**Postgres**

Postgres: The world’s most advanced open source relational database management system.

Install postgreSql : password 1234

To check sql version

SELECT version();

See all databases \l

Connect other database \c databaseName

See relation or table \d

See list of schema: \dn

Clear console \! cls

See only table \dt

See table with extra info \d+

See all database user \du

See connection information \conninfo

Create database: create database dabaseName;

Quit \q

To create Database: CREATE DATABASE databaseName

# Exploring PSQL and its Default Behaviour, Creating Database

If powershell or other terminal not work this command: psql (show error).

Then setup:

Press win-> search env -> Environment Variable -> (system variables find) path -> edit -> New ->

Then go to file Explorer -> c drive -> program files -> 17(virson) -> bin -> copy directory and past in path save.

Open shell and run psql

Then -> psql -U postgres(default username) -> enter -> password -> enter

-: User, Role And Privilege Management in Postgres :-

Create user : CREATE USER user1 WITH LOGIN ENCRYPTED PASSWORD '1234'

Create role: CREATE ROLE role1 WITH LOGIN ENCRYPTED PASSWORD '1234'

To create Table: create table test\_table (name varchar(50));

Insert data in table: insert into test\_table(name) values('mezba');

Postgres database in postgres user : To see table: SELECT \* FROM test\_table;

///show data

Another shell

Go to user1 with postgres database :

psql -U user1 -d postgres

To see table: SELECT \* FROM test\_table;

//access denied

You need to access permission to see database information(Full access)

Main Shell

grant all privileges on table test\_table to user1;

Only select permission:

grant select on table test\_table to user2

Remove permission:

revoke select on table test\_table from user1

Give **user2** permission to do **everything** (read, write, update, delete, etc.) on **all existing tables** inside the **public schema**.

grant all privileges on all tables in schema public to user2

If you create a new table **after** this command, user2 won’t automatically get access to it.

To also give permissions for **future** tables, use this:

ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO user2;

Allow role1 to **read (SELECT)** data from **all existing tables** in the public schema.

grant select on all tables in schema public to **role1**;

Give **user3** the **permissions and access rights** of **role1. In other word**

* user3 becomes a **member of role1**
* user3 inherits **all privileges** granted to role1

grant role1 to user3

To Check Permissions:

\z table\_name

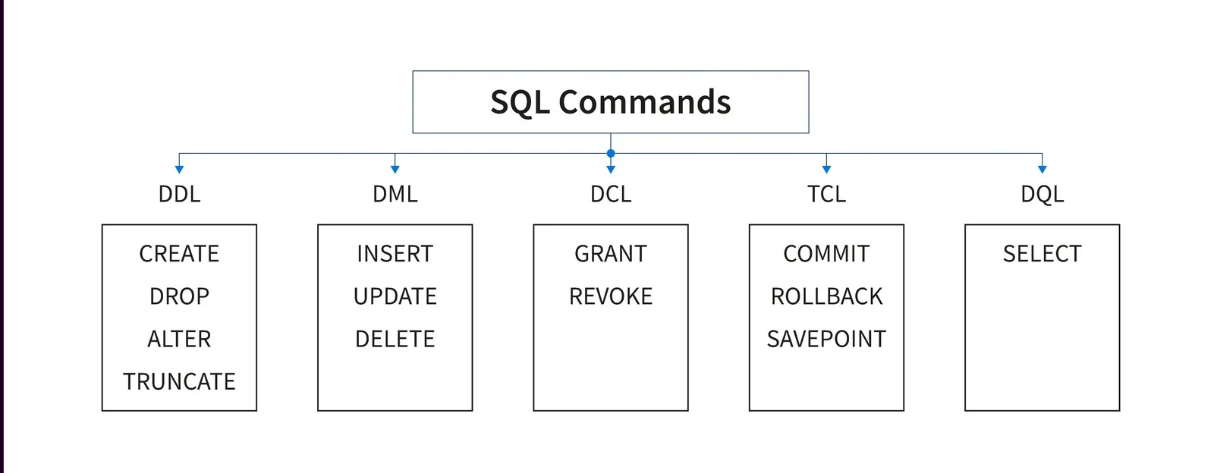
Another shell

Now run this command

SELECT \* FROM test\_table;

You see table info

-: Structured Query Language(SQL):-



DDL -> (Data definition language)

DML -> (Data manipulation/modification language)

DCL ->(Data control language)

TCL ->(Transaction control language)

Dql -> (Data query language)

**Valentina Studio install**

**Valentina Studio is a graphical database management tool used to work with different databases like PostgreSQL, MySQL, SQLite, MariaDB, and more.**

**Create, update and delete database and data types in postgres**

Create database: create database dabaseName;

Change database: alter database oldName rename to newName;

Delete database: drop database databaseName;

**Data Types:**

1. Boolean (true, false, null)
2. Numbers
3. Binary
4. Date/Time(DATE, TIME, TIMESTAMP, TIMESTAMPTZ, INTERVAL)
5. Json
6. Character
7. UUID
8. Array
9. XML

## **Integers**

### **INT:**

* **Range**: -2,147,483,648 to 2,147,483,647
* **Storage**: 4 bytes
* The INT data type is commonly used for integer values within the standard range.

