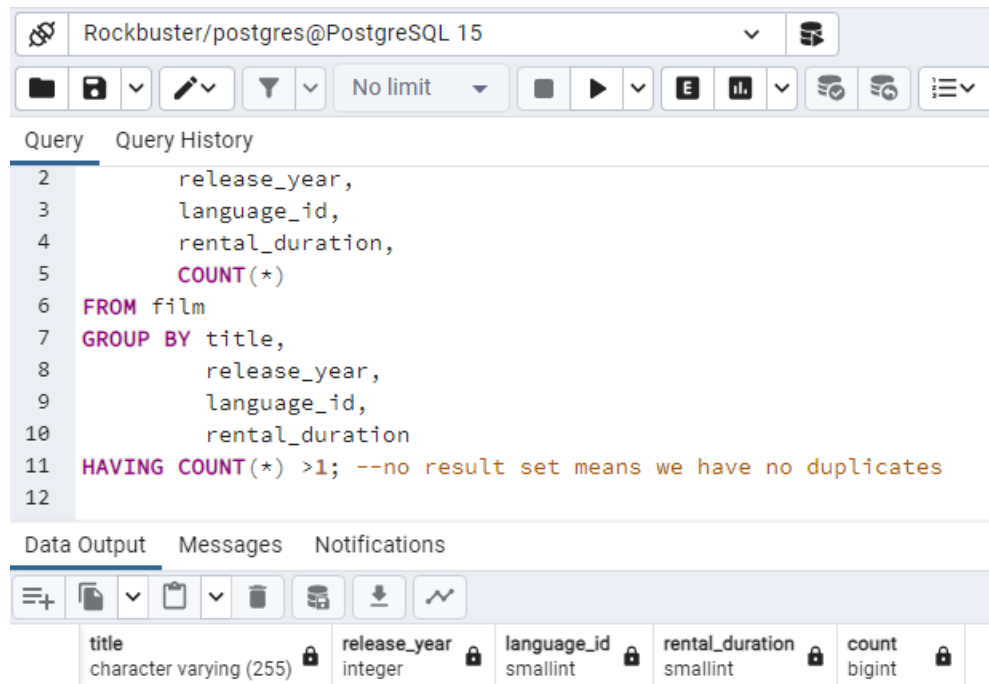


Cleaning data

DUPLICATE DATA

Query to check for **duplicate data** in the **film table**



The screenshot shows a PostgreSQL query editor interface. The top bar indicates the connection is 'Rockbuster/postgres@PostgreSQL 15'. Below the toolbar, the 'Query' tab is active, displaying the following SQL query:

```
2      release_year,  
3      language_id,  
4      rental_duration,  
5      COUNT(*)  
6 FROM film  
7 GROUP BY title,  
8         release_year,  
9         language_id,  
10        rental_duration  
11 HAVING COUNT(*) >1; --no result set means we have no duplicates  
12
```

Below the query editor, the 'Data Output' tab is active, showing the schema of the result set:

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

Query to check for **duplicate data** in the **customers table**

Rockbuster/postgres@PostgreSQL 15

No limit

Query Query History

```
1 SELECT first_name,
2     last_name
3     email,
4     address_id,
5     COUNT(*)
6 FROM customer
7 GROUP BY first_name,
8     last_name,
9     email,
10    address_id
11 HAVING COUNT(*) >1; --no result set means we have no duplicates
```

Data Output Messages Notifications

first_name	email	address_id	count
character varying (45)	character varying (45)	smallint	bigint

Due to the sensitivity and complexity of large databases, changes with big impact such as deleting records need layers of security and supervision. For that reason, as a data analyst, to deal and “delete” duplicates for our analyses I would create a view (virtual table) with only the unique records. In this case it isn't needed because there aren't duplicates.

