1. PostgreSQL Data Operations - Insert, Update, Delete

Contents

[1. Loading Sample Data 1](#_Toc23740)

[2. Getting Started 2](#_Toc2046)

[3. Scenario 1 Task 1: Update Table With New Data 2](#_Toc31036)

[4. Scenario 1 Task 2: Insert Data Into Table 3](#_Toc933)

[5. Scenario 4: Delete Rows From Table 4](#_Toc25709)

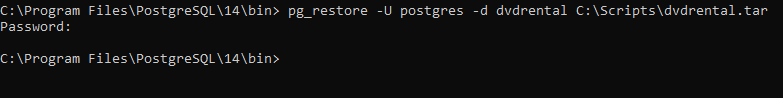
[6. Summary 4](#_Toc12984)

# 1. Loading Sample Data

=>slides: Pg. 1

We will start pgAdmin IV, which is the graphical interface for Postgres and install sample database pagilia. First we will go to browser and enter URL type **https://neon.tech/postgresql/postgresql-getting-started/postgresql-sample-database**. Click Enter, and we'll bring you to a website where you can download sample database pagilia. Pagilia sample database is best known very famous DVD rental database of MySQL commonly known as Sakila. It has pretty much same structure. It contains 15 tables, 1 trigger, 7 views, 8 functions, 1 domain, and 13 sequences. You can see ER model of DVD rental database over here.

In Command Prompt first we will change the context of our current directory to the bin directory where we have installed Postgres. Here is the change directory command where I will change the context of execution to this folder. Click Enter, and now context of Command Prompt is changed. Next we will be using command pg\_restore to create new sample database. However, there is one small task, which we have to do before we execute pg\_restore command. We are to go back to pgAdmin and over here create a new database. I will click on the icon of execute SQL queries, and it will open this SQL Editor. Over here I will type CREATE DATABASE dvdrental and click on Execute by clicking on this Play button. The message suggests query returned successfully. That means dvdrental database is now created successfully on my Postgres server. To see that let's minimize this. Now over here in pgAdmin Object explorer click on Refresh, and it will display new database dvdrental. Click over here, and it will expand various objects. Further expand Schemas and expand public schema. Under public schema you will see count of Tables is 0. This is because we have just created this dvdrental database, and we have not populated the same. To populate this data now once again we'll go back to Command Prompt and execute our pg\_restore command.



pg\_restore -U postgres -d dvdrental D:\Test\Database\PostgreSQL\_Data\_Operations-Insert\_Update\_Delete\dvdrental.tar

Here is our restore command, pg\_restore -U for username and -d is for database dvdrental. Remember, we have just created this database a few seconds ago. Next is the part where our RAR file for dvdrental sample database is located. Now click Enter, and within a few seconds you will see command prompt once again. There is no error. That means our sample database was restored successfully. To see that let's go back to pgAdmin, and now once again click on Refresh. As soon as we click on Refresh we notice the count of Tables is now changed to 15 from 0, as well as a few other objects as well. Further expand Tables, and now there are 15 different tables that exist. Click on table actor, and on the right side we will see properties of this table. Once again click on execute SQL queries. On this screen we'll execute our very first query. Remember, context of our query execution is dvdrental database with Postgres user on localhost listening on port 5432. That is listed in this dropdown over here. Now I will write our very first SELECT statement, SELECT \* FROM film. Now click on Execute. Within a few seconds under Output pane you will see result of this query. That's it

# Getting Started

=>slides: Pg. 3

In this module we are going to take our understanding of Postgres to the next level.We will see how we can update, as well as insert data in database tables. We will also see how to delete data from database table. For this we will use again real-world scenarios.

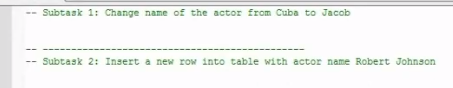
=>slides: Pg. 4

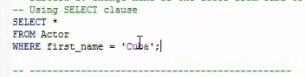
The scenario is very simple.We have Rahul and Mike. Rahul is a senior database administrator, and Mike is a junior database administrator. Between them we will recreate various scenarios from the real world. Let's start with our very first scenario.