### **BIGINT:**

* **Range**: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,80
* **Storage**: 8 bytes
* BIGINT provides a larger range than INT and is suitable for storing very large integer values.

### **SMALLINT:**

* **Range**: -32,768 to 32,767
* **Storage**: 2 bytes
* SMALLINT is suitable for storing small integer values within a more limited range.

### **FLOAT4 (Single Precision):**

* **Storage**: 4 bytes
* **Precision**: 6-digit precision

### **FLOAT8 (Double Precision):**

* **Precision**: 15-digit precision
* **Storage**: 8 bytes

### **NUMERIC(precision, scale):**

* **Range**: Unlimited precision
* **Storage**: Variable (depends on the input precision and scale)
* **Example**: NUMERIC(4,3)

### **Serial:**

* **Range**: Like integers
* **Use**: Auto-incrementing integer

## **Characters**

### **CHAR:**

* **CHAR is a fixed-length character type.**
* **Storage**: The length is fixed and specified when defining the column.
* If the actual string is shorter than the specified length, it is padded with spaces.
* **Example**: CHAR(10) means a fixed-length string of 10 characters.

### **VARCHAR:**

* **VARCHAR is a variable-length character type.**
* **Storage**: The length is not fixed and can vary up to the specified maximum length.
* No padding with spaces is done, so it is more storage-efficient when dealing with variable-length strings.
* **Example**: VARCHAR(255) means a variable-length string with a maximum length of 255 characters.

### **TEXT:**

* **TEXT is also a variable-length character type.**
* **Storage**: Similar to VARCHAR, it is not fixed in length.  
  Typically used for longer text strings where the exact length is not known or can vary widely.
* **Example**: TEXT is often used for columns containing large amounts of text.

**Creating a table with multiple columns and different data types(valentina):**

CREATE TABLE table\_name

(

Column1 datatype constraint,

Column2 datatype constraint,

Column3 datatype constraint,

**…**

)

**Column constraints**

Not null

CREATE TABLE example(

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL

)

**Unique**

CREATE TABLE example(

id SERIAL PRIMARY KEY,

Email VARCHAR(255) UNIQUE

);

**Foreign Key**

CREATE TABLE orders(

order\_id SERIAL PRIMARY KEY,

Customer\_id INTEGER REFERENCES customers (customer\_id)

);

**Default**

CREATE TABLE example(

id SERIAL PRIMARY KEY,

status BOOLEAN DEFAULT true

);

**Check**

CREATE TABLE example(

id SERIAL PRIMARY KEY,

Age INTEGER CHECK (age >= 18)

);

Example:

CREATE TABLE person(

person\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

Last\_name VARCHAR(50) NOT NULL,

is\_active BOOLEAN DEFAULT TRUE,

age INTEGER CHECK (age >= 0),

email VARCHAR(255) UNIQUE

)

**Inserting data into a Table and Checking Constraints:**

**Single-Row Insert:**

INSERT INTO your\_table (column1, column2, column3)

VALUES(value1, value2, value3)

Example:

INSERT INTO employees (first\_name, last\_name, hire\_date)

VALUES(‘Jhon’, ‘Doe’, ‘2022-01-15’)

**Multi-Row Insert:**

INSERT INTO you\_table (column1,column2,column3)

VALUES

(value1\_1, value2\_1, value3\_1),

(value1\_2, value2\_2, value3\_2),

………………..;.

example:

INSERT INTO employees (first\_name, last\_name, hire\_date)

VALUES

(‘Jan’, ‘Smith’, ‘2022-02-25’),

(‘Bob’, ‘Jhonson’, ‘2021-08-2021’);

**Alter:**

ALTER TABLE table\_name

action(rename a table, add/drop column, modify data type of a column, setting default value for a column, rename a column, add/drop constraint for a column) ;

**Select:**

The SELECT statement is used to retrieve data from one or more tables and can be customized with conditions, sorting, and other clauses.

**SELECT**: Retrieves data from one or more tables.

**FROM**: Specifies the table from which to retrieve data.

**WHERE**: Filters data based on specified conditions.

**ORDER BY**: Sorts the result set based on specified columns.

**GROUP BY**: Groups rows that have the same values in specified columns.

**HAVING**: Filters the results of a GROUP BY clause based on specified conditions.  
**JOIN**: Combines rows from two or more tables based on a related column.

**DISTINCT**: Returns unique values in the result set.  
**LIMIT**: Specifies the maximum number of rows to return.  
**OFFSET**: Specifies the number of rows to skip before starting to return rows.

**Functions:**

1. **Scalar**
2. **Aggregate**

**Scalar:**

Scalar functions operate on a single value and return a single value. They perform an operation on each row’s data independently.

Scalar functions:

UPPER() : Converts a string to uppercase.

LOWER(): Converts a string to lowercase.

CONCAT(): Concatenates two or more strings.

LENGTH(): Returns the number of characters in a string.

**Aggregate:**

Aggregate functions operate on a set of values and return a single value summarizing that set. They perform an operation across multiple rows, often used with the GROUP BY clause.

Aggregate functions:

AVG(): Calculate the average of a set of values.

MAX(): Returns the max value in a set.

MIN(): Returns the minimum value in a set.

SUM(): Calculate the sum of values in a set.

COUNT(): Counts the number of rows in a set.