

## TASK 3.3

# SQL for Data Analysts

## Step 1

Processes × Rockbuster/postgres@PostgreSQL 16 × Rockbuster/postgres@PostgreSQL 16 × Untitled\* × Rockbuster/postgres@PostgreSQL 16 × Rockbuster/postgres@PostgreSQL 16\*

Rockbuster/postgres@PostgreSQL 16

No limit

Query Query History

Scratch Pad ×

SELECT category\_id, name  
FROM category

Data Output Messages Notifications

	category_id [PK] integer	name character varying (25)
1	1	Action
2	2	Animation
3	3	Children
4	4	Classics
5	5	Comedy
6	6	Documentary
7	7	Drama
8	8	Family
9	9	Foreign
10	10	Games
11	11	Horror
12	12	Music
13	13	New
14	14	Sci-Fi
15	15	Sports
16	16	Travel

Total rows: 16 of 16    Query complete 00:00:00.130    Ln 2, Col 14

## STEP 2

Rockbuster/postgres@PostgreSQL 16

Query

```
1 INSERT INTO category (name)
2 VALUES ('Thriller'), ('Crime'), ('Mystery'), ('Romance'), ('War')
3
```

Execute script (F5)

Scratch Pad

Data Output Messages Notifications

INSERT 0 5

Query returned successfully in 378 msec.

Total rows: 0 of 0 Query complete 00:00:00.378 Ln 3, Col 1

Rockbuster/postgres@PostgreSQL 16

Query Query History Scratch Pad

```
1 SELECT * FROM Category
2
```

Data Output Messages Notifications

	category_id [PK] integer	name character varying (25)	last_update timestamp without time zone
1	1	Action	2006-02-15 09:46:27
2	2	Animation	2006-02-15 09:46:27
3	3	Children	2006-02-15 09:46:27
4	4	Classics	2006-02-15 09:46:27
5	5	Comedy	2006-02-15 09:46:27
6	6	Documentary	2006-02-15 09:46:27
7	7	Drama	2006-02-15 09:46:27
8	8	Family	2006-02-15 09:46:27
9	9	Foreign	2006-02-15 09:46:27
10	10	Games	2006-02-15 09:46:27
11	11	Horror	2006-02-15 09:46:27
12	12	Music	2006-02-15 09:46:27
13	13	New	2006-02-15 09:46:27
14	14	Sci-Fi	2006-02-15 09:46:27
15	15	Sports	2006-02-15 09:46:27
16	16	Travel	2006-02-15 09:46:27

Total rows: 21 of 21 Query complete 00:00:00.263 Ln 1, Col 15

Rockbuster/postgres@PostgreSQL 16

Query Query History Scratch Pad

```
1 SELECT * FROM Category
```

Data Output Messages Notifications

	category_id [PK] integer	name character varying (25)	last_update timestamp without time zone
2	2	Animation	2006-02-15 09:46:27
3	3	Children	2006-02-15 09:46:27
4	4	Classics	2006-02-15 09:46:27
5	5	Comedy	2006-02-15 09:46:27
6	6	Documentary	2006-02-15 09:46:27
7	7	Drama	2006-02-15 09:46:27
8	8	Family	2006-02-15 09:46:27
9	9	Foreign	2006-02-15 09:46:27
10	10	Games	2006-02-15 09:46:27
11	11	Horror	2006-02-15 09:46:27
12	12	Music	2006-02-15 09:46:27
13	13	New	2006-02-15 09:46:27
14	14	Sci-Fi	2006-02-15 09:46:27
15	15	Sports	2006-02-15 09:46:27
16	16	Travel	2006-02-15 09:46:27
17	17	Thriller	2024-04-11 10:46:54.687271
18	18	Crime	2024-04-11 10:46:54.687271
19	19	Mistery	2024-04-11 10:46:54.687271
20	20	Romance	2024-04-11 10:46:54.687271
21	21	War	2024-04-11 10:46:54.687271