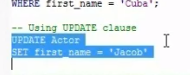
# Scenario 1 Task 1: Update Table With New Data

=>slides: Pg. 5

In this scenario Rahul gives a task to Mike. Mike has to update and insert data into the table.This scenario has two subtasks. First was to change the name of one of the actors from Cuba to Jacob, and in the second subtask Mike has to insert a new row for actor Robert Johnson. Well, let's see how we can help Mike to achieve this task. For that let's go to SQL Editor of pgAdmin IV.



Here we are in SQL Editor, and we have two subtasks to complete in this scenario. First Mike has asked to change the name of one of the actors. The name of the actor is Cuba. For that first let's retrieve all the data for actor who has first name as Cuba. Here is the SELECT statement for that: 

SELECT \* FROM Actor WHERE first\_name is Cuba. Let's select this and click on Execute. There are three records with actor name Cuba. The records are with actor\_id 15, 118, and 189. Now our next task is to update the name of the actor from Cuba to Jacob. For that we have to write an UPDATE statement. 

Let's write this UPDATE statement step-by-step. First we write keyword UPDATE, and right after that we write the table name which we want to update. Here the table which we want to update is Actor. Next we specify the column which we want to for that, the keyword is SET, name of the column first\_name, and the value which we want to update with. For example, over here we want to update column first\_name with the name Jacob. Now if we execute the query, which we have highlighted in SQL Editor, it will go ahead and change the name of every single first name of actor in our table. This is because we do not have a WHERE clause. In our task we are asked to change the name of actor from Cuba to Jacob; hence, we have to write a WHERE condition. The construction of the WHERE condition is identical to what we write in the SELECT statement.



Let's copy the same WHERE condition and paste it over here. Now we have the entire UPDATE statement. Let's select the statement and click on Execute. If everything is correct, this statement should update three rows, which are displayed on the screen. Here it is. The message is query returned successfully. That means our query was executed without any error. Three rows affected. This is what we had expected. And this query was executed in 31 ms. That's pretty fast. Now let's go back to our SELECT statement and see if we have any actor with the name Cuba.



We will execute the same statement we had executed a few seconds ago. Click on Execute, and our result set is empty. This is because we have updated every single record where actor's first name is Cuba. We have changed that to Jacob. That means if we execute this SELECT query with the word Jacob we should retrieve the same three rows which have seen earlier. Let's check that.

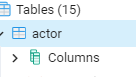


Type here Jacob instead of Cuba, and now execute same select statement. Click on Execute, and here it is. We have the same three rows which we have seen earlier. A close look at the very first column of actor\_id we can notice actor\_id is the same as what we have seen earlier. They are 15, 118, and 189. Well, with the help of this script Mike can definitely change various columns of table in Postgres database. Then next task was to insert a new row into table with actor name Robert Johnson. Well, we will see how we can do that in the next section.

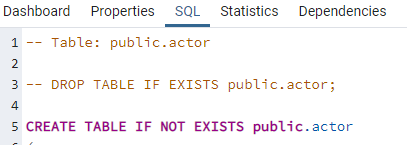
# Scenario 1 Task 2: Insert Data Into Table

=>slides: Pg. 5

Here we are in SQL Editor where we left off from previous section. This time our subtask is to insert a new row in the table actor where actor name is Robert Johnson. For this we have to write an INSERT statement. Let's write INSERT statement based on table Actor. We already know that table has actor\_id, first\_name, last\_name, and last\_update date as a column, but when we write INSERT statement it is a little bit trickier than just looking at four columns. Before we write INSERT statement we need to see schema of table Actor. Let's go to our Object Explorer and observe schema of table actor.



