PostgreSQL

Fanavaran Anisa

Iran Linux House

Linux & Open Source Training Center

www.anisa.co.ir

Section 2 : DDL & PSQL

Section 2 Overview

Section 1: Basics of DDL and Table Creation

- Basic DDL Commands
- Keys
- Data Types
- Identity Fields

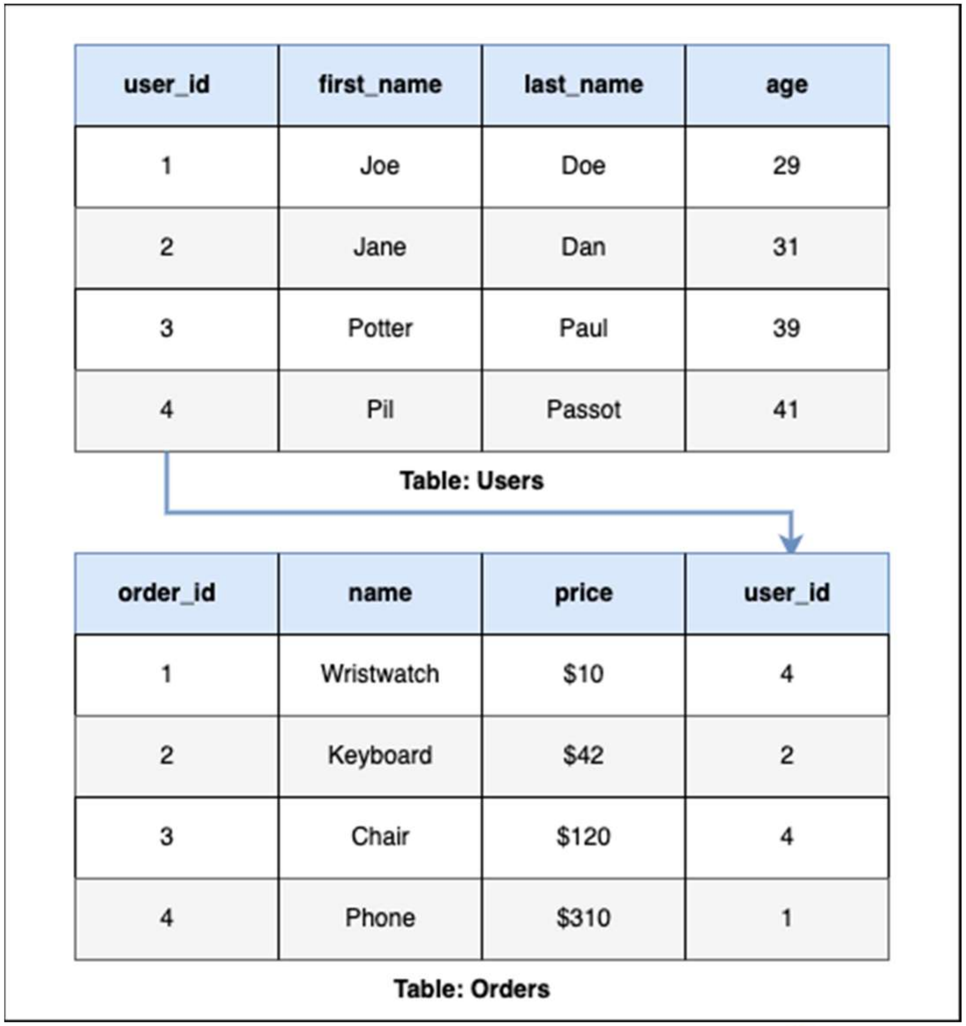
Section 2: Table Alteration

- A closer look at the Sequnces
- Alter/Drop

Section 3: Some Advanced Concepts

SQL

-**Structured Query Language (SQL)** is a query language used with relational databases such as MySQL, Oracle, MSSQL, PostgreSQL, and many others. It is a query language that you can use to create and delete databases and tables, insert and read data into tables, delete data from tables, and much more (and Query the data).



