

### Step 1

**Your first task is to find out what film genres already exist in the category table.**

Output:

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

Query

Query History

```

1 SELECT
2     name,
3     category_id
4 FROM
5     category
6

```

Data output

Messages

Notifications

+

📄

▼

🗑️

📁

⬇️

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

Total rows: 16 of 16

Query complete 00:00:00.167

## Step 2

**Insert the following genres into the Category Table: Thriller, Crime, Mystery, Romance, and War.**

Output:

```
INSERT INTO category (name)
VALUES ('Thriller')
```

I tried to insert all the values at the same time, but PostgreSQL gives me the following error:

*“ERROR: INSERT has more expressions than the target columns*

*LINE 4:               'Mystery'”*

Query	Query History
1	
2	<b>INSERT INTO</b> category( <b>name</b> )
3	<b>VALUES</b> ('Crime',
4	'Mystery',
5	'Romance',
6	'War')

Data output	Messages	Notifications
ERROR: INSERT has more expressions than target columns		
LINE 4:               'Mystery',		
^		
SQL state: 42601		
Character: 55		

The values were manually inserted:

1	
2	<b>INSERT INTO</b> category( <b>name</b> )
3	<b>VALUES</b> ('Crime')

Data output	Messages	Notifications
INSERT 0 1		
Query returned successfully in 32 msec.		

Resulting in the following categories:

Query Query History

```
1 SELECT*
2 FROM category
```

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
17	17	Thriller	2022-06-30 15:54:58.228341
18	18	Crime	2022-06-30 15:58:40.723144
19	19	Mystery	2022-06-30 15:59:09.548346
20	20	Romance	2022-06-30 15:59:19.098093
21	21	War	2022-06-30 16:04:09.789501

Total rows: 21 of 21 Query complete 00:00:00.055

### What do these constraints do exactly? Why are they important?

```
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)
);
```

There are two types of constraints in this query:

1. *NOT NULL*—constraint makes sure there are no missing values in the selected table's columns.  
*category\_id*, to make sure a number value is inserted in this column  
*name*, to make sure a name value is inserted in this column  
*last\_update*, to make sure a date value is inserted in this column
2. *PRIMARY KEY*—act as a primary identifier for an entire row in a table  
*category\_id*, to make all the values in this column a PRIMARY KEY

### Step 3

The genre for the movie *African Egg* needs to be updated to thriller.

Output to find *film\_id* *African Egg*:

```
SELECT film_id
FROM film
WHERE title = 'African Egg'
```

Query Query History

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

Data output Messages Notifications

	film_id [PK] integer
1	5

Then *category\_id*:

```
SELECT film_id, category_id
FROM film_category
WHERE film_id = 5
```

Query Query History

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

Data output Messages Notifications

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

Then the UPDATE output as follows:

```
UPDATE film_category
SET category_id = 17
WHERE film_id = 5
```

Query Query History

```
1 UPDATE film_category
2 SET category_id = 17
3 WHERE film_id = 5
4
```

Data output Messages Notifications










UPDATE 1

Query returned successfully in 139 msec.

Query Query History

```
1 SELECT
2     film_id, category_id
3 FROM
4     film_category
5 WHERE
6     film_id = 5
```

Data output Messages Notifications

						
	film_id [PK] smallint 	category_id [PK] smallint 				
1	5	17				

#### Steps 4

Since there aren't many movies in the mystery category, a decision to remove it from the category table has been made. Write a DELETE command to do so.

**DELETE FROM**

**category**

**WHERE**

**category\_id = 17**

and/or

**DELETE FROM**  
**category**  
**WHERE**  
**name = 'Thriller'**

Query Query History Scratch Pad x

```
1 DELETE FROM
2     category
3 WHERE
4     category_id = 17
```

Data output Messages Notifications

ERROR: update or delete on table "category" violates foreign key constraint "film\_category\_category\_id\_fkey" on table "film\_category"  
DETAIL: Key (category\_id)=(17) is still referenced from table "film\_category".  
SQL state: 23503

Query Query History Scratch Pad x

```
1 DELETE FROM
2     category
3 WHERE
4     name = 'Thriller'
```

Data output Messages Notifications

ERROR: update or delete on table "category" violates foreign key constraint "film\_category\_category\_id\_fkey" on table "film\_category"  
DETAIL: Key (category\_id)=(17) is still referenced from table "film\_category".  
SQL state: 23503

Since there is a Foreign Key constraint, the *category\_id* tuple 17 and 'Thriller' cannot be deleted. The error is as follows:

"ERROR: update or delete on table "category" violates foreign constraint "film\_category\_id\_fkey on table "film\_category""

## Step 5

What it would be like to complete steps 1 to 4 with Excel instead of SQL. Are there any pros and cons to using SQL?

For the **Step 1**, first a pivot table would have to be created to see the *names* and *category\_id* listed together by counting them.

For **Step 2**, first filtering the table to check on the order of the values (alphabetical or by ID), then manually entering the new genres into the list, assign an ID for each category, and using date and time of the moment the new genre is being added up into the list. Then I need to restrict data entry in the specified cells or using DATA VALIDATION to constraint the data entry in the selected cells.

For **Step 3**, filtering the tuple or record and then manually change the information.

For **Step 4**, filtering the requested tuples or records with “Thriller” value and delete them.

All in all, even though some steps are not difficult per se in Excel, they take longer and limit our time in creating, reading, updating and/or deleting information.

### **Bonus**

**Fix the query's typos.**

```
CREATE TBL 3EMPLOYEES
{
employee_id VARINT(30) NOT EMPTY
name VARCHAR(50),
contact_number VARCHAR(30) ,
designation_id INT,
last_update TIMESTAMP NOT NULL DEF now()
CONSTRAIN employee_pkey PRIMARY KEY (employee_id)
}
```

Please, see the 2 screenshots below:



Query Query History

```
1 CREATE TABLE employee
2     (employee_id INT NOT NULL,
3       name VARCHAR(50),
4       contact_number INT NOT NULL,
5       designation_id INT NOT NULL,
6       last_update INT NOT NULL,
7       employee_pkey INT PRIMARY KEY);
```

Data output Messages Notifications

CREATE TABLE

Query returned successfully in 151 msec.

Query   Query History

```
1
2 SELECT *
3 FROM employee
```

Data output   Messages   Notifications















	<b>employee_id</b> integer	<b>name</b> character varying (50)	<b>contact_number</b> integer	<b>designation_id</b> integer	<b>last_update</b> integer	<b>employee_pkey</b> [PK] integer
--	-------------------------------	---------------------------------------	----------------------------------	----------------------------------	-------------------------------	--------------------------------------