Total rows: 21 of 21 Query complete 00:00:00.263 Ln 1, Col 15

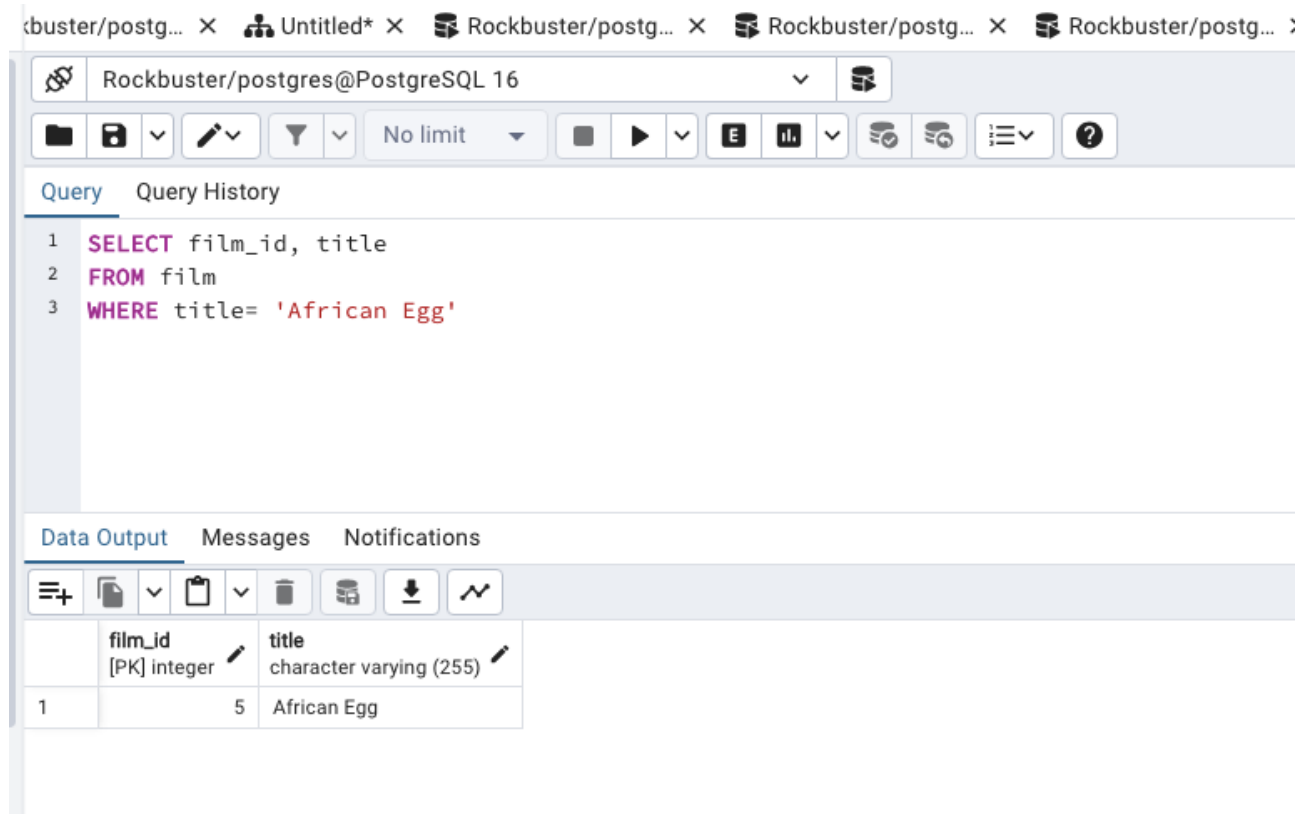
```
CREATE TABLE category
(
    category_id integer NOT NULL DEFAULT nextval('category_category_id_seq'::regclass),
    name text COLLATE pg_catalog."default" NOT NULL,
    last_update timestamp with time zone NOT NULL DEFAULT now(),
    CONSTRAINT category_pkey PRIMARY KEY (category_id)
);
```

in the category table, we're setting up some rules for how the data should look and behave. First off, we've got the `category_id`, which is like a special ID number for each category. We're saying it can't be left blank (NOT NULL) and it has to be unique for each category (PRIMARY KEY). If we don't give it a number when we add a new category, it'll automatically get the next available number.