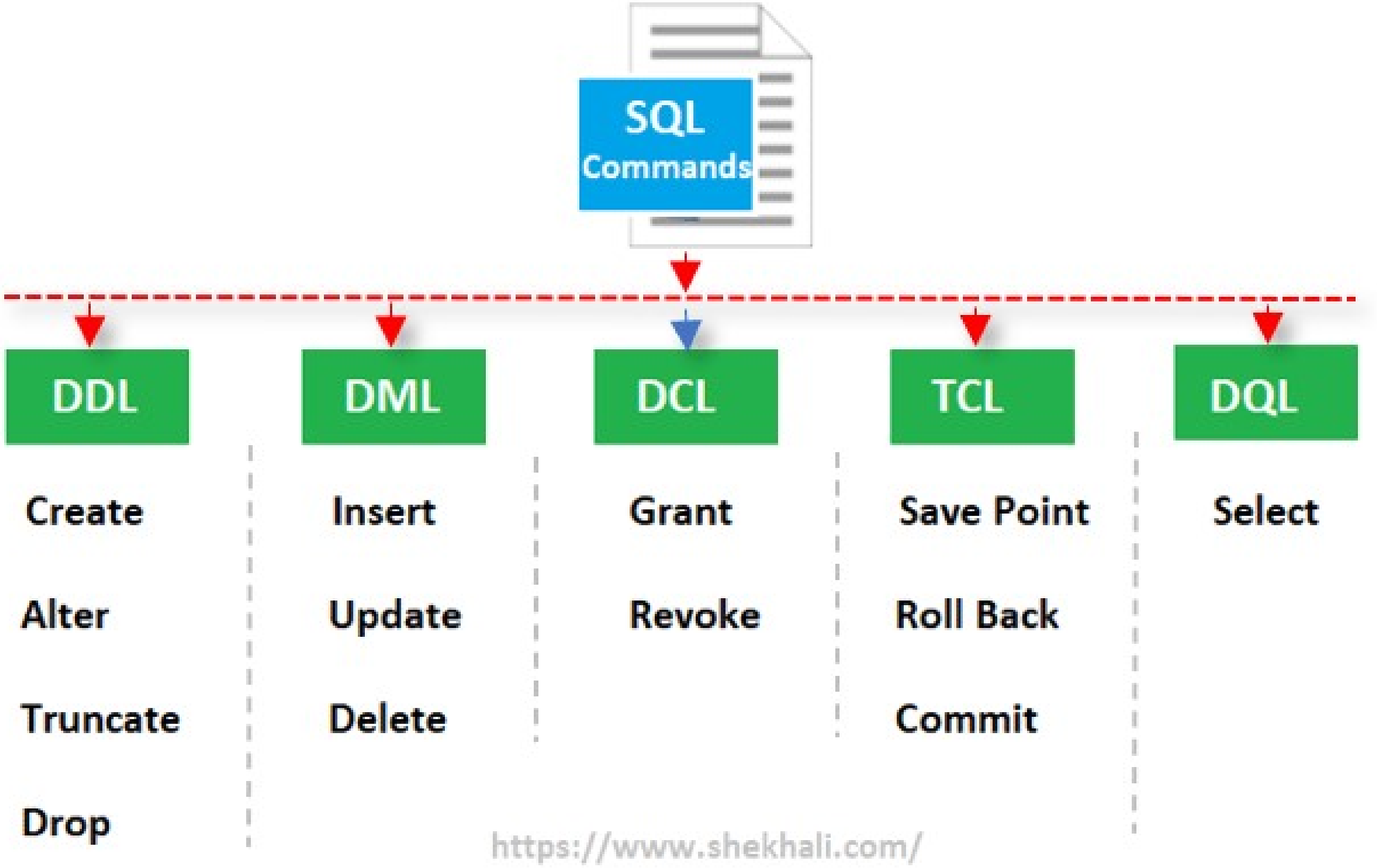
user_id	first_name	last_name	age
1	Joe	Doe	29
2	Jane	Dan	31
3	Potter	Paul	39
4	Pil	Passot	41

Table: Users

order_id	name	price	user_id
1	Wristwatch	$10	4
2	Keyboard	$42	2
3	Chair	$120	4
4	Phone	$310	1

