ADVANCED DATABASES

GI: 2nd year

CONTENTS:

All SQL statements will be given according to postresql syntax

CHAPTER III

Postgresql Administration

CONTENTS:

MANAGING DATABASES

In this section, we will learn how to manage databases in PostgreSQL including creating databases, modifying existing database features, and deleting databases.

MANAGING DATABASES: CREATE DATABASE

The CREATE DATABASE statement allows you to create a new PostgreSQL database.

```
CREATE DATABASE database_name
WITH
   [OWNER =  role_name]
   [TEMPLATE = template]
   [ENCODING = encoding]
   [LC_COLLATE = collate]
   [LC_CTYPE = ctype]
   [TABLESPACE = tablespace_name]
   [ALLOW_CONNECTIONS = true | false]
   [CONNECTION LIMIT = max_concurrent_connection]
   [IS_TEMPLATE = true | false ]
```

MANAGING DATABASES: CREATE DATABASE

- To execute the CREATE DATABASE statement you need to have a superuser role or a special CREATEDB privilege.
- To create a new database:
  - First, specify the name of the new database after the CREATE DATABASE keywords:
    - The database name must be unique in the PostgreSQL database server.
    - If you try to create a database whose name already exists, PostgreSQL will issue an error
  - Then, specify one or more parameters for the new database

MANAGING DATABASES: CREATE DATABASE

Parameters:

OWNER

Assign a role that will be the owner of the database. If you omit the OWNER option, the owner of the database is the role that you use to execute the CREATE DATABASE statement.

TEMPLATE

Specify the template database from which the new database is created. By default, PostgreSQL uses the template1 database as the template database if you don't explicitly specify the template database.

ENCODING

Determine the character set encoding in the new database.

MANAGING DATABASES: CREATE DATABASE

Parameters:

LC_COLLATE

Specify the collation order (LC_COLLATE) that the new database will use. This parameter affects the sort order of strings in the queries that contain the ORDER BY clause. It defaults to the LC_COLLATE of the template database.

LC_CTYPE

Specify the character classification that the new database will use. It affects the classification of characters e.g., lower, upper, and digit. It defaults to the LC_CTYPE of the template database.

TABLESPACE

Specify the tablespace name for the new database. The default is the tablespace of the template database.

MANAGING DATABASES: CREATE DATABASE

Parameters:

CONNECTION LIMIT

Specify the maximum concurrent connections to the new database. The default is -1 i.e., unlimited. This parameter is useful in shared hosting environments where you can configure the maximum concurrent connections for a particular database.

ALLOW_CONNECTIONS

The allow_connections parameter is a boolean value. If it is false, you cannot connect to the database.

IS_TEMPLATE

If the IS_TEMPLATE is true, any user with the CREATEDB privilege can clone it. If false, only superusers or the database owner can clone it

MANAGING DATABASES: CREATE DATABASE

Examples:

1)
```
CREATE DATABASE im
WITH
  ENCODING = 'UTF8'
  OWNER = im
  CONNECTION LIMIT = 100;
```

2) Create a new database using pgAdmin

MANAGING DATABASES: ALTER DATABASE

The ALTER DATABASE statement allows you to carry the following action on the database:

- Change the attributes of the database
- Rename the database
- Change the owner of the database
- Change the default tablespace of a database
- Change the session default for a run-time configuration variable for a database

MANAGING DATABASES: ALTER DATABASE

1) Changing attributes of a database

To change the attributes of a database, you use the following form of the ALTER TABLE statement:

```
ALTER DATABASE name WITH option;
```

The option can be:

- IS_TEMPLATE
- CONNECTION LIMIT
- ALLOW_CONNECTIONS

Note that only superusers or database owners can change these setttings.

MANAGING DATABASES: ALTER DATABASE

2) Rename the database

The following ALTER DATABASE RENAME TO statement renames a database:

```
ALTER DATABASE database_name RENAME TO new_name;
```

- It is not possible to rename the current database.
- You need to connect to another database and rename it from that database.
- Only superusers and database owners with CREATEDB privilege can rename the database.

MANAGING DATABASES: ALTER DATABASE

3) Change the owner of the database

The following ALTER DATABASE statement changes the owner of a database to the new one:

```
ALTER DATABASE database_name OWNER TO new_owner | current_user | session_user;
```

The following users can change the onwer of the database:

- The database owner with CREATEDB privilege and is a direct or indirect member of the new owning role.
- The superusers

MANAGING DATABASES: ALTER DATABASE

4) Change the default tablespace of a database

The following statement changes the default tablespace of the database:

```
ALTER DATABASE database_name SET TABLESPACE new_tablespace;
```

- The statement physically moves tables and indexes from the legacy tablespace to the new one.
- To set the new tablespace, the tablespace needs to be empty and there is connection to the database.
- Superusers and database owner can change the default tablespace of the database.

MANAGING DATABASES: ALTER DATABASE

5) Change session defaults for run-time configuration variables

Whenever you connect to a database, PostgreSQL loads the configuration variables from the postgresql.conf file and uses these variables by default.

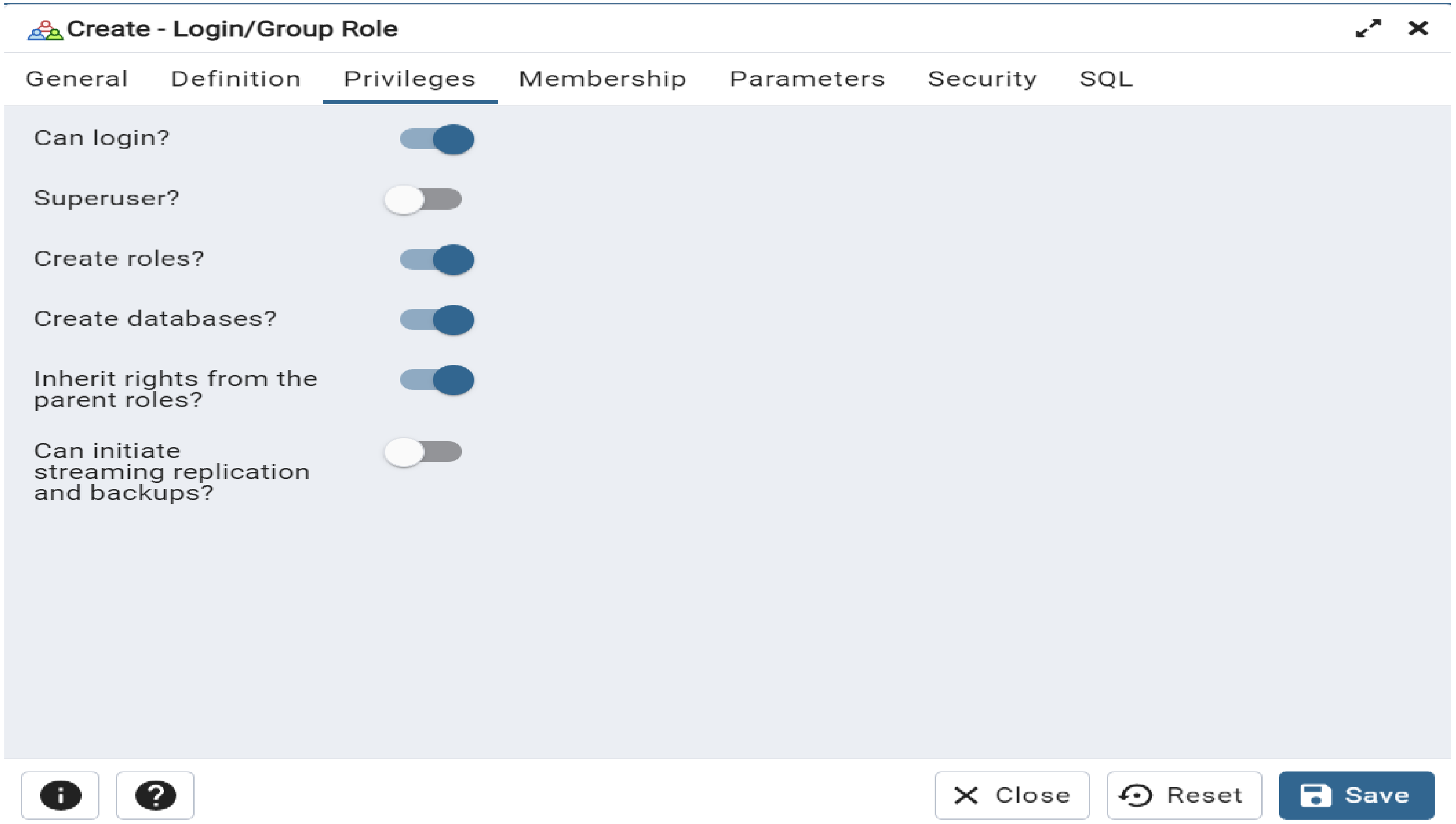
To override these settings for a particular database, you use ALTER DATABASE SET statement as follows:

```
ALTER DATABASE database_name SET configuration_parameter = value;
```

