

Tutorial 04: DCL and TCL commands

Data Control Language (DCL):

DCL commands are used to enforce database security in a multiple user database environment. Two types of DCL commands are GRANT and REVOKE. Only Database Administrator's or owners of the database object can provide/remove privileges on a database object.

SQL GRANT Command:

SQL GRANT is a command used to provide access or privileges on the database objects to the users. **The Syntax for the GRANT command is:**

```
GRANT privilege_name ON object_name TO {user_name | PUBLIC | role_name}  
[WITH GRANT OPTION];
```

- **privilege_name** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- **object_name** is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- **user_name** is the name of the user to whom an access right is being granted.
- **role_name** is the name of the role to whom an access right is being granted.
- **PUBLIC** is used to grant access rights to all users.
- **ROLES** are a set of privileges grouped together.
- **WITH GRANT OPTION** - allows a user to grant access rights to other users.

For Example:

```
GRANT SELECT ON employee TO user1;
```

This command grants a SELECT permission on employee table to user1. You should use the WITH GRANT option carefully because for example if you GRANT SELECT privilege on employee table to user1 using the WITH GRANT option, then user1 can GRANT SELECT privilege on employee table to another user, such as user2 etc. Later, if you

REVOKE the SELECT privilege on employee from user1, still user2 will have SELECT privilege on employee table.

SQL REVOKE Command:

The REVOKE command removes user access rights or privileges to the database objects. The Syntax for the REVOKE command is:

```
REVOKE privilege_name ON object_name FROM {user_name |PUBLIC  
|role_name}
```

For Example:

```
REVOKE SELECT ON employee FROM user1;
```

This command will REVOKE a SELECT privilege on employee table from user1. When you REVOKE SELECT privilege on a table from a user, the user will not be able to SELECT data from that table anymore. However, if the user has received SELECT privileges on that table from more than one users, he/she can SELECT from that table until everyone who granted the permission revokes it. You cannot REVOKE privileges if they were not initially granted by you.

Create a new User:

Connect to the psql using superuser and create new database use “dcl” for database name. use the following command to create dcl database. Create a new user use “user1” for username and “1234” for password. Use the following command to create user1.

```
CREATE USER user1 WITH PASSWORD '1234';
```

Connect to the dcl database using user1. Type the following command to connect to the dcl database using user1. And the Enter the password of user.

```
\connect world user1
```

Try to insert the following record in the emp table when you are connected to the database as user1.

```
INSERT INTO employees VALUES (1, 'Mohammed', 4553);
```

You will get permission denied error message, this error because user1 does not have enough permission to insert data in the emp table. Now connect to the dcl database as superuser. Type the following command to connect to dcl database using postgres user. And then enter the password of postgres user.

```
\connect world postgres
```

Grant user1 select and insert permission on employees table. Type the following command to grant user1 select and insert permission on employees table.

```
GRANT SELECT, INSERT ON employees TO user1;
```

Connect to the dcl database using user1. Type the following command to connect to the dcl database using user1. And the Enter the password of user.

```
\connect world user1
```

Try to insert the following record in the employees table when you are connected to the database as user1.

```
INSERT INTO employees VALUES (11, 'Mohammed', 4553, 56);
```

The record inserted successfully in employees table. Use the select command to show the inserted records.

```
SELECT * FROM employees;
```

Let us to delete the inserted record from employees table. Type the following command to delete record from the employees table.

```
DELETE FROM employees WHERE id = 1;
```

You will get permission denied error message, because user1 does not have delete permission on employees table. We will use the revoke command to revoke insert permission on employees table from user1. Now connect to the dcl database as superuser. Type the following command to connect to dcl database using postgres user. And then enter the password of postgres user.

```
\connect world postgres
```

Type the following command to revoke the insert permission on employees table from user1.

```
REVOKE INSERT ON employees FROM user1 ;
```

Connect to the dcl database using user1. Type the following command to connect to the dcl database using user1. And then Enter the password of user.

```
\connect world user1
```

Try to insert the following record in the employees table when you are connected to the database as user1.

```
INSERT INTO employees VALUES (2, 'Omer', 5543, 50);
```

You will get permission denied error message. Because insert permission on employees table has been revoked from user1.