Table: Orders

SQL Command Types

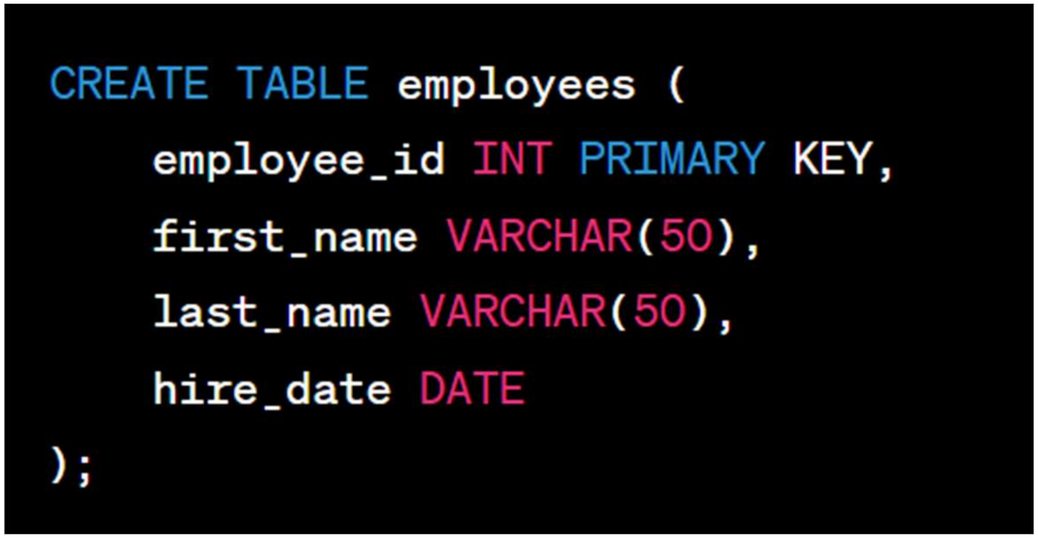


DDL	DML	DCL	TCL	DQL
Create	Insert	Grant	Save Point	Select
Alter	Update	Revoke	Roll Back	
Truncate	Delete		Commit	
Drop				

https://www.shekhali.com/

DDL

- **DDL stands for Data Definition Language**
- DDL statements are responsible for defining, altering, and dropping database objects, such as tables, indexes, and views.



```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    hire_date DATE
);
```

Overview of DDL Commands

Schema and Database Commands:

- **CREATE SCHEMA schema_name;** - Creates a new schema.

- **CREATE DATABASE database_name;** - Creates a new database.
- **CREATE DATABASE database_name TEMPLATE template_name;** - Creates a new database based on a template.
- **DROP SCHEMA schema_name CASCADE;** - Drops a schema and its objects.
- **DROP DATABASE database_name;** - Drops a database

Table Creation Commands:

CREATE TABLE table_name (column1 datatype1, column2 datatype2, ...); - Creates a new table.

ALTER TABLE table_name ADD COLUMN column_name datatype; - Adds a new column to an existing table.

ALTER TABLE table_name DROP COLUMN column_name; - Drops a column from a table.

Column Constraints & Primary Key:

NOT NULL - Ensures that a column cannot have NULL values.

UNIQUE - Ensures that all values in a column are unique.

CHECK (condition) - Adds a check constraint.

DEFAULT default_value - Provides a default value for a column.

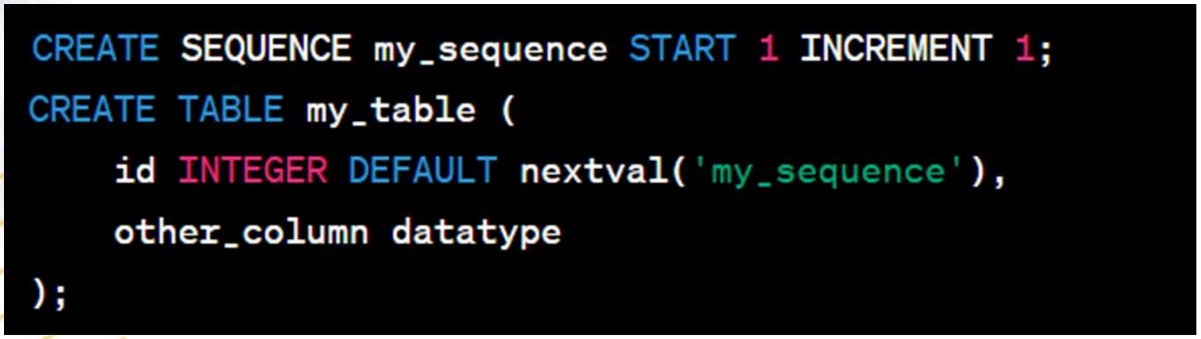
PRIMARY KEY (column1, column2, ...) - Defines a primary key on one or more columns.