In the subsequent sessions, PostgreSQL will override the settings in the postgresql.conf file.

Only superusers or database owners can change the session default for a run-time configuration for the database.

MANAGING DATABASES: ALTER DATABASE



Create - Login/Group Role

Can login?	✓
Superuser?	✗
Create roles?	✓
Create databases?	✓
Inherit rights from the parent roles?	✓
Can initiate streaming replication and backups?	✗

MANAGING DATABASES: ALTER DATABASE

```
-- create a new database named testdb2 for the demonstration
CREATE DATABASE testdb2;
-- rename the testdb2 to testimdb using the following statement:
ALTER DATABASE testdb2
RENAME TO testimdb;
-- execute the following statement to change the owner of the testimdb database from postgres to
im,
-- with the assumption that the im role already exists.
ALTER DATABASE testimdb
OWNER TO im;
-- If the hr role does not exist, you can create it by using the CREATE ROLE statement:
CREATE ROLE im
LOGIN
CREATEDB
PASSWORD 'securePa$$1';
-- change the default tablespace of the testimdb from pg_default to im_default,
-- with the assumption that the im_default tablespace already exists.
ALTER DATABASE testimdb
SET TABLESPACE im_default;
-- If the im_default tablespace does not exist, you can create it by using the following statement:
CREATE TABLESPACE im_default
OWNER im
LOCATION 'C:\Users\Public\Documents\im';
-- set escape_string_warning configuration variable to off by using the following statement:
ALTER DATABASE testimdb
SET escape_string_warning = off;
```

MANAGING DATABASES: DROP DATABASE

The following illustrates the syntax of the DROP DATABASE statement:

```
DROP DATABASE [IF EXISTS] database_name;
```

To delete a database:

- Specify the name of the database that you want to delete after the DROP DATABASE clause.
- Use IF EXISTS to prevent an error from removing a non-existent database. PostgreSQL will issue a notice instead.

MANAGING DATABASES: DROP DATABASE

- The DROP DATABASE statement deletes catalog entries and data directory permanently. This action cannot be undone so you have to use it with caution.
- Only superusers and the database owner can execute the DROP DATABASE statement.
- You cannot execute the DROP DATABASE statement if the database still has active connections. In this case, you need to disconnect from the database and connect to another database e.g., postgres to execute the DROP DATABASE statement.

MANAGING DATABASES: SCHEMA

- In PostgreSQL, a schema is a namespace that contains named database objects such as tables, views, indexes, data types, functions, stored procedures and operators.
- To access an object in a schema, you need to qualify the object by using the following syntax:

  schema_name.object_name

- A database can contain one or multiple schemas and each schema belongs to only one database.
- Two schemas can have different objects that share the same name.

MANAGING DATABASES: SCHEMA

For example, you may have sales schema that has staff table and the public schema which also has the staff table.

When you refer to the staff table you must qualify it as follows:

public.staff

Or

sales.staff

MANAGING DATABASES: SCHEMA

- Schemas allow you to organize database objects e.g., tables into logical groups to make them more manageable.
- Schemas enable multiple users to use one database without interfering with each other.

MANAGING DATABASES: SCHEMA

The public schema:

PostgreSQL automatically creates a schema called **public** for every new database.

Whatever object you create without specifying the schema name, PostgreSQL will place it into this **public** schema.

Therefore, the following statements are equivalent:

```
CREATE TABLE table_name(
  ...
);
```

```
CREATE TABLE public.table_name(
  ...
);
```

MANAGING DATABASES: SCHEMA

The schema search path

- In practice, you will refer to a table without its schema name e.g., staff table instead of a fully qualified name such as sales.staff table.
- When you reference a table using its name only, PostgreSQL searches for the table by using the **schema search path**, which is a list of schemas to look in.
- PostgreSQL will access the first matching table in the schema search path. If there is no match, it will return an error, even the name exists in another schema in the database.
- The first schema in the search path is called the current schema. Note that when you create a new object without explicitly specifying a schema name, PostgreSQL will also use the current schema for the new object.

MANAGING DATABASES: SCHEMA

The schema search path

The current_schema() function returns the current schema:

```
SELECT current_schema();
```

Here is the output:

```
current_schema
----------------
public
```

(1 row)

This is why PostgreSQL uses public for every new object that you create

MANAGING DATABASES: SCHEMA

To view the current search path, you use the SHOW command in psql tool:

```
SHOW search_path;
```

The output is as follows:

```
search_path
-----------------
"$user", public
(1 row)
```

- The "$user" specifies that the first schema that PostgreSQL will use to search for the object, which has the same name as the current user.
- For example, if you use the postgres user to login and access the staff table. PostgreSQL will search for the staff table in the postgres schema. If it cannot find any object like that, it continues to look for the object in the public schema.
- The second element refers to the public schema as we have seen before.

