Database

Database is a collection of related data serves as a container for

- tables
- indexes
- views
- etc.

and other database objects.

Create new databse

Syntax:

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

- OWNER: Assign a role(user) that will be the owner of the database. the owner have the highest level of control over the database.

  The owner has full privileges over the database, including the ability to drop it, alter it, and grant/revoke privileges to other users.

  ```
  OWNER = myuser

  -- DEFAULT USER IS postgres
  ```

- TEMPLATE: Specify the template database for the new database. postgres use the [template1] database as default template database.

  - PostgreSQL comes with two default templates:

    - template0: A minimal database with no user-created objects.
    - template1: A more commonly used template that may include objects and customizations.

```
TEMPLATE=template1
```

NOTE:

The database created from a template inherits all objects `(tables, functions, etc.)` in the template. However, certain templates, like `template0`, are clean and have no `user-defined` objects.

- ENCODING:

    - Defines the character encoding for the database.

    - This setting determines how `text` is stored in the `database`.

    - Common encodings include `UTF8` (recommended for most applications).

    - Common values: `UTF-8`,`LATIN1`,`SQL_ASCII`, etc.

    ```
    ENCODING='UTF-8'
    ```

- LC_COLLATE:

    - Specifies the `collation` order to use for string `sorting` and `comparison.`

    - it determines the rules for character ordering in the database.

    - `Collations` are based on `locale` settings and can differ for different `languages` and `regions`.

    - Common values: `en_US.UTF-8`, `fr_FR.UTF-8`, etc

    ```
    LC_COLLATE = 'en_US.UTF-8'
    ```

    NOTE:

    You cannot change these properties after the database is created, so they should be set carefully.

- LC_TYPE: Defines the `character` classification and `case conversion` behavior, such as upper and lower case conversions. like `LC_COLLATE`, this is local-based.

    Common values: `en_US.UTF-8`, `fr_FR.UTF-8`, etc

    ```
    LC_TYPE = 'en_US.UTF-8'
    ```

NOTE:

You cannot change these properties after the database is created, so they should be set carefully.

- TABLESPACE: Specifies the tablespace where the database should reside. A tablespace is a storage location on the disk where the database files are stored.
    - If not specified, PostgreSQL uses the default tablespace.
    - The default tablespace is PostgreSQL is pg_default, which maps to /data directory in PostgreSQL.
    - PostgreSQL also has a second default tablespace called pg_global, which stores global data.

```
TABLESPACE=my_tablespace

-- if you want to print available tablespace use below commands.
SELECT spcname from pg_tablespace;

-- show the physical location of tablespace
show data_directory;

-- To list all tables that are stored in the pg_default tablespace
SELECT tabelname from pg_tables;
```

- ALLOW_CONNECTIONS: Determines whether the database should allow connections. Setting this to false will prevent anyone from connecting to the database, but the database will still exist for administrative purposes.

  Possible values: TRUE|FASLE

```
ALLOW_CONNECTIONS = true
```

- CONNECTION LIMIT: Defines the maximum number of concurrent connections allowed to the database. A -1 value (or omitting this option) means unlimited connections.

```
CONNECTION LIMIT = 100
```

- IS_TEMPLATE: Specifies whether the database should be treated as a template for creating new databases.

  Possible values: TRUE|FALSE

```
IS_TEMPLATE=true
```

```
CREATE DATABASE blogs
  OWNER = pkuser
  TEMPLATE = template1
  ENCODING = 'UTF8'
  LC_COLLATE = 'en_US.UTF-8'
  LC_CTYPE = 'en_US.UTF-8'
  TABLESPACE = my_tablespace
  ALLOW_CONNECTIONS = true
  CONNECTION LIMIT = 100
  IS_TEMPLATE = false;


-- OR
-- rest using default values of each parameter
CREATE DATABASE blogs;

-- retrieve the database names from the `pg_database`
SELECT datname from pg_database;

-- list all database in [psql]
\l

-- connect to created or any database
\c db_name
```

Alter database

ALTER DATABASE statement allow 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 non-runtime configuration variable for a database.

Changing attributes of a database

```
-- syntax
ALTER DATABASE name WITH option;

-- option can be;
-- IS_TEMPLATE
-- CONNECTION LIMIT
-- ALLOW_CONNECTIONS
```

```
-- Only superusers or database owner can change these settings;
```

Rename database:

```
-- syntax
ALTER DATABASE db_name
RENAME TO new_db_name;

-- It is not possible to rename the current database.
-- Only superusers and database owners with [CREATEDB] privilege can rename the
database.
```

Change the owner of the database:

```
-- syntax
ALTER DATABASE db_name
OWNER TO new_owner | current_user | session_user;

-- to check current user or session user run below command
SELECT current_user; -- Returns the current role executing the query
SELECT session_user;  -- Returns the role that authenticated the session
SELECT user;  -- Equivalent to SELECT CURRENT_USER;

-- session information: pg_stat_activity
SELECT usename, application_name, client_addr, backend_start, state
FROM ps_stat_activity
WHERE pid=pg_backend_pid();

-- In PostgreSQL,[ pg_backend_pid()] is a function that returns the[ process ID
(PID)] of the current backend process.

-- how to terminate the current session
SELECT pg_terminate_backend(pg_backend_pid());
```