Then there's the `name` column, which holds the name of each category. We're saying it can't be left blank either (NOT NULL). We want to make sure each category has a name. Lastly, we've got the `last_update` column, which keeps track of when each category was last updated. This one also can't be left blank (NOT NULL). And if we don't give it a specific time, it'll just use the current time as the default.

So, basically, these rules help keep our data organized and make sure it's accurate. They stop us from accidentally leaving out important information or adding duplicate categories. It's like having a checklist to make sure everything is in order.

### STEP 3



The screenshot shows a PostgreSQL client interface with a query editor and a results pane. The query editor contains the following SQL query:

```
1 SELECT film_id, title
2 FROM film
3 WHERE title= 'African Egg'
```

The results pane shows the output of the query, which is a single row with the following data:

film_id	title
5	African Egg

Query Query History

```
1 SELECT film_id, Category_id
2 FROM film_category
3 WHERE film_id= 5
```

Data Output Messages Notifications

1

	film_id [PK] smallint 	category_id [PK] smallint 
1	5	8

## STEP 4

The screenshot shows the pgAdmin interface with the following details:

- Toolbar:** Includes icons for file operations, query execution (a play button icon is highlighted), and other database management functions.
- Query Editor:** Contains the SQL query:
 











```
1 DELETE FROM Category
2 Where name= 'Mistry'
```
- Execution:** A tooltip "Execute script (F5)" is visible over the play button icon.
- Messages Tab:** The "Messages" tab is active, displaying the output:
 


```
DELETE 1

Query returned successfully in 240 msec.
```
- Other Tabs:** "Query History" and "Scratch Pad" are also visible.

stg... X Rockbuster/postg... X Rockbuster/postg... X Rockbuster/postgres@PostgreSQL 16\* X R

Rockbuster/postgres@PostgreSQL 16



Query  Query History

```
1 UPDATE film_category
2 SET category_id = 15
3 WHERE film_id= 5
```











Data Output Messages Notifications

UPDATE 1

Query returned successfully in 364 msec.

stg... X Rockbuster/postg... X Rockbuster/postgres@PostgreSQL 16\* X Rockbuster/pc










Rockbuster/postgres@PostgreSQL 16



Query Query History

```
1 SELECT film_id, Category_id
2 FROM film_category
3 WHERE film_id= 27
4
```

Data Output Messages Notifications

         		
	film_id [PK] smallint	category_id [PK] smallint
1	27	15

To see if it's still there

stg... X Rockbuster/postg... X Rockbuster/postg... X Rockbuster/postg... X Rockbuste

Rockbuster/postgres@PostgreSQL 16

Query Query History

```
1 SELECT category_id, name
2 FROM category
```

Data Output Messages Notifications

	category_id [PK] integer	name character varying (25)
1	1	Action
2	2	Animation
3	3	Children
4	4	Classics
5	5	Comedy
6	6	Documentary
7	7	Drama
8	8	Family
9	9	Foreign
10	10	Games
11	11	Horror
12	12	Music
13	13	New
14	14	Sci-Fi
15	15	Sports
16	16	Travel
17	17	Thriller
18	18	Crime
19	20	Romance
20	21	War

Total rows: 20 of 20 Query complete 00:00:00.153

## STEP 5

ANSWER: When it comes to using Excel or SQL for steps 1 to 4, there's a clear difference in how they work. Excel is easier to understand visually. You can easily add new categories like Mystery or Romance just by adding rows and filling them in. But with SQL, you need to know where the data is and how to use commands to work with it. Plus, SQL helps keep your data safe because it won't let you make changes without permission. On the other hand, in Excel, you might accidentally delete something important and run into problems.