Transactions Control Language (TCL):

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record, updating a record, or deleting a record from the table, then you are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

Practically, you will club many PostgreSQL queries into a group and you will execute all of them together as a part of a transaction. TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements:

- **BEGIN Transaction** – To start a transaction.
- **COMMIT Transaction** – To save the changes, alternatively you can use END command.
- **ROLLBACK Transaction** – To rollback the changes.

- **SAVEPOINT Transaction** – establishes a new savepoint within the current transaction.

Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

Case Study:

Connect to SQL Shell (psql) using postgres superuser and create a new table for cities use city for table name the city table consists of three columns id, name and pop (population), write the following command to create the table.

```
CREATE TABLE city (ID SERIAL, NAME VARCHAR(100), POP INTEGER);
```

For transaction control it is necessary to declare the beginning of transaction using begin command. Write the following command to begin transaction.

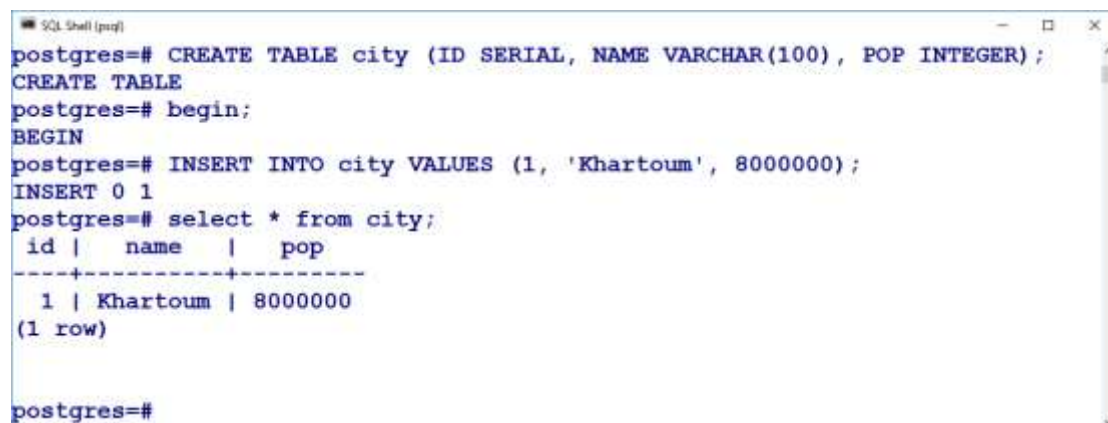
```
BEGIN;
```

Use the insert command to insert data into city table, use the following SQL command to insert data.

```
INSERT INTO city VALUES (1, 'Khartoum', 8000000);
```

Use SELECT statement to show the inserted data. The select command will return the insert data.

```
SELECT * FROM city;
```




```
SQL Shell (psql)
postgres=# CREATE TABLE city (ID SERIAL, NAME VARCHAR(100), POP INTEGER);
CREATE TABLE
postgres=# begin;
BEGIN
postgres=# INSERT INTO city VALUES (1, 'Khartoum', 8000000);
INSERT 0 1
postgres=# select * from city;
 id |  name   |  pop
----+-----+-----
  1 | Khartoum | 8000000
(1 row)

postgres=#
```

Open another SQL Shell (psql) window connect to PostgreSQL Server using postgres superuser and use SELECT command to show the inserted data. In the second SQL Shell (psql) the SELECT command does not return inserted data, this because the data is not committed yet.

```
SELECT * FROM city;
```