FOREIGN KEY (column_name) REFERENCES parent_table(parent_column);

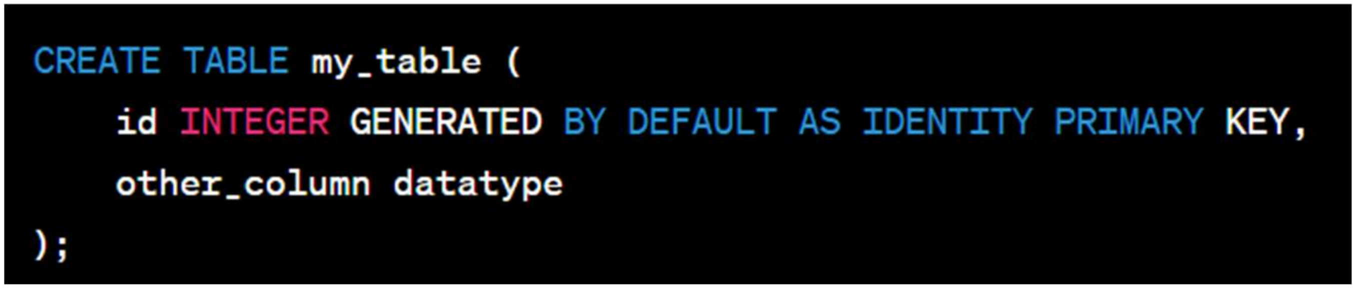
SEQUENCE, IDENTITY, and SERIAL

- SEQUENCE is a generic SQL standard concept for generating a sequence of numbers.
  - provides the most flexibility but requires more manual setup.
- IDENTITY is part of the SQL standard and provides a standardized way of defining auto-incrementing columns.
  - is more standard-compliant and is recommended for better cross-database compatibility
- SERIAL is a PostgreSQL-specific type that is often used to create auto-incrementing columns.
  - SERIAL is a PostgreSQL-specific shortcut that is widely used for simplicity.

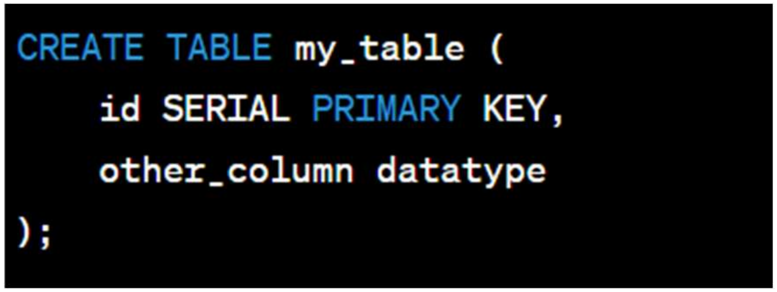
SEQUENCE, IDENTITY, and SERIAL



```
CREATE SEQUENCE my_sequence START 1 INCREMENT 1;
CREATE TABLE my_table (
    id INTEGER DEFAULT nextval('my_sequence'),
    other_column datatype
);
```



```
CREATE TABLE my_table (
    id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    other_column datatype
);
```



```
CREATE TABLE my_table (
    id SERIAL PRIMARY KEY,
    other_column datatype
);
```

Integer Keys vs UUID

- **A Sample UUID** : de1aa74c-270c-4843-b45a-cc1b0aa4676f

- **UUID Usage :**
- No sequential pattern (Not Predictability)
- Global Uniqueness & Concurrent Parts
- No Central Coordination (Distributed Systems)

Cons :
- Larger storage size
- Slower indexing

Standard Data Types – Numeric Types

Data Type	Description	Example
INTEGER	Integer values	age INTEGER
SMALLINT	Small integer values	quantity SMALLINT
BIGINT	Large integer values	balance BIGINT
DECIMAL(p, s)	Fixed-point numbers with precision p	price DECIMAL(10,2)
NUMERIC(p, s)	Synonym for DECIMAL	quantity NUMERIC(8,2)
REAL	Single-precision floating-point numbers	weight REAL
DOUBLE PRECISION	Double-precision floating-point numbers	height DOUBLE PRECISION

Standard Data Types – Character Types

Data Type	Description	Example
CHAR(n)	Fixed-length character strings	name CHAR(50)
VARCHAR(n)	Variable-length character strings	address VARCHAR(255)
TEXT	Variable-length character strings	comments TEXT
NCHAR(n)	Fixed-length Unicode character strings	n_name NCHAR(50)
NVARCHAR(n)	Variable-length Unicode character strings	n_address NVARCHAR(255)