MANAGING DATABASES: SCHEMA

To create a new schema, you use the CREATE SCHEMA statement:

```
CREATE SCHEMA sales;
```

To add the new schema to the search path, you use the following command:

```
SET search_path TO sales, public;
```

If you create a new table named staff without specifying the schema name, PostgreSQL will put this staff table into the sales

MANAGING DATABASES: SCHEMA

To create a new schema, you use the CREATE SCHEMA statement:

```
CREATE SCHEMA sales;
```

To add the new schema to the search path, you use the following command:

```
SET search_path TO sales, public;
```

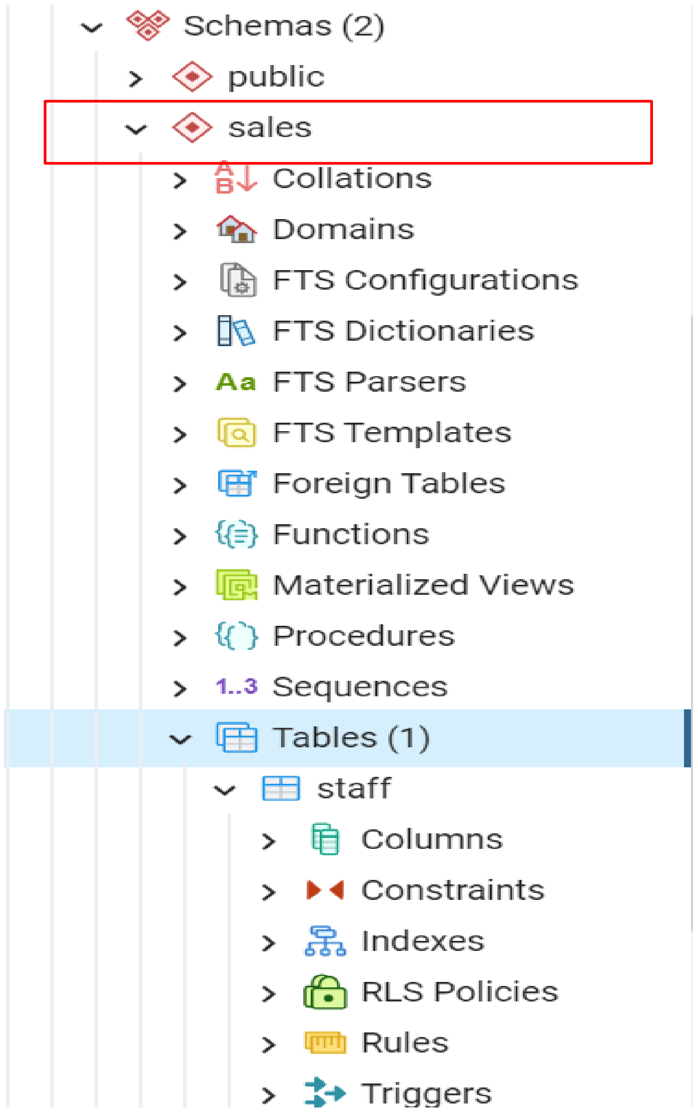
If you create a new table named staff without specifying the schema name, PostgreSQL will put this staff table into the sales

MANAGING DATABASES: SCHEMA

```
CREATE SCHEMA sales;
SET search_path TO sales, public;


CREATE TABLE staff(
    staff_id SERIAL PRIMARY KEY,
    first_name VARCHAR(45) NOT NULL,
    last_name VARCHAR(45) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);
```

The following picture shows the new schema **sales** and the **staff** table that belongs to the **sales** schema:



MANAGING DATABASES: SCHEMA

PostgreSQL schemas and privileges

- Users can only access objects in the schemas that they own. It means they cannot access any objects in the schemas that do not belong to them.
- To allow users to access the objects in the schema that they do not own, you must grant the USAGE privilege of the schema to the users:

  ```
  GRANT USAGE ON SCHEMA schema_name TO role_name;
  ```
- To allow users to create objects in the schema that they do not own, you need to grant them the CREATE privilege of the schema to the users:

  ```
  GRANT CREATE ON SCHEMA schema_name TO user_name;
  ```
- By default, every user has the CREATE and USAGE privilege on the public schema.

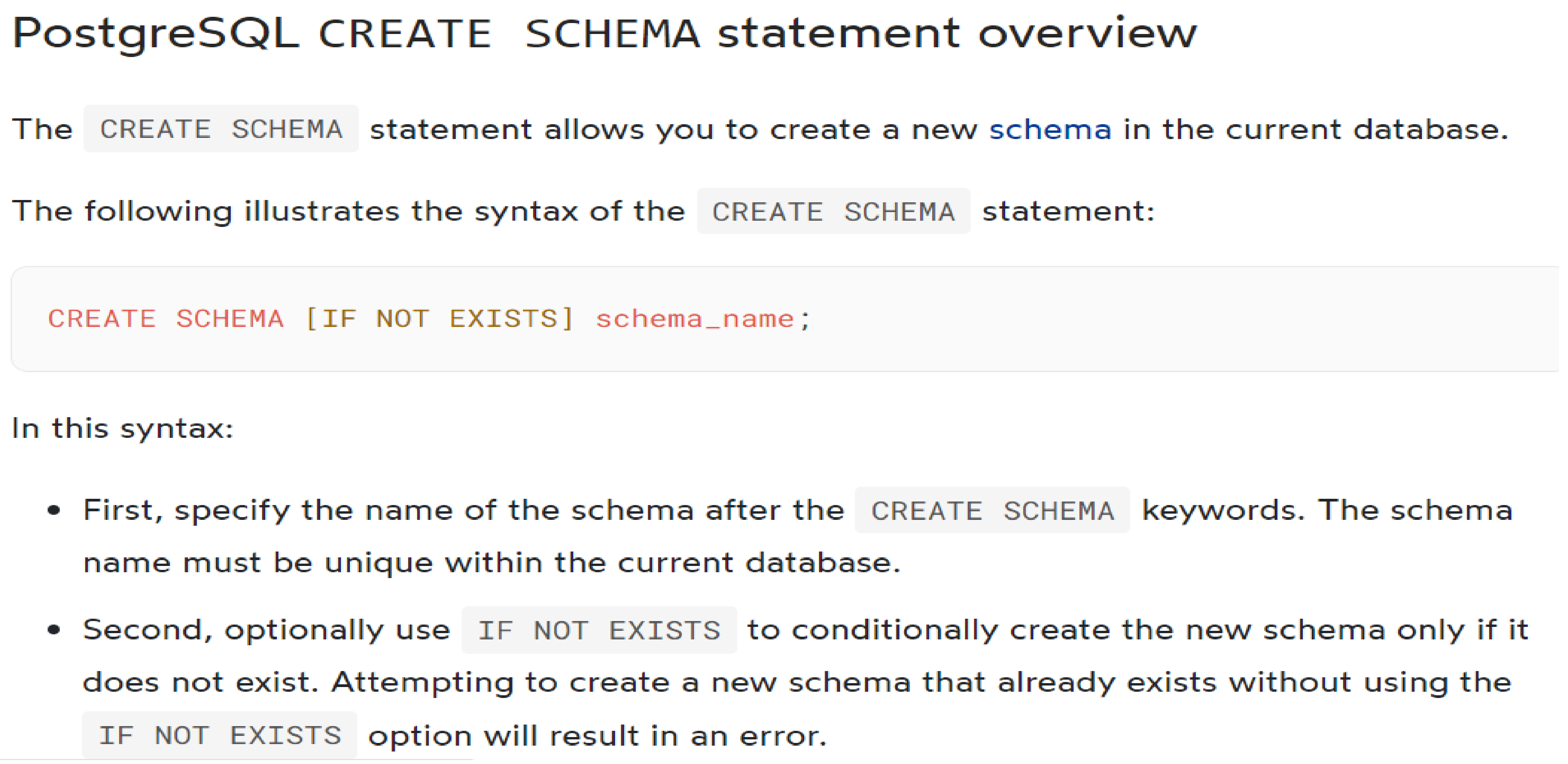
MANAGING DATABASES: SCHEMA

PostgreSQL schema operations

- To create a new schema, use the CREATE SCHEMA statement.
- To rename a schema or change its owner, use the ALTER SCHEMA statement.
- To drop a schema, use the DROP SCHEMA statement.

