Query Editor Query History

```
SELECT category_id, name
FROM category
```

Data Output Explain Messages

	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

Step 2

```
INSERT INTO category
(name)
VALUES('Thriller'),('Crime'),('Mystery'),('Romance'),('war')
```

Data Output Explain **Messages** Notifications

```
INSERT 0 5

Query returned successfully in 100 msec.
```

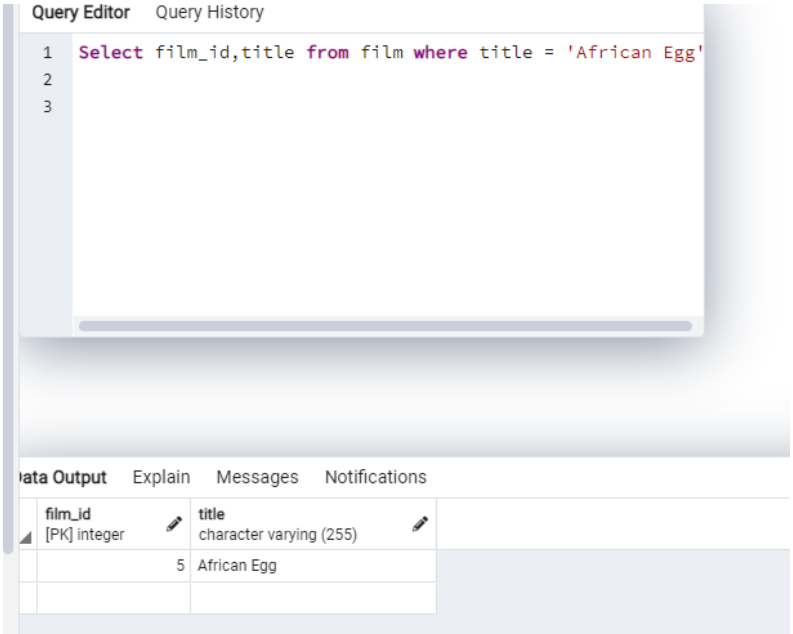
```
CREATE TABLE category
(
  category_id integer NOT NULL DEFAULT nextval('category_category_id_seq'::regclass),
  name text COLLATE pg_catalog."default" NOT NULL,
  last_update timestamp with time zone NOT NULL DEFAULT now(),
  CONSTRAINT category_pkey PRIMARY KEY (category_id)
);
```

1. **UNIQUE Constraint:** This ensures that every value in a column is unique. It's helpful if you want to prevent duplicate values from being entered into a column. Category_id can be a unique constraint.

2. **NOT NULL Constraint:** This ensures that a column can't have any empty or missing values. Use NOT NULL if your table contains columns that should never be empty. When this constraint is applied correctly, an error message will appear if someone try to insert empty values. Here we have category_id, name, and time for last update with not null constraint.

3. **PRIMARY KEY Constraint:** The primary key gives each record in a table a unique ID. The primary key column can't contain any null or duplicate values. We have category_pKey primary key constraint here.

Step 3 Update



```
SELECT film_id, category_id from film_category where film_id = 5
```

ta Output Explain Messages Notifications

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

Query Editor Query History

```
update film_category set category_id = 17  where film_id = 5
```

Data Output Explain **Messages** Notifications

```
UPDATE 1

Query returned successfully in 130 msec.
```

Step 4 Delete

Query Editor Query History

```
delete from category where name = 'Mystery'
```

Data Output Explain **Messages** Notifications

```
DELETE 0

Query returned successfully in 85 msec.
```

Step 5:

Based on what you've learned so far, think about what it would be like to complete steps 1 to 4 with Excel instead of SQL

There is no doubt that SQL is very fast and can be very helpful if we are dealing with large amount of data but I still need to do lots of practice to master SQL. In excel filter method would be useful to get all this information. We can use pivot table and then use slicer method to find out whatever we need to know. But in SQL we just get the information that have been asked for.

For example if we need to know how many films in the film category we can run query

select count(film_id) from film

It will show how many films in film table with just a number.

Output was 1000;

But if I will be trying same thing in excel It will show all the data with all the films and count will be on right hand side at bottom.

Although It might not be a big difference but for the large datasets it could take longer time to load all the data but SQL makes it more faster.

Bonus Task