Standard Data Types – Datetime Types

Data Type	Description	Example
DATE	Date values	birth_date DATE
TIME	Time values	event_time TIME
TIMESTAMP	Date and time values	created_at TIMESTAMP
INTERVAL	Time intervals	duration INTERVAL

```
INSERT INTO event_table (birth_date, event_time, created_at, duration) VALUES
('1990-05-15', '14:30:00', '2023-01-01 08:45:30', '2 days 3 hours’),
('1985-08-22', '18:15:45', '2023-01-02 12:30:00', '1 week 4 days 6 hours’),
('2000-11-10', '09:00:00', '2023-01-03 15:20:10', '4 hours 30 minutes');
```

Standard Data Types – Miscellaneous Types

Data Type	Description	Example
BOOLEAN	Boolean values	is_active BOOLEAN
BINARY	Binary large objects	image BINARY
VARBINARY(n)	Variable-length binary strings	blob VARBINARY(1000)
UUID	Universally unique identifier	session_id UUID
CLOB	Character large objects	long_text CLOB

- Use **CLOB** when dealing with extremely large documents or when the content may exceed typical character data size limitations.
- Use **TEXT** for storing large paragraphs or documents where the length is not known in advance.
- Use **VARCHAR** when you want to enforce a maximum length for your string data.
- Use **Binary** for small binary files such as profile pic , otherwise use the OID

Postgres Data Types – Numeric Types

Data Type	Description	Example
SMALLINT	Small integer values	quantity SMALLINT
INTEGER	Integer values	age INTEGER
BIGINT	Large integer values	balance BIGINT
DECIMAL(p, s)	Fixed-point numbers with precision p	price DECIMAL(10,2)
NUMERIC(p, s)	Synonym for DECIMAL	quantity NUMERIC(8,2)
REAL	Single-precision floating-point numbers	weight REAL
DOUBLE PRECISION	Double-precision floating-point numbers	height DOUBLE PRECISION
SERIAL	Auto-incrementing integer	id SERIAL PRIMARY KEY
BIGSERIAL	Large auto-incrementing integer	id BIGSERIAL PRIMARY KEY

Postgres Data Types - Character Types

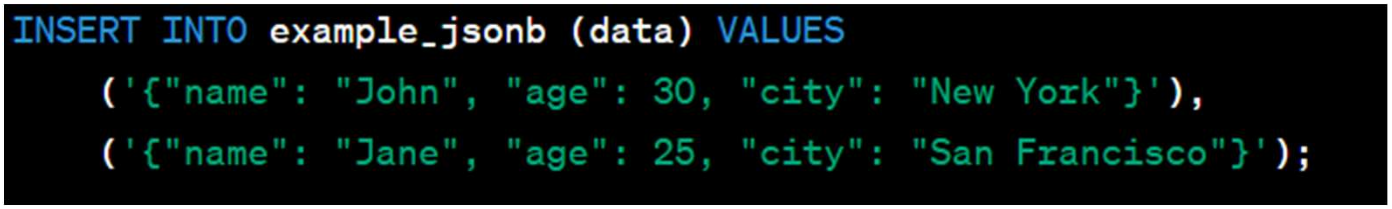
Data Type	Description	Example
CHAR(n)	Fixed-length character strings	name CHAR(50)
VARCHAR(n)	Variable-length character strings	address VARCHAR(255)
TEXT	Variable-length character strings	comments TEXT
CHAR VARYING(n)	Synonym for VARCHAR	title VARCHAR(100)
CHARACTER(n)	Synonym for CHAR	code CHARACTER(10)
CHARACTER VARYING(n)	Synonym for VARCHAR	description VARCHAR(500)

Postgres Data Types - Date/Time Types

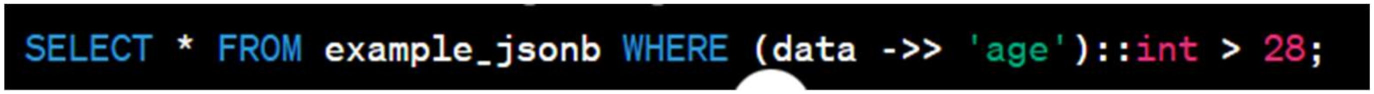
Data Type	Description	Example
DATE	Date values	birth_date DATE
TIME	Time values	event_time TIME
TIMESTAMP	Date and time values	created_at TIMESTAMP
TIMESTAMPTZ	Timestamp with time zone	logged_at TIMESTAMPTZ
INTERVAL	Time intervals	duration INTERVAL