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations



PostgreSQL CREATE SCHEMA statement overview

The `CREATE SCHEMA` statement allows you to create a new schema in the current database.

The following illustrates the syntax of the `CREATE SCHEMA` statement:

```
CREATE SCHEMA [IF NOT EXISTS] schema_name;
```

In this syntax:

- First, specify the name of the schema after the `CREATE SCHEMA` keywords. The schema name must be unique within the current database.
- Second, optionally use `IF NOT EXISTS` to conditionally create the new schema only if it does not exist. Attempting to create a new schema that already exists without using the `IF NOT EXISTS` option will result in an error.

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations

You can also create a schema for a user:

```
CREATE SCHEMA [IF NOT EXISTS]
AUTHORIZATION username;
```

In this case, the schema will have the same name as the `username`;

PostgreSQL also allows you to create a schema and a list of objects such as tables and views using a single statement as follows:

```
CREATE SCHEMA schema_name
    CREATE TABLE table_name1 (...)
    CREATE TABLE table_name2 (...)
    CREATE VIEW view_name1
        SELECT select_list FROM table_name1;
```

Notice that each subcommand does not end with a semicolon (;).

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations

PostgreSQL CREATE SCHEMA examples

Let's take some examples of using the `CREATE SCHEMA` statement to get a better understanding.

1) Using CREATE SCHEMA to create a new schema example

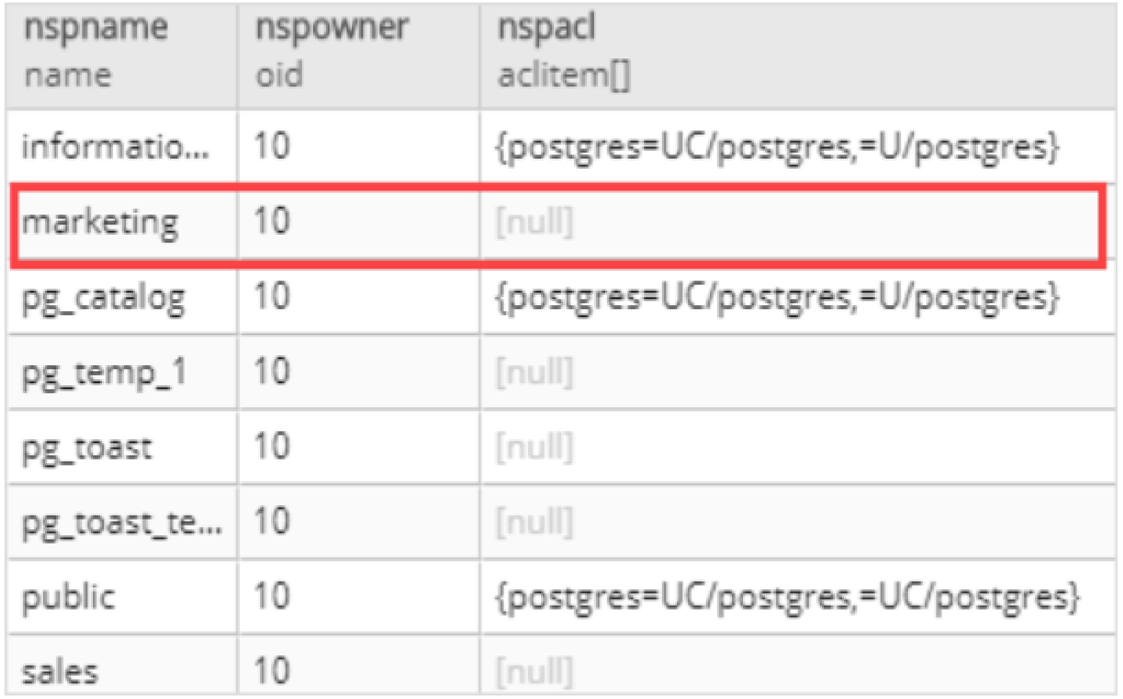
The following statement uses the `CREATE SCHEMA` statement to create a new schema named `marketing`:

```
CREATE SCHEMA marketing;
```

The following statement returns all schemas from the current database:

```
SELECT *
FROM pg_catalog.pg_namespace
ORDER BY nspname;
```

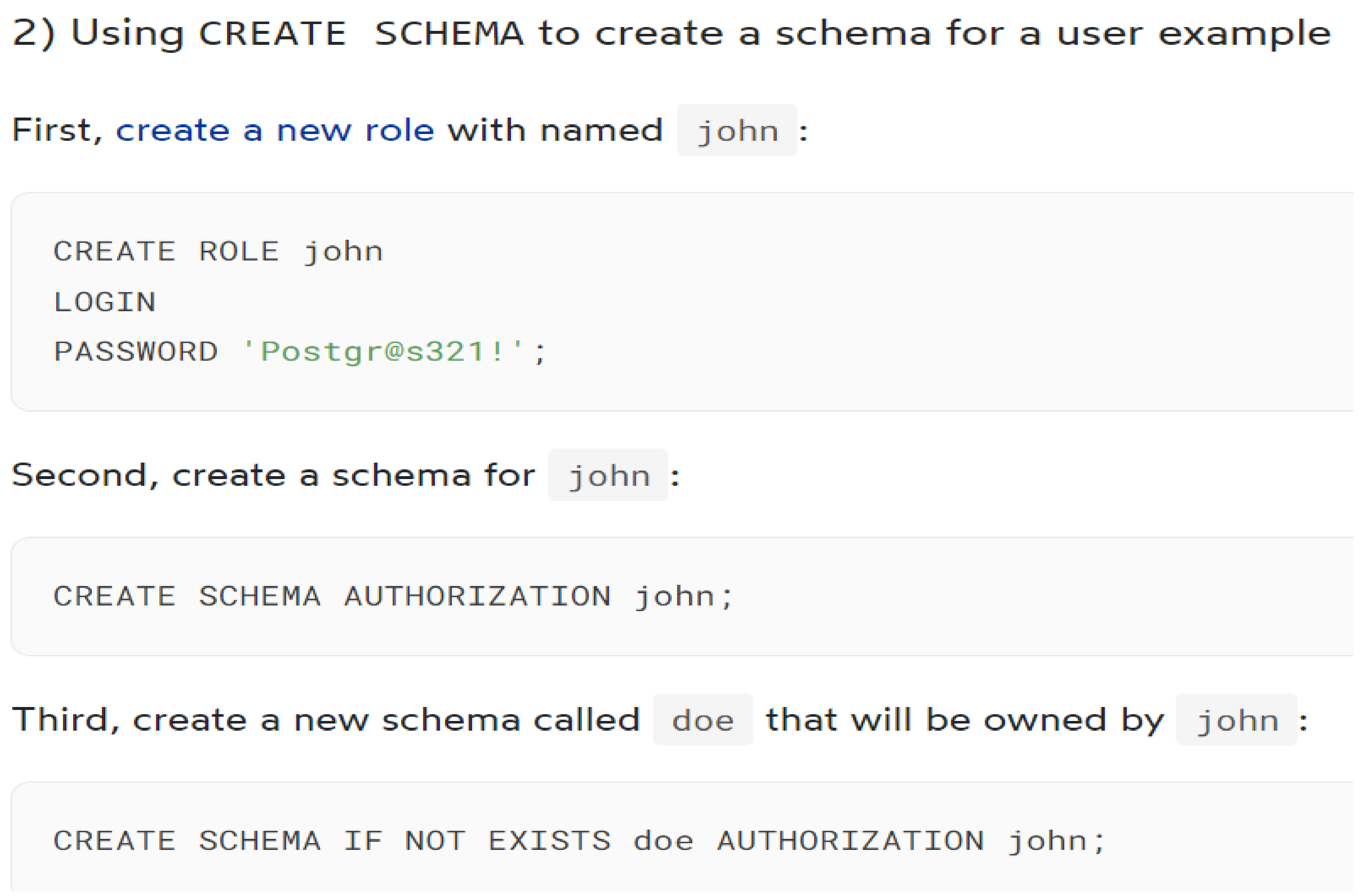
This picture shows the output:



nspname name	nspowner oid	nspacl aclitem[]
informatio...	10	{postgres=UC/postgres,=U/postgres}
marketing	10	[null]
pg_catalog	10	{postgres=UC/postgres,=U/postgres}
pg_temp_1	10	[null]
pg_toast	10	[null]
pg_toast_te...	10	[null]
public	10	{postgres=UC/postgres,=UC/postgres}
sales	10	[null]

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations



2) Using CREATE SCHEMA to create a schema for a user example

First, create a new role with named `john` :

```
CREATE ROLE john
LOGIN
PASSWORD 'Postgr@s321!';
```

Second, create a schema for `john` :

```
CREATE SCHEMA AUTHORIZATION john;
```

Third, create a new schema called `doe` that will be owned by `john` :

```
CREATE SCHEMA IF NOT EXISTS doe AUTHORIZATION john;
```

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations

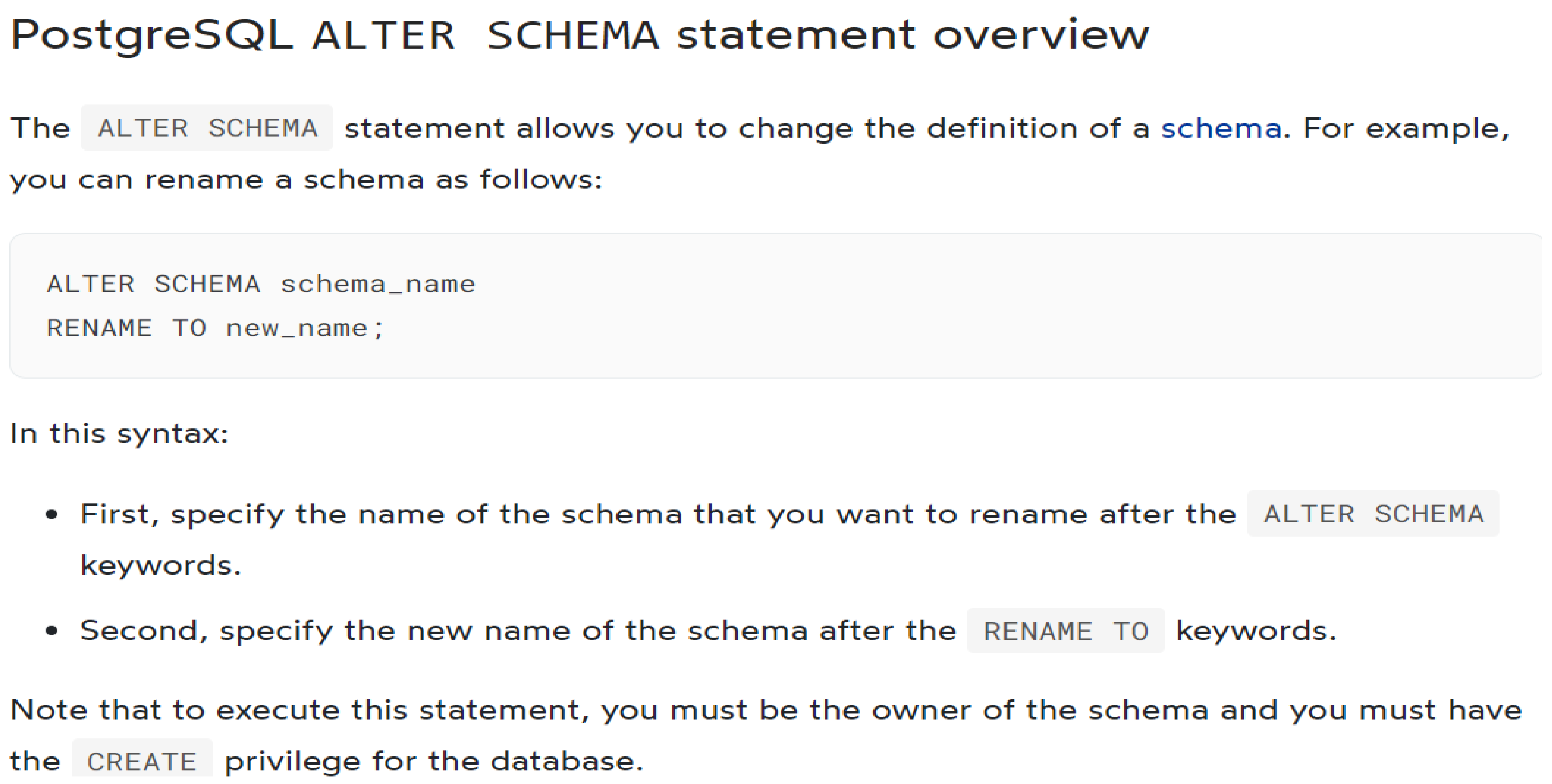
3) Using CREATE SCHEMA to create a schema and its objects example

The following example uses the `CREATE SCHEMA` statement to create a new schema named `scm`. It also creates a table named `deliveries` and a view named `delivery_due_list` that belongs to the `scm` schema:

```
CREATE SCHEMA scm
    CREATE TABLE deliveries(
        id SERIAL NOT NULL,
        customer_id INT NOT NULL,
        ship_date DATE NOT NULL
    )
    CREATE VIEW delivery_due_list AS
        SELECT ID, ship_date
        FROM deliveries
        WHERE ship_date <= CURRENT_DATE;
```

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations



PostgreSQL ALTER SCHEMA statement overview

The `ALTER SCHEMA` statement allows you to change the definition of a schema. For example, you can rename a schema as follows:

```
ALTER SCHEMA schema_name
RENAME TO new_name;
```