Change the default tablespace of a database:

```
-- syntax
ALTER DATABASE db_name
SET TABLESPACE new_tablespace;

-- To set the new tablespace, the tablespace needs to be empty and there is a
connection to the database.
-- Superusers and database owners can change the default tablespace of the
database
```

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

```sql
-- syntax
ALTER DATABASE database_name
SET configuration_parameter = value;

-- check the current setting from [pg_settings]
SELECT name, setting FROM pg_settings;

-- change configuration variables for current session
SET <configuration_parameter> TO <value>;


-- examples
SET work_mem TO '64MB'; -- (Memory used for sorting operations)
SET search_path TO my_schema, public; -- (Schema search order)
SET timezone TO 'UTC'; -- (Time zone setting)
SET statement_timeout TO '5min'; -- (Maximum execution time for a statement)
SET log_statement TO 'ddl'; -- (Logging level for SQL statements)
SET default_transaction_isolation TO 'READ COMMITTED'; -- (Transaction isolation
level)

-- Reset configuration variable to default values
RESET <<variable_name>>
-- OR
RESET ALL; -- reset all

-- exmple reset work_mem only
RESET work_mem;
```

Examples:

```sql
-- create database
CREATE DATABASE testdb2;

-- rename to testhrdb
ALTER DATABASE testdb2
RENAME TO testhrdb;

-- change owner postgres to hr
ALTER DATABASE testhrdb
OWNER TO hr;

-- change the default tablespace of the testhrdbfrom pg_default to hr_default
ALTER DATABASE testhrdb
SET TABLESPACE hr_default;
```

```
-- set escape_string_warning configuration variable to off by using the following
statement:
ALTER DATABASE testhrdb
SET escape_string_warning = off;
```

Drop database

The DROP DATABASE statement deletes a database from a PostgreSQL server.

```
-- syntax
DROP DATABASE [IF EXISTS] database_name
[WITH (FORCE)]

-- The FORCE option will attempt to terminate all existing connections to the
target database.
```

NOTE:

- The DROP DATABASE statement deletes the database from both catalog entry and data directory.

- Since PostgreSQL does not allow you to roll back this operation, you should use it with caution.

- To execute the DROP DATABASE statement, you need to be the database owner.

Examples:

```
-- Create some database
CREATE DATABASE hr;
CREATE DATABASE test;

-- Drop the hr database
DROP DATABASE hr;

-- Removing a non-existing database example (IF EXISTS WILL CHECK THE DATABASE
THEN DELETE IF EXISTS OTHERWISE DO NOTHING)
DROP DATABASE IF EXISTS non_existing_database;

-- Drop a database that has active connections example
DROP DATABASE test WITH (FORCE)
```

Rename database

```
-- Create database bots
CREATE DATABASE bots;

-- RENAME TO robots
```

```
ALTER DATABASE bots
RENAME TO robots;
```

Copy database within the same server

You want to copy a PostgreSQL database wihin a database server for testing purpose.

```
-- syntax
CREATE DATABASE targetDb
WITH TEMPLATE sourceDb;

-- example
CREATE DATABASE dvdrental_test
WITH TEMPLATE dvdrental;
```

Copy database from one server to another:

```
-- Step-1: dump the source database into a file.
pg_dump -U postgres -d sourcedb -f sourcedb.sql

-- Step-2: create new database
CREATE DATABSE demo;

-- Step-3: restore the dump file
psql -U postgres -d demo -f sourcedb.sql
```

How to Get Sizes of Database Objects in PostgreSQ

- Use the `pg_size_pretty()` function to format the size.
- Use the `pg_relation_size()` function to get the size of a table.
- Use the `pg_total_relation_size()` function to get the total size of a table.
- Use the `pg_database_size()` function to get the size of a database.
- Use the `pg_indexes_size()` function to get the size of an index.
- Use the `pg_total_index_size()` function to get the size of all indexes on a table.
- Use the `pg_tablespace_size()` function to get the size of a tablespace.
- Use the `pg_column_size()` function to obtain the size of a column of a specific type.

Examples:

```
-- Getting table sizes:
select pg_relation_size('actor');

-- he pg_size_pretty() function formats a number using bytes, kB, MB, GB, or TB
appropriately. For example:
SELECT
    pg_size_pretty (pg_relation_size('actor')) size;
```

```sql
-- To get the total size of a table
SELECT
    pg_size_pretty (
        pg_total_relation_size ('actor')
    ) size;


-- the following query returns the top 5 biggest tables in the dvdrental database
SELECT
    relname AS "relation",
    pg_size_pretty (
        pg_total_relation_size (C .oid)
    ) AS "total_size"
FROM
    pg_class C
LEFT JOIN pg_namespace N ON (N.oid = C .relnamespace)
WHERE
    nspname NOT IN (
        'pg_catalog',
        'information_schema'
    )
AND C .relkind <> 'i'
AND nspname !~ '^pg_toast'
ORDER BY
    pg_total_relation_size (C .oid) DESC
LIMIT 5;


-- Getting database size
SELECT
    pg_size_pretty (
        pg_database_size ('dvdrental')
    ) size;

-- Getting index sizes
SELECT
    pg_size_pretty (pg_indexes_size('actor')) size;

-- Getting tablespace sizes
SELECT
    pg_size_pretty (
        pg_tablespace_size ('pg_default')
    ) size;

-- Getting PostgreSQL value sizes
SELECT
  pg_column_size(5 :: smallint) smallint_size,
  pg_column_size(5 :: int) int_size,
  pg_column_size(5 :: bigint) bigint_size;
```