Postgres Data Types - Miscellaneous Types

Data Type	Description	Example
BOOLEAN	Boolean values	is_active BOOLEAN
UUID	Universally unique identifier	session_id UUID
JSON	JSON data type	data JSON
JSONB	Binary JSON data type	binary_data JSONB
BYTEA	Binary data	image BYTEA
OID	Object identifier	document_oid OID
XML	XML data type	xml_data XML
CITEXT	Case-insensitive text	case_insensitive CITEXT



```
INSERT INTO example_jsonb (data) VALUES
    ('{"name": "John", "age": 30, "city": "New York"}'),
    ('{"name": "Jane", "age": 25, "city": "San Francisco"}');
```

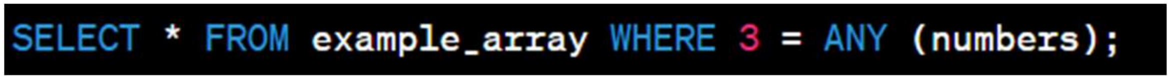


```
SELECT * FROM example_jsonb WHERE (data ->> 'age')::int > 28;
```

Postgres Data Types - Network Address & Arrays and Enumerations

Data Type	Description	Example
INET	IPv4 or IPv6 network address	ip_address INET : '192.168.1.0'
CIDR	IPv4 or IPv6 network address with mask	subnet CIDR : '192.168.1.0/24'
MACADDR	MAC (Media Access Control) address	mac_address MACADDR : 08:00:2B:01:02:03

Data Type	Description	Example
ARRAY	Array data type	numbers INTEGER[] : '{1, 2, 3, 4}'
ENUM	Enumeration data type	status ENUM('active', 'inactive')



```
SELECT * FROM example_array WHERE 3 = ANY (numbers);
```

Postgres Data Types – Array Sample

- Define a table with an array of fixed-char phone numbers

```
CREATE TABLE contact (
              id SERIAL PRIMARY KEY,
             phone_numbers CHAR(10)[3]
)
 -- Insert data into the table
INSERT INTO contact (phone_numbers) VALUES
('1234567890', '9876543210', '5551234567’),
('1112223333', '4445556666’);
```

```
INSERT INTO contact (phone_numbers) VALUES ('1234567890', '9876543210',
'5551234567', '9998887777');
```

Postgres Data Types - Geometric Types

Data Type	Description	Example
POINT	Geometric point	location POINT: (2.3, 4.5)
LINE	Geometric line segment	path LINE :({1,2}, {3,4})
LSEG	Line segment	line_segment LSEG:((1,2), (3,4))
PATH	Geometric path	route PATH:((1,2), (3,4), (5,6))
POLYGON	Geometric polygon	boundary POLYGON:((1,2), (3,4), (5,6))
CIRCLE	Geometric circle	disk CIRCLE:((1,2), 5)

Line:

- Represents an infinite straight line in two-dimensional space.
- Defined by two distinct points.
- Equation of a line: Ax + By + C = 0.
- Example: **LINE '((1,2),(3,4))'** represents the line passing through the points (1,2) and (3,4).

Lseg:

- Represents a line segment in two-dimensional space.
- Defined by two distinct points.
- Example: **LSEG '((1,2),(3,4))'** represents the line segment between the points (1,2) and (3,4).

Postgres Data Types – Hstore (Hash Store)

Data Type	Description	Example
HSTORE	Key-value store	properties HSTORE -> ' "color" => "red", "size" => "large" '
		SELECT * FROM example_hstore WHERE properties -> 'color' = 'red';

Use Hstore When:

•You have a fixed set of key-value pairs.

•Quick retrieval and indexing are essential.

•The structure is relatively flat.

Use JSON When:

•Your data has a hierarchical or dynamic structure.

•Flexibility is required for handling varying attributes.