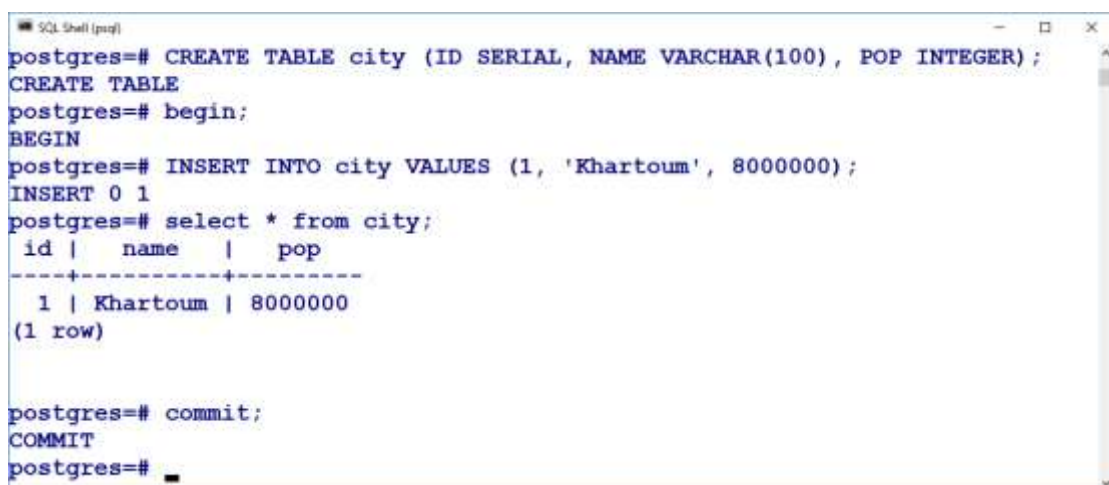


```
SQL Shell (psql)
postgres=# select * from city;
 id | name | pop
----+-----+----
(0 rows)

postgres=#
```

Return to the First SQL Shell (psql) window where the data has been inserted, and use commit command to commit the inserted data.

```
COMMIT;
```



```
SQL Shell (psql)
postgres=# CREATE TABLE city (ID SERIAL, NAME VARCHAR(100), POP INTEGER);
CREATE TABLE
postgres=# begin;
BEGIN
postgres=# INSERT INTO city VALUES (1, 'Khartoum', 8000000);
INSERT 0 1
postgres=# select * from city;
 id |  name  |  pop
----+-----+----
  1 | Khartoum | 8000000
(1 row)

postgres=# commit;
COMMIT
postgres=#
```

Go to the second SQL Shell (psql) and use SELECT statement to show inserted data.

```
SELECT * FROM city;
```

```
SQL Shell (psql)
postgres=# select * from city;
 id | name | pop
-----+-----+-----
(0 rows)

postgres=# select * from city;
 id |  name  |  pop
-----+-----+-----
  1 | Khartoum | 8000000
(1 row)

postgres=#
```

Now start a new transaction using begin; command and use INSERT command to insert additional data into city table.

```
BEGIN;
INSERT INTO city VALUES (2, 'Nyala', 2000000);
```

Then use select command to show inserted data from city table.

```
SELECT * FROM city;
```

Use rollback; command to undo from the last insert transaction.

```
ROLLBACK;
```

The use select command to see city data.

```
SELECT * FROM city;
```

```
SQL Shell (psql)
postgres=# BEGIN;
BEGIN
postgres=# INSERT INTO city VALUES (2, 'Nyala', 2000000);
INSERT 0 1
postgres=# SELECT * FROM city;
 id |  name  |  pop
-----+-----+-----
  1 | Khartoum | 8000000
  2 | Nyala    | 2000000
(2 rows)

postgres=# ROLLBACK;
ROLLBACK
postgres=# SELECT * FROM city;
 id |  name  |  pop
-----+-----+-----
  1 | Khartoum | 8000000
(1 row)

postgres=#
```

Now start a new transaction using begin; command and use INSERT command to insert additional data into city table.

```
BEGIN;  
INSERT INTO city VALUES (3, 'Kusti', 1500000);
```

Use savepoint command to save the current transactions situation and use **point1** for the savepoint name.

```
SAVEPOINT point1;
```

Use INSERT command to insert additional data into city table.

```
INSERT INTO city VALUES (4, 'Gadarig', 1350000);
```