PostgreSQL Schema

In PostgreSQL, a schema is a named collection of database object, including

- Tables
- Indexes
- Views
- Data Types
- Functions
- Procedures
- Operators
- Sequences
- Triggers
- Materialized Views
- Domains
- Aggregates
- Collations
- Foreign Tables

and many more.

A schema allows you to organize and namespace objects within a database.

A schema can be thought of as a `container` for `database` objects.

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

Access an object in schema:

```
-- syntax
schema_name.object_name

-- example
SELECT * FROM public.users; -- public is schema
```

For example:

You may have `auth` schema that has `user` table and `public` schema which also has the `user` table.

```
public.user
-- Or
auth.user
```

Advantage of using schema:

- Schemas allow you to organize database objects, tables into logical group to make them more managabale.
- Schema enable multiple users to use one database without interfering with each other.

The public schema:

PostgreSQL autometically 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.

```sql
CREATE TABLE table_name(
  ...
)

-- and
CREATE TABLE public.table_name(
  ....
)
```

Schema search path:

When you refer to a table name without its schema name. `user` table instead of a fully qualified name such as `public.user` table.

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 if the name exists in another schema in the database.

The first schema in the search path is called the current schema.

```sql
-- access current schema
SELECT current_schema(); -- it will return the current schema, default is public

-- To view the current search path
SHOW search_path; -- RESULT "$user", public
```

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 cxample:

If you use the `postgres` user to log in and access `user` table. PostgreSQL will search for the `user` 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 referes to the `public` schema as we have seen in the result.

List schema of current database:

```
-- run this command in psql
\dn
```

Add new created schema to the search path:

```
-- suppose you have created new [auth] schema
SET search_path TO auth, public;
```

Now, if you create new table named user without specifying the schema name, PostgreSQL will put this user table into the auth schema:

```sql
CREATE TABLE user(
    user_id int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    age int NOT NULL CHECK(age > 0 and age < 100),
    mobile VARCHAR(10) NOT NULL CHECK(LENGTH(TRIM(mobile))=10)
)
```

The user table belongs to the auth schema.

To access it

```sql
SELECT * FROM user;
-- OR
SELECT * FROM auth.user;
```

The public schema is the second element in the search path

```sql
-- you can set public
SET search_path TO public;
```

The public schema is not a special schema, you can drop it too.

PostgreSQL schemas and previleges:

User can only access objects in the schema that they own (USAGE previlege).

To allow users to access the objects in the schema that they do not own, you must grant the USAGE previlege of the schema to the users.

```
-- syntax
GRANT ON SCHEMA schema_name
TO role_name;

-- example
GRANT USAGE ON SCHEMA auth
TO pkumar;
```

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:

```
-- syntax
GRANT CREATE ON SCHEMA schema_name
TO user_name

-- example
GRANT CREATE ON SCHEMA auth
TO pkumar;
```

Note:

By default, every user has the `CREATE` and `USAGE` on the public schema.

Create schema:

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

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

-- You can also create schema for a user
CREATE SCHEMA [IF NOT EXISTS]
AUTHORIZATION username;
-- In this case, the schema will have the same name as the username.
```

Note: To execute the `CREATE SCHEMA` statement, you must have the `CREATE` privilege in the current database.

To return all schemas from the current database:

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

Using CREATE SCHEMA statement to create a schema for a user example:

First, create a new role with named pkumar

```
CREATE ROLE pkumar
LOGIN
PASSWORD '12345';
```

Second, create a schema for pkumar.

```
CREATE SCHEMA AUTHORIZATION pkumar;
-- OR
CREATE SCHEMA IF NOT EXISTS demo AUTHORIZATION pkumar;
```

Alter schema:

The ALTER SCHEMA statement allows you to change the definition of a schema.

Rename schema:

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

Note

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

Change the owner of schema:

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

Examples:

```
-- Rename schema [demo] to [demo1`]
ALTER SCHEMA demo
RENAME TO demo1;

-- Change the owner
ALTER SCHEMA demo
OWNER TO postgres;
```

Drop schema

The DROP SCHEMA removes a schema and all of its objects from a database.

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

- 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.

Note:

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

Example:

```
-- drop schema demo
DROP SCHEMA IF EXISTS demo;

-- drop multiple schema
DROP SCHEMA IF EXISTS demo, auth, private;

-- drop schema if not empty
DROP SCHEMA blogs CASCADE;
```