NON-UNIFORM DATA

Query to check for **non-uniform data** in the **film** table

Rockbuster/postgres@PostgreSQL 15

Query Query History

```
1 SELECT DISTINCT replacement_cost
2 FROM film
3 GROUP BY replacement_cost
4 ORDER BY replacement_cost
```

Data Output Messages Notifications

	replacement_cost numeric (5,2)
1	9.99
2	10.99
3	11.99
4	12.99
5	13.99
6	14.99
7	15.99
8	16.99
9	17.99
10	18.99
11	19.99
12	20.99
13	21.99
14	22.99
15	23.99
16	24.99
17	25.99
18	26.99
19	27.99
20	28.99
21	29.99

Total rows: 21 of 21 Query complete 00:00:00.074

Rockbuster/postgres@PostgreSQL 15

Query Query History

```
1 SELECT DISTINCT release_year
2 FROM film
3 GROUP BY release_year
4 ORDER BY release_year
```

Data Output Messages Notifications

	release_year integer
1	2006

Rockbuster/postgres@PostgreSQL 15

Query Query History

```
1 SELECT DISTINCT rental_duration
2 FROM film
3 GROUP BY rental_duration
4 ORDER BY rental_duration
```

Data Output Messages Notifications

	rental_duration smallint
1	3
2	4
3	5
4	6
5	7

Query to check for **non-uniform data** in the **customers table**

The screenshot shows two database query windows side-by-side, both connected to 'Rockbuster/postgres@PostgreSQL 15'. Each window has a toolbar with icons for file operations, editing, and filtering, and a 'No limit' dropdown.

Left Window Query:

```
1 SELECT DISTINCT email
2 FROM customer
3 GROUP BY email
4 ORDER BY email
```

Left Window Data Output:

	email character varying (50)
1	aaron.selby@sakilacustomer.org
2	adam.gooch@sakilacustomer.org
3	adrian.clary@sakilacustomer.org
4	agnes.bishop@sakilacustomer.org
5	alan.kahn@sakilacustomer.org
6	albert.crouse@sakilacustomer.org
7	alberto.henning@sakilacustomer.org
8	alex.gresham@sakilacustomer.org
9	alexander.fennell@sakilacustomer.org
10	alfred.casillas@sakilacustomer.org

Right Window Query:

```
1 SELECT DISTINCT last_name
2 FROM customer
3 GROUP BY last_name
4 ORDER BY last_name
```

Right Window Data Output:

	last_name character varying (45)
1	Abney
2	Adam
3	Adams
4	Alexander
5	Allard
6	Allen
7	Alvarez
8	Anderson
9	Andrew
10	Andrews

I didn't find any non-uniform data. But if I had found some, I would fix it with the following query:

UPDATE table

```
SET column = ' _'  
WHERE column IN ('replacement word')
```

MISSING DATA

Query to check for **missing data** in the **film table**

The screenshot shows a PostgreSQL client interface with the following components:

- Top Bar:** Displays the connection name "Rockbuster/postgres@PostgreSQL 15".
- Toolbar:** Includes icons for file operations, query execution, and settings. A dropdown menu shows "No limit".
- Query Editor:** Contains the following SQL query:

```
1 SELECT *  
2 FROM film  
3 WHERE title = NULL
```
- Data Output Panel:** Shows the schema of the "film" table with the following columns:

film_id	title	description	release_year	language
[PK] integer	character varying (255)	text	integer	smallint

After checking all the columns in the film table with the query above, I didn't find missing data.

Query to check for **missing data** in the **customers table**

The same goes for the customer table.

In the case there was missing data, we have two options:

- 1- Replacing the missing data with average values.
- 2- If the percentage of missing data is too high we are better off ignoring that column completely for the analysis.

Summarize your data: descriptive statistics for both the film table and the customer table.

FILM TABLE

Numerical values

Rockbuster/postgres@PostgreSQL 15

No limit

Query Query History

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

SELECT

MIN(rental_rate)

AS min_rental_rate,

MAX(rental_rate)

AS max_rental_rate,

AVG(rental_rate)

AS avg_rental_rate,

MIN(rental_duration)

AS min_rental_duration,

MAX(rental_duration)

AS max_rental_duration,

AVG(rental_duration)

AS avg_rental_duration,

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,

COUNT(rental_rate)

AS count_rent_values,

COUNT(rental_duration)

AS count_rental_duration,

COUNT(length)

AS count_length,

COUNT(replacement_cost)

AS count_replacement_cost,

COUNT(*)

AS count_rows

FROM film;