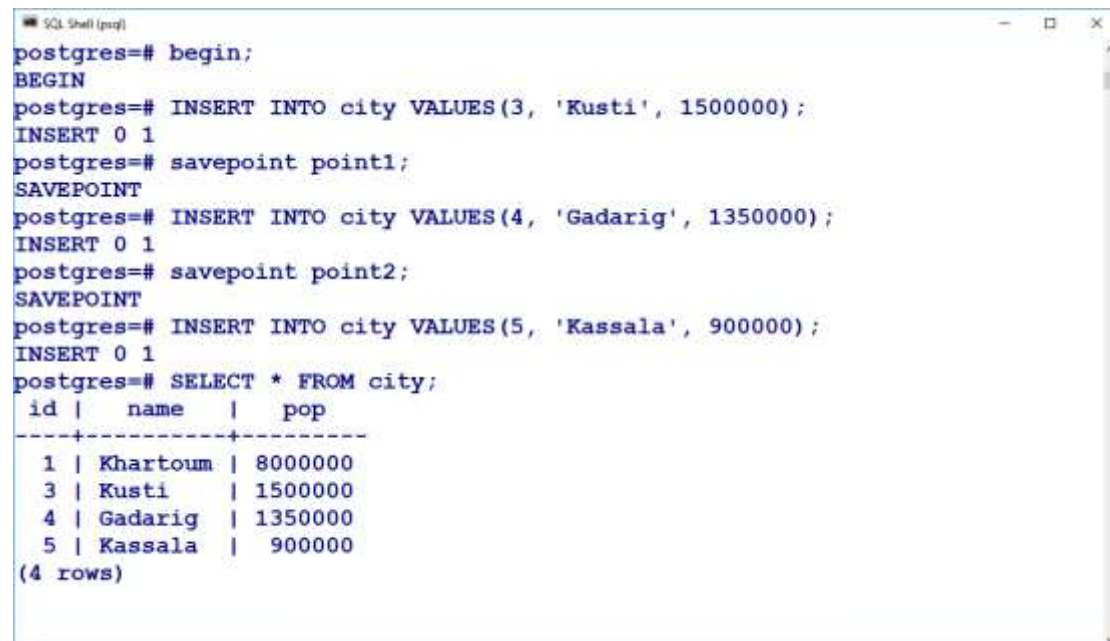
Use savepoint command to save the current transactions situation and use **point2** for the savepoint name.

```
SAVEPOINT point2;
```

Use INSERT command to insert additional data into city table.

```
INSERT INTO city VALUES (5, 'Kassala', 900000);
```

Use SELECT command to show inserted data.



```
SQL Shell (psql)  
postgres=# begin;  
BEGIN  
postgres=# INSERT INTO city VALUES(3, 'Kusti', 1500000);  
INSERT 0 1  
postgres=# savepoint point1;  
SAVEPOINT  
postgres=# INSERT INTO city VALUES(4, 'Gadarig', 1350000);  
INSERT 0 1  
postgres=# savepoint point2;  
SAVEPOINT  
postgres=# INSERT INTO city VALUES(5, 'Kassala', 900000);  
INSERT 0 1  
postgres=# SELECT * FROM city;  
 id | name   | pop  
----+-----+----  
  1 | Khartoum | 8000000  
  3 | Kusti    | 1500000  
  4 | Gadarig  | 1350000  
  5 | Kassala  | 900000  
(4 rows)
```

Use ROLLBACK command to undo to point2.

```
ROLLBACK TO point2;
```

Then use SELECT command to show the city data.


```
SELECT * FROM city;
```

```
SQL Shell (psql)
postgres=# SELECT * FROM city;
 id |  name  |  pop
-----+-----+-----
  1 | Khartoum | 8000000
  3 | Kusti   | 1500000
  4 | Gadarig | 1350000
  5 | Kassala |  900000
(4 rows)

postgres=# ROLLBACK TO point2;
ROLLBACK
postgres=# SELECT * FROM city;
 id |  name  |  pop
-----+-----+-----
  1 | Khartoum | 8000000
  3 | Kusti   | 1500000
  4 | Gadarig | 1350000
(3 rows)

postgres=#
```