# STORED PROCEDURE

Stored Procedure is a prepared SQL(Structure Query Language) code that you can save so the code can be reused over and over again.

You can also pass parameters to stored procedures so that the procedures can act based on the parameter value(s) that is passed.

To define a new stored procedure you use the create procedure statement the following illustrates the basic syntax of a create procedure statement:

```
create [or replace] procedure procedure_name(parameter_list)
language plpgsql
as $$
declare
-- variable declaration
begin
-- stored procedure body
end; $$
```

In this syntax:

- First, specify the name of the stored procedure after the create procedure keywords.
- Second, define parameters for the stored procedure. A stored procedure can accept zero or more parameters.
- Third, specify plpgsql as the procedural language for the stored procedure. Note that you can use other procedural languages for the stored procedure such as SQL, C, etc.
- Finally, use the dollar-quoted string constant syntax to define the body of the stored procedure.

A stored procedure does not return a value. You cannot use the return statement with a value inside a stored procedure like this:

*return expression;*

However, you can use the return statement without the expression to stop the stored procedure immediately:

*return;*

If you want to return a value from a stored procedure, you can use parameters with the inout mode.

## Postgresql create procedure statement examples:

We will use the following accounts table for illustration:

```
drop table if exists accounts;

create table accounts (
    id int generated by default as identity,
    name varchar(100) not null,
    balance dec(15,2) not null,
    primary key(id)
);

insert into accounts(name,balance)
values('Bob',10000);

insert into accounts(name,balance)
values('Alice',10000);
```

The “SELECT * FROM accounts” statement shows the data from the accounts table

| | id<br>integer | name<br>character varying (100) | balance<br>numeric (15,2) |
|---|---|---|---|
| 1 | 1 | Bob | 10000.00 |
| 2 | 2 | Alice | 10000.00 |

The following example creates a stored procedure named transfer that transfers a specified amount of money from one account to another.

```
create or replace procedure transfer(
   sender int,
   receiver int,
   amount dec
)
language plpgsql
as $$
begin
    -- subtracting the amount from the sender's account
    update accounts
    set balance = balance - amount
    where id = sender;

    -- adding the amount to the receiver's account
    update accounts
    set balance = balance + amount
    where id = receiver;

    commit;
end;$$
```

## Calling a Stored Procedure

To call a stored procedure, you use the CALL statement as follows:

*call stored_procedure_name(argument_list);*

For example, this statement invokes the transfer stored procedure to transfer $1,000 from Bob's account to Alice's account.

*call transfer(1,2,1000);*

The following statement verifies the data in the accounts table after the transfer:

*SELECT * FROM accounts*

| | id integer | name character varying (100) | balance numeric (15,2) |
|---|---|---|---|
| 1 | 1 | Bob | 9000.00 |
| 2 | 2 | Alice | 11000.00 |

## Postgresql drop procedure statement examples:

The drop procedure statement removes a stored procedure. The following illustrates the syntax of the drop procedure statement:

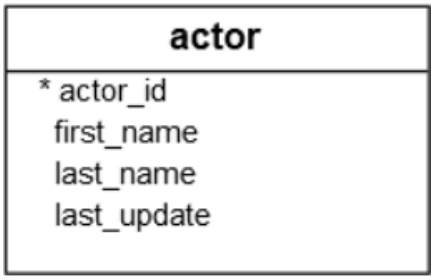
```
drop procedure [if exists] procedure_name (argument_list)
[cascade | restrict]
```

To drop multiple stored procedures, you specify a comma-list of stored procedure names after the drop procedure keyword like this:

```
drop procedure [if exists] name1, name2, ...;
```

Let's create a couple of stored procedures that manage actors so that you can learn how to drop them:

## Creating Sample Stored Procedures:




The following insert_actor() stored procedure inserts a new row into the actor table. It accepts two arguments which are the first name and last name of the actor.

```
create or replace procedure insert_actor(
        fname varchar,
        lname varchar)
language plpgsql
as $$
begin
        insert into actor(first_name, last_name)
        values('John','Doe');
end;
$$;
```

The following insert_actor stored procedure also inserts a row into the actor table. However, it accepts one argument which is the full name of the actor. The insert_actor()

uses the split_part() function to split the full name into first name and last name before inserting them into the actor table.

```
create or replace procedure insert_actor(
        full_name varchar
)
language plpgsql
as $$
declare
        fname varchar;
        lname varchar;
begin
        -- split the fullname into first & last name
        select
                split_part(full_name,' ', 1),
                split_part(full_name,' ', 2)
        into fname,
             lname;

        -- insert first & last name into the actor table
        insert into actor(first_name, last_name)
        values('John','Doe');
end;
$$;
```

The following stored procedure deletes an actor by id:

```
create or replace procedure delete_actor(
        p_actor_id int
)
language plpgsql
as $$
begin
        delete from actor
        where actor_id = p_actor_id;
end;
$$;
```

And the following stored procedure updates the first name and last name of an actor:

```
create or replace procedure update_actor(
        p_actor_id int,
        fname varchar,
        lname varchar
)
language plpgsql
as $$
begin
        update actor
        set first_name = fname,
            last_name = lname
        where actor_id = p_actor_id;
end;
$$;
```

## Postgresql drop procedure examples:

First, attempt to drop the insert_actor stored procedure:

```
drop procedure insert_actor;
```

PostgreSQL issued the following error:

```
ERROR:  procedure name "insert_actor" is not unique
HINT:  Specify the argument list to select the procedure unambiguously.
SQL state: 42725
```

Because there are two insert_actor stored procedures, you need to specify the argument list so that PostgreSQL can select the right stored procedure to drop.

Second, drop the insert_actor(varchar) stored procedure that accepts one argument:

```
drop procedure insert_actor(varchar);
```

Since the insert_actor stored procedure is unique now, you can drop it without specifying the argument list:

```
drop procedure insert_actor;
```

It is the same as:

```
drop procedure insert_actor(varchar,varchar);
```

Third, drop two stored procedures using a single drop procedure statement:

```
drop procedure
        delete_actor,
        update_actor;
```

## Mssql Stored Procedure

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers @City = 'London';
```

## Mssql Stored Procedure with multiple parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
GO;
```

Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```