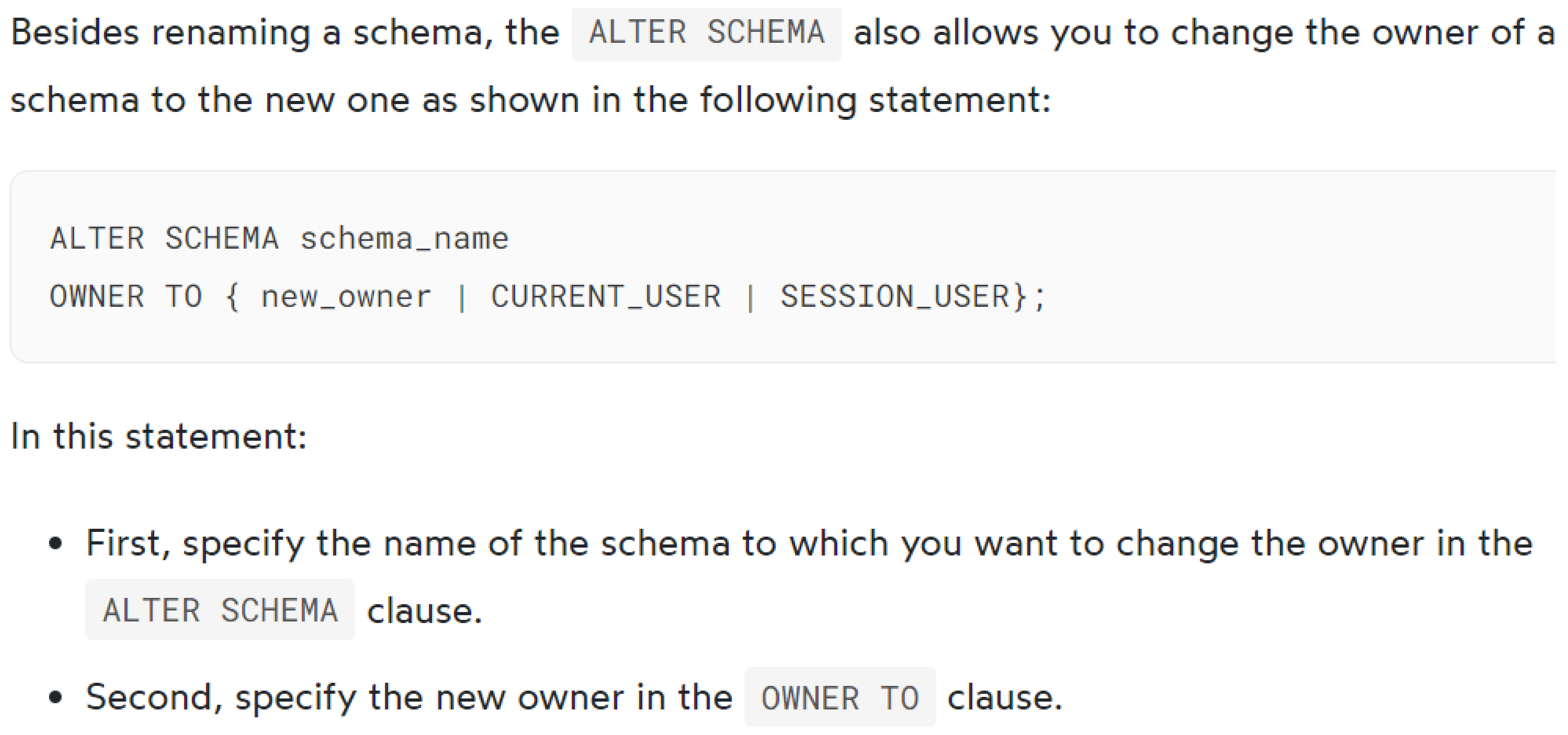
In this syntax:

- First, specify the name of the schema that you want to rename after the `ALTER SCHEMA` keywords.
- Second, specify the new name of the schema after the `RENAME TO` keywords.

Note that to execute this statement, you must be the owner of the schema and you must have the `CREATE` privilege for the database.

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations



Besides renaming a schema, the `ALTER SCHEMA` also allows you to change the owner of a schema to the new one as shown in the following statement:

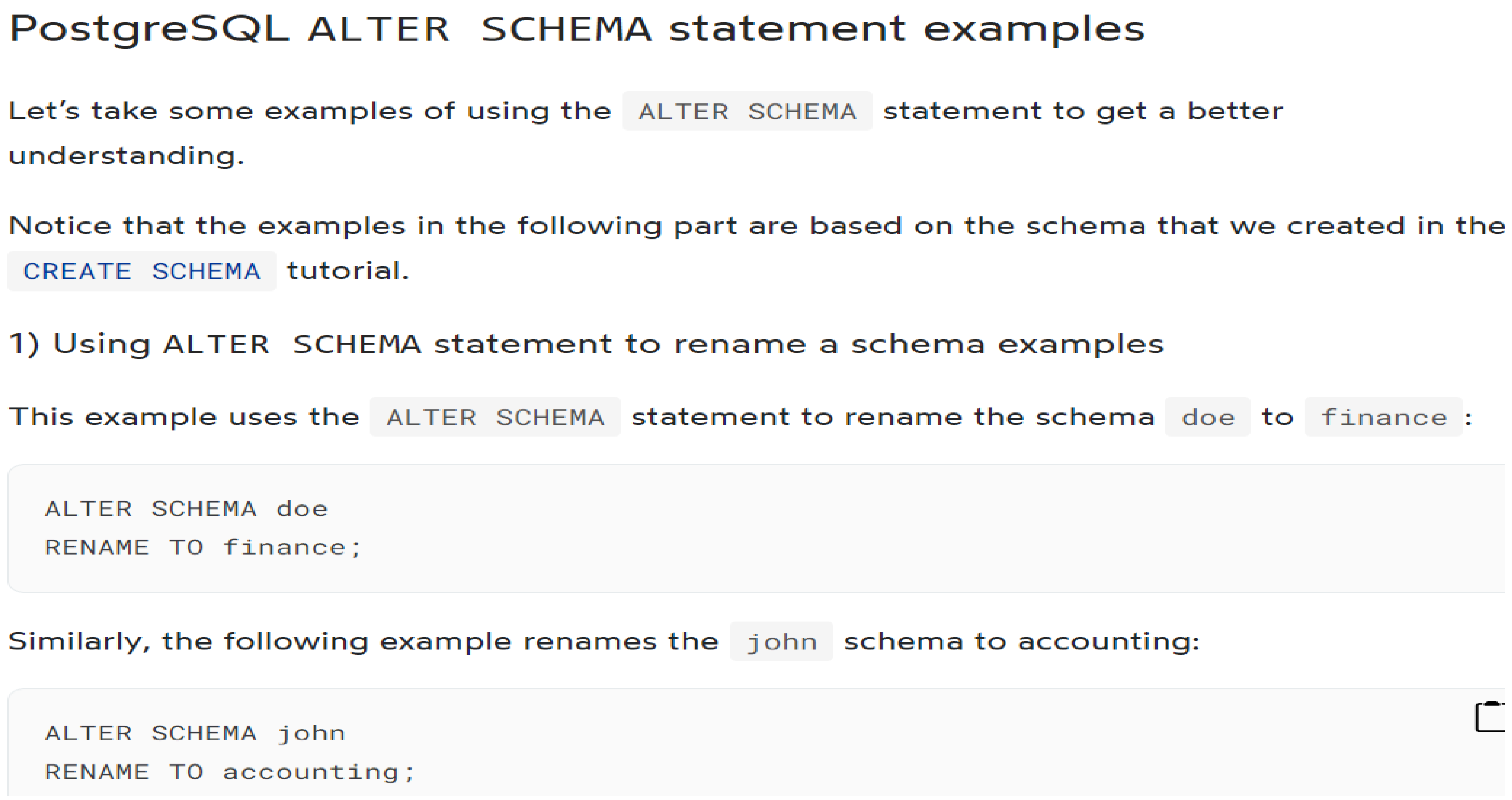
```
ALTER SCHEMA schema_name
OWNER TO { new_owner | CURRENT_USER | SESSION_USER};
```

In this statement:

- First, specify the name of the schema to which you want to change the owner in the `ALTER SCHEMA` clause.
- Second, specify the new owner in the `OWNER TO` clause.