Data Output Messages Notifications

	min_rent numeric	max_rent numeric	avg_rent numeric	min_duration smallint	max_duration smallint	avg_duration numeric
1	0.99	4.99	2.9800000000000000	3	7	4.9850000000000000

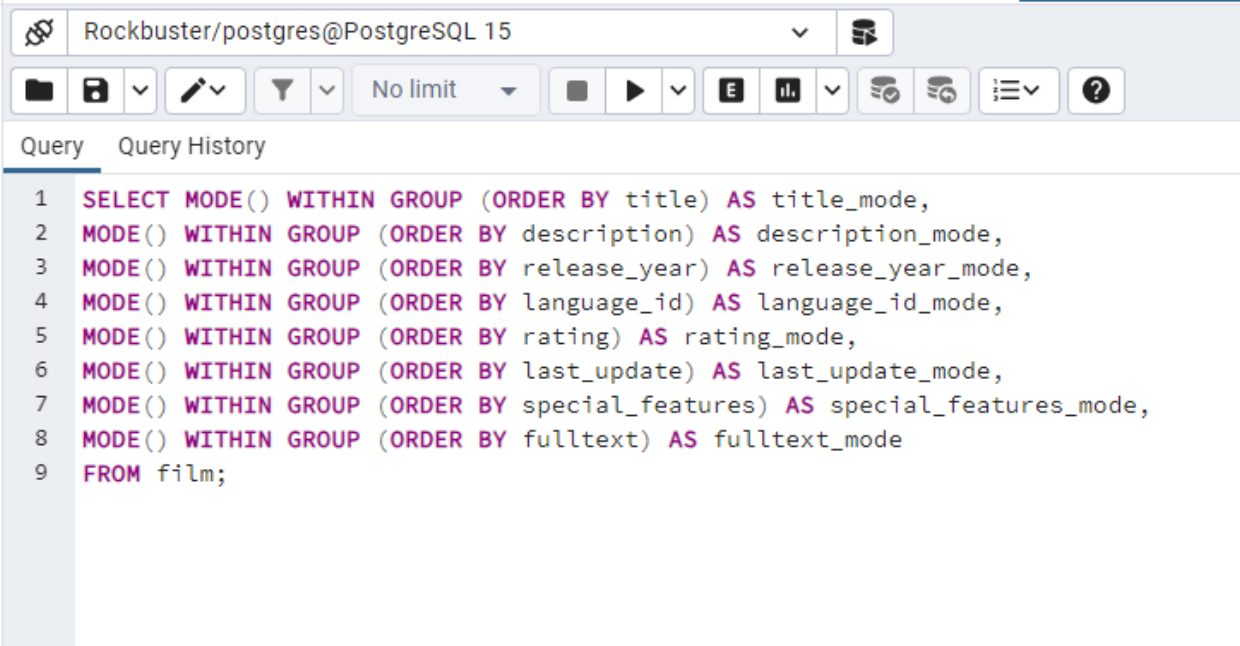
min_rental_rate	max_rental_rate	avg_rental_rate	count_rental_rate
0.99	4.99	2.98	1000

min_duration	max_duration	avg_duration	count_duration
3	7	4.985	1000

min_length	max_length	avg_length	count_length
46	185	115.272	1000

min_replacement_cost	max_replacement_cost	avg_replacement_cost	count_replacement_cost
9.99	29.99	19.984	1000