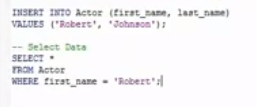
Here we are in pgAdmin IV in object browser. On the left side we have database dvdrental, and here is table actor. Table actor is already selected,



and on the right side we can see in the SQL pane schema of TABLE actor. Actor\_id is of data type integer, and by default the value is generated from sequence. That means actor\_id is auto implemented every time a new row is inserted in table actor. Next is first\_name. The data type of the same is character varying up to 45. That means the first name can have a string which can be as big as 45 characters. Same thing is for last\_name as well. Right after that we have last\_update. It also has DEFAULT value of now. That means every single time any record is inserted in column first\_name and last\_name at the same time this particular column is auto updated with current timestamp. Well, this is very good information for us. In summary, we have four columns in table actor, and two of them are auto populated. We just have to insert value in first\_name column, as well as last\_name column, and last\_update column will be auto populated. This is very good information for us. Now let's go back to our SQL Editor and write INSERT statement. Here we are back in our SQL Editor.



Let's write INSERT statement INSERT INTO table Actor, and over here we'll specify name of the columns where we want to insert our data first\_name and last\_name. Next we will write keyword VALUES, and over here we will type values which we want to insert, Robert and Johnson. We successfully completed construction of INSERT statement. Now let's select this and click on Execute. Query returned successfully. One row affected, and it was executed in 10 ms. That means this particular query was executed successfully, and it has inserted one row.



Well, to check that let's write SELECT statement right after it and check if there is any actor with the name Robert. SELECT \* FROM Actor WHERE first\_name is Robert. Now let's select this and click on Execute. Query immediately returned us a result with one row. The first name of actor is Robert, and last name is Johnson. This is what we have just inserted, and it is confirmed by the timestamp, which we see in the column over here. Actor\_id was also auto-generated in table Actor. Well, what we have seen over here is how we can write SQL and do UPDATE, as well as INSERT.

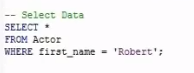
# Scenario 2: Delete Rows From Table

=>slides: Pg. 6

Scenario 2. So far Rahul was giving very difficult tasks to Mike, and Mike was successfully able to complete them. This time Rahul has decided to give a very simple task to Mike. He asks Mike to delete a row with the specific condition from table actor. Mike has to delete all the rows where first name of the actor is Robert. If you remember, in the last section we have just inserted a record where the first name of the actor was Robert. Let's see how Mike can complete this task with the help of SQL Editor of pgAdmin IV. Here we are in SQL Editor.



We have to delete all the rows where first name of the actor is Robert.



First let's check in Actor table how many rows are there where first name of the actor is Robert. SELECT \* FROM Actor WHERE first\_name is equal to Robert. Let's select this statement and click on Execute. The result set indicates that we have only one row where name of the actor is Robert. Now we have to delete this record set. Let me teach you a very simple trick. Whenever you have to delete any data, first make sure you write a SELECT statement, and once you get necessary result set based on your business logic you can quickly turn the same SELECT statement to a DELETE statement. Now in our case this is our necessary result set.



We will take the same SELECT statement, copy right below that, and replace SELECT \* with DELETE keyword. That's it. Our DELETE script is ready. Select this statement, and click on Execute. Query returned successfully. One row affected. That means this particular query has deleted one row.



Let's check from our Actor table by executing the SELECT statement if the row which we have seen a few moments ago is deleted or not. Click on Execute, and our result set is empty. That means our DELETE statement has successfully deleted a row where first name was Robert. Remember, if you ever execute DELETE statement without WHERE clause, it will delete all the rows from your table. Be very diligent and vigilant when you execute DELETE command in your SQL Editor. Well, with this we complete all the scenarios and tasks given by Rahul. I'm very confident that Rahul is happy with the progress of Mike. Now let's see a summary of what we have learned in this module.

# Summary

=>slides: Pg. 7

Well, with the help of real-world scenarios, which we have created between Mike and Rahul, we have learned how to update, insert, as well as delete data in this module. The most important learning which we had was basic structure of SELECT statement. We learned that with the help of basic structure of SELECT statement we can conditionally update and delete data from table. Here is the basic structure one more time: SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY. So far we were returning data from a single table.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*