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations



PostgreSQL ALTER SCHEMA statement examples

Let's take some examples of using the `ALTER SCHEMA` statement to get a better understanding.

Notice that the examples in the following part are based on the schema that we created in the `CREATE SCHEMA` tutorial.

1) Using ALTER SCHEMA statement to rename a schema examples

This example uses the `ALTER SCHEMA` statement to rename the schema `doe` to `finance`:

```
ALTER SCHEMA doe
RENAME TO finance;
```

Similarly, the following example renames the `john` schema to accounting:

```
ALTER SCHEMA john
RENAME TO accounting;
```

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations

2) Using ALTER SCHEMA statement to change the owner of a schema example

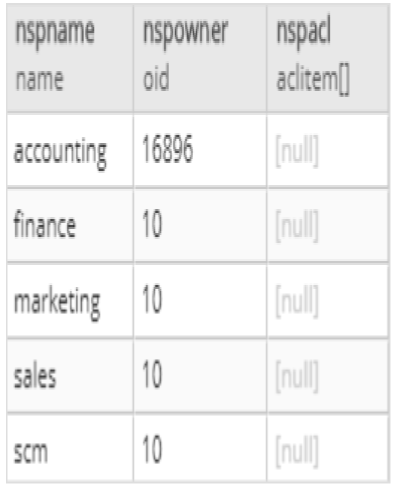
The following example uses the `ALTER SCHEMA` statement to change the owner of the schema accounting to from `john` to `postgres` :

```
ALTER SCHEMA finance
OWNER TO postgres;
```

Here is the statement to query the user-created schema:

```
SELECT *
FROM
    pg_catalog.pg_namespace
WHERE
    nspacl is NULL AND
    nspname NOT LIKE 'pg_%'
ORDER BY
    nspname;
```

The output is:

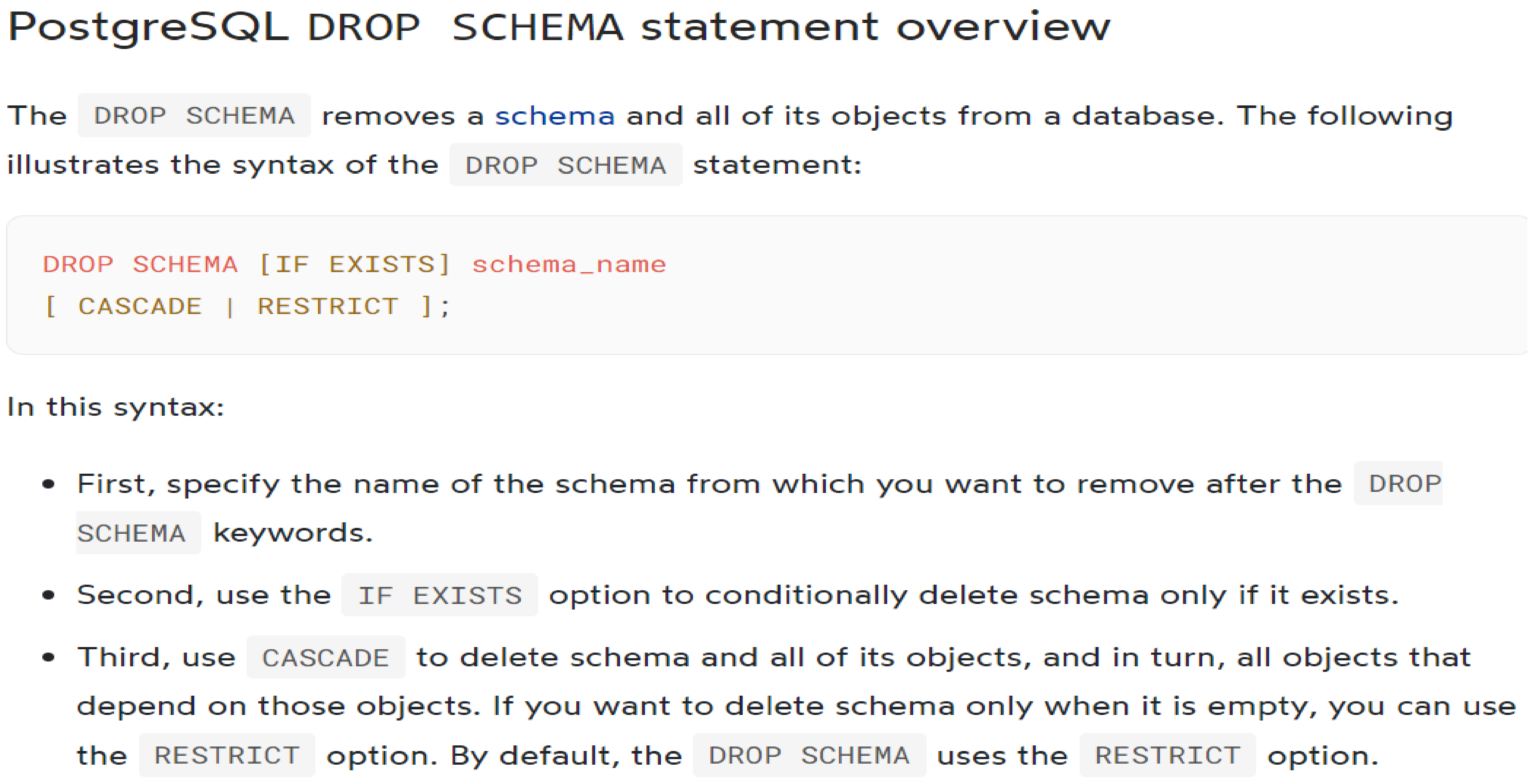


nspname	nspowner	nspacl
name	oid	aclitem[]
accounting	16896	[null]
finance	10	[null]
marketing	10	[null]
sales	10	[null]
scm	10	[null]

As you can see clearly from the output, the `finance` schema now is owned by the owner with id 10, which is `postgres` .

MANAGING DATABASES: SCHEMA

PostgreSQL schema operations



PostgreSQL DROP SCHEMA statement overview

The `DROP SCHEMA` removes a schema and all of its objects from a database. The following illustrates the syntax of the `DROP SCHEMA` statement:

```
DROP SCHEMA [IF EXISTS] schema_name
[ CASCADE | RESTRICT ];
```

In this syntax:

- First, specify the name of the schema from which you want to remove after the `DROP SCHEMA` keywords.
- Second, use the `IF EXISTS` option to conditionally delete schema only if it exists.
- Third, use `CASCADE` to delete schema and all of its objects, and in turn, all objects that depend on those objects. If you want to delete schema only when it is empty, you can use the `RESTRICT` option. By default, the `DROP SCHEMA` uses the `RESTRICT` option.