•You want to take advantage of indexing and querying capabilities offered by JSONB

Postgres Data Types - User Defined Types

```
CREATE TYPE measurement AS (
    value DOUBLE PRECISION,
    unit TEXT
);

CREATE TABLE product_dimensions (
    id SERIAL PRIMARY KEY,
    length measurement,
    width measurement,
    height measurement
);

INSERT INTO product_dimensions (length, width, height)
VALUES
    ((10.0, 'cm'), (5.0, 'cm'), (3.0, 'cm')),
    ((20.0, 'in'), (8.0, 'in'), (5.0, 'in'));
```

Postgres Data Types - User Defined Types

```
CREATE TYPE address AS (
    street VARCHAR(100),
    city VARCHAR(50),
    state VARCHAR(20),
    zip_code VARCHAR(10)
);

CREATE TABLE customer_addresses (
    id SERIAL PRIMARY KEY,
    location address
);

INSERT INTO customer_addresses (location) VALUES
    (('123 Main St', 'Cityville', 'CA', '90210')),
    (('456 Elm St', 'Townsville', 'NY', '10001'));
```

Postgres Data Types - User Defined Types

```
CREATE TYPE task_status AS ENUM ('TODO', 'IN_PROGRESS',
'DONE');
CREATE TYPE task_list AS task_status[];

CREATE TABLE task_lists (
    id SERIAL PRIMARY KEY,
    tasks task_list
);

INSERT INTO task_lists (tasks) VALUES
    ('{"TODO", "DONE"}'),
    ('{"IN_PROGRESS", "TODO", "DONE"}');
```

Postgres Data Types - User Defined Types

```
CREATE TYPE person AS (
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE
);

CREATE TABLE people (
    id SERIAL PRIMARY KEY,
    info person
);

INSERT INTO people (info) VALUES
    (('John', 'Doe', '1990-05-15')),
    (('Jane', 'Smith', '1985-08-20'));
```

Postgres Data Types - User Defined Types

```sql
CREATE TYPE complex AS (
    real_part DOUBLE PRECISION,
    imag_part DOUBLE PRECISION,
    FUNCTION complex_in(cstring TEXT) RETURNS complex,
    FUNCTION complex_out(c complex) RETURNS TEXT
);

CREATE TABLE complex_numbers (
    id SERIAL PRIMARY KEY,
    value complex
);

INSERT INTO complex_numbers (value) VALUES
    ('(3.0, 4.0)'),
    ('(-1.5, 2.5)');

-- Query using custom output function
SELECT id, complex_out(value) AS complex_representation FROM
complex_numbers;
```

Postgres Data Types - User Defined Types

```
-- Output function implementation
CREATE OR REPLACE FUNCTION complex_out(c complex) RETURNS TEXT AS
$$
BEGIN
    -- Return the textual representation of the complex number
    RETURN '(' || c.real_part || ',' || c.imag_part || ')';
END;
$$ LANGUAGE plpgsql;
```

Postgres Data Types - User Defined Types

```
CREATE TYPE complex AS (
    real_part DOUBLE PRECISION,
    imag_part DOUBLE PRECISION,
    FUNCTION complex_add(c1 complex, c2 complex) RETURNS complex
);

CREATE TABLE complex_numbers (
    id SERIAL PRIMARY KEY,
    value complex
);

INSERT INTO complex_numbers (value) VALUES
    ('(1.0, 2.0)'),
    ('(3.0, 4.0)');

-- Query using overloaded addition operator
SELECT id, complex_add(value, ROW(5.0, 6.0)) AS result_addition
FROM complex_numbers;
```

Postgres Data Types - How to Create User Defined Types?

```
CREATE TYPE udt_name AS (
    attribute1 data_type1,
    attribute2 data_type2,
    -- Additional attributes
    attribute3 data_type3,
    -- ...

    -- Optional: Adding constraints
    CONSTRAINT constraint_name CHECK (expression),
    -- ...

    -- Optional: Adding methods
    MEMBER FUNCTION method_name(parameters) RETURNS
return_type,
    -- ...
);
```

Postgres Data Types - How to Create User Defined Types?

```
-- Create a UDT representing a person with additional
attributes and constraints
CREATE TYPE person_udt AS (
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    email VARCHAR(100),

    -- Additional attributes
    address VARCHAR(255),

    -- Constraints
    CONSTRAINT valid_email CHECK (email ~* '^[a-zA-Z0-
9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'),

    -- Method
    MEMBER FUNCTION get_full_name() RETURNS VARCHAR(100)
);
```

Postgres Data Types - User Defined Types

```sql
-- Sample table using the UDT
CREATE TABLE people (
    id SERIAL PRIMARY KEY,
    