Non-numerical values



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'Rockbuster/postgres@PostgreSQL 15'. Below the title bar is a toolbar with icons for file operations, query execution, and other database functions. The main area displays a SQL query with line numbers 1 through 9. The query uses the 'MODE()' function within groups defined by 'ORDER BY' clauses for various attributes: title, description, release_year, language_id, rating, last_update, special_features, and fulltext. The results are aliased as *_mode. The query is executed against the 'film' table.

```

1 SELECT MODE() WITHIN GROUP (ORDER BY title) AS title_mode,
2 MODE() WITHIN GROUP (ORDER BY description) AS description_mode,
3 MODE() WITHIN GROUP (ORDER BY release_year) AS release_year_mode,
4 MODE() WITHIN GROUP (ORDER BY language_id) AS language_id_mode,
5 MODE() WITHIN GROUP (ORDER BY rating) AS rating_mode,
6 MODE() WITHIN GROUP (ORDER BY last_update) AS last_update_mode,
7 MODE() WITHIN GROUP (ORDER BY special_features) AS special_features_mode,
8 MODE() WITHIN GROUP (ORDER BY fulltext) AS fulltext_mode
9 FROM film;

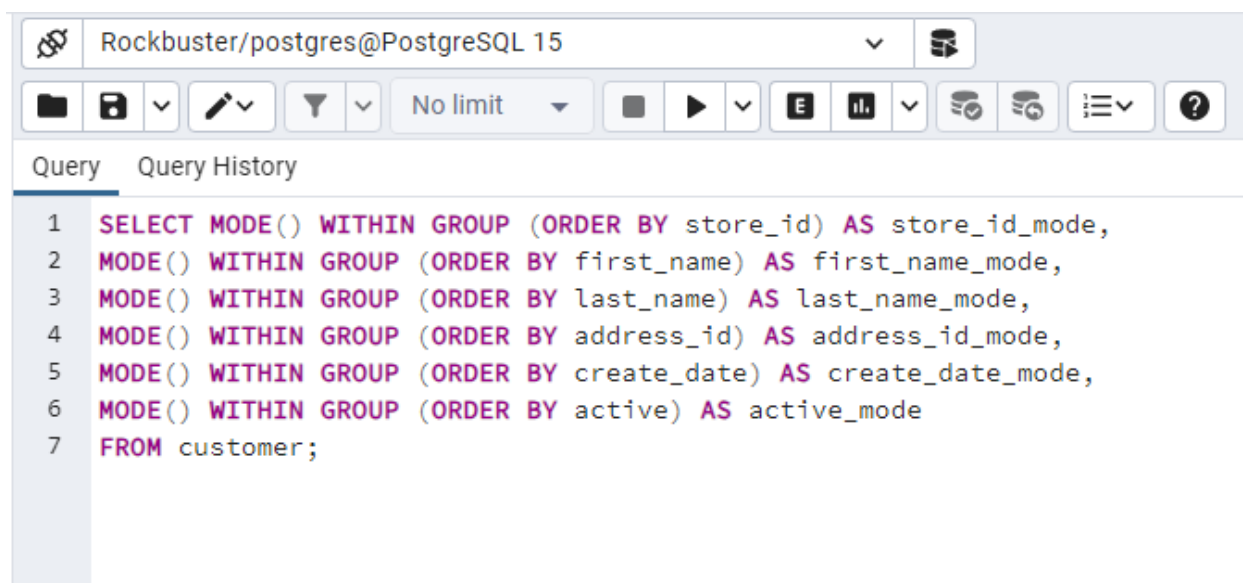
```

title_mode	description_mode	release_year_mode	language_id_mode	rating_mode	last_update_mode	special_features_mode	fulltext_mode
------------	------------------	-------------------	------------------	-------------	------------------	-----------------------	---------------

Academy Dinosaur	A Action-Packed Character Study of a Astronaut And a Explorer who must Reach a Monkey in A MySQL Convention	2006	1	PG-13	50:59.0	{Trailers,Commentaries,"Behind the Scenes"}	'baloon':19 'confront':14 'documentari':5 'feminist':8,11,16 'mile':2 'must':13 'spi':1 'thrill':4
------------------	---	------	---	-------	---------	---	---

CUSTOMER TABLE

Non-numerical values



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'Rockbuster/postgres@PostgreSQL 15'. The query editor has a toolbar with icons for file operations, query execution, and settings. The query text is as follows:

```

1 SELECT MODE() WITHIN GROUP (ORDER BY store_id) AS store_id_mode,
2 MODE() WITHIN GROUP (ORDER BY first_name) AS first_name_mode,
3 MODE() WITHIN GROUP (ORDER BY last_name) AS last_name_mode,
4 MODE() WITHIN GROUP (ORDER BY address_id) AS address_id_mode,
5 MODE() WITHIN GROUP (ORDER BY create_date) AS create_date_mode,
6 MODE() WITHIN GROUP (ORDER BY active) AS active_mode
7 FROM customer;

```

store_id_m ode	first_name_m ode	last_name_m ode	address_id_ mode	create_date_ mode	active_m ode
1	Jamie	Abney	5	2006-02-14	1