MANAGING DATABASES: SCHEMA

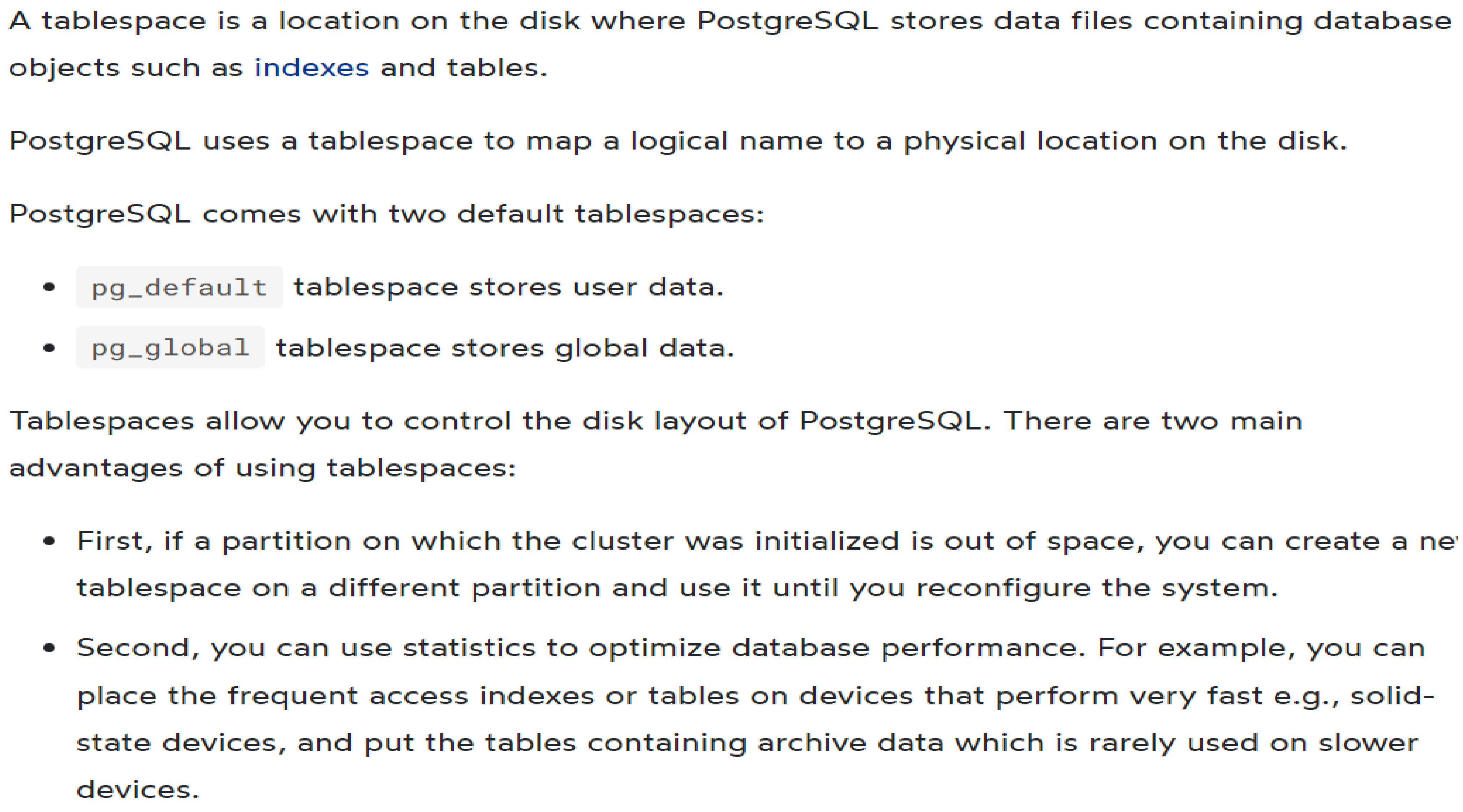
PostgreSQL schema operations

To execute the `DROP SCHEMA` statement, you must be the owner of the schema that you want to drop or a superuser.

PostgreSQL allows you to drop multiple schemas at the same time by using a single `DROP SCHEMA` statement:

```
DROP SCHEMA [IF EXISTS] schema_name1 [,schema_name2,...]
[CASCADE | RESTRICT];
```

MANAGING DATABASES: TABLESPACES



A tablespace is a location on the disk where PostgreSQL stores data files containing database objects such as indexes and tables.

PostgreSQL uses a tablespace to map a logical name to a physical location on the disk.

PostgreSQL comes with two default tablespaces:

- `pg_default` tablespace stores user data.
- `pg_global` tablespace stores global data.

Tablespaces allow you to control the disk layout of PostgreSQL. There are two main advantages of using tablespaces:

- First, if a partition on which the cluster was initialized is out of space, you can create a ne tablespace on a different partition and use it until you reconfigure the system.
- Second, you can use statistics to optimize database performance. For example, you can place the frequent access indexes or tables on devices that perform very fast e.g., solid-state devices, and put the tables containing archive data which is rarely used on slower devices.

MANAGING DATABASES: TABLESPACES

PostgreSQL CREATE TABLESPACE statement

To create new tablespaces, you use the `CREATE TABLESPACE` statement as follows:

```
CREATE TABLESPACE tablespace_name
OWNER user_name
LOCATION directory_path;
```

The name of the tablespace should not begin with `pg_`, because these names are reserved for the system tablespaces.

By default, the user who executes the `CREATE TABLESPACE` is the owner of the tablespace. To assign another user as the owner of the tablespace, you specify it after the `OWNER` keyword.

The `directory_path` is the absolute path to an empty directory used for the tablespace. PostgreSQL system users must own this directory to read and write data into it.

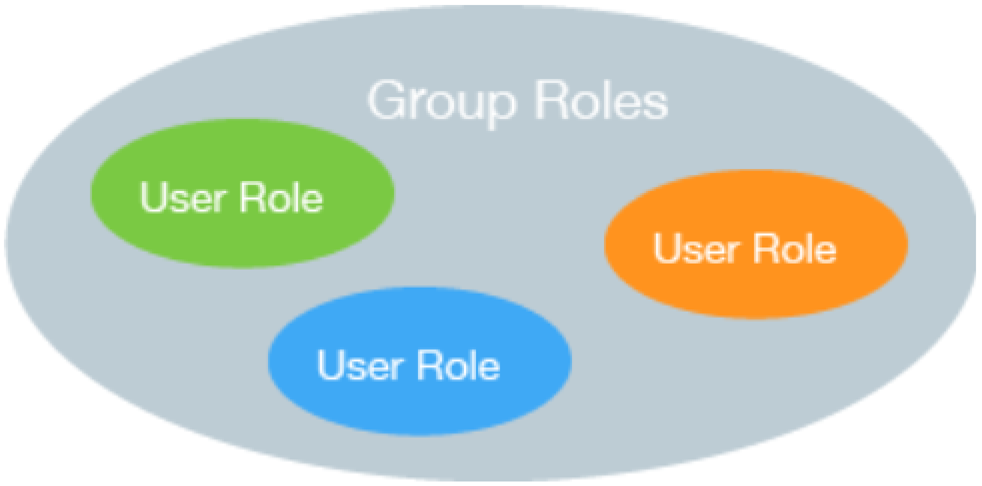
Once a tablespace is created, you can specify it in the `CREATE DATABASE`, `CREATE TABLE` and `CREATE INDEX` statements to store data files of the objects in the tablespace.

MANAGING DATABASES: ROLES

PostgreSQL uses roles to represent user accounts. It doesn't use the user concept like other database systems.

Typically, roles can log in are called login roles. They are equivalent to users in other database systems.

When roles contain other roles, they are call group roles.



> Note that PostgreSQL combined the users and groups into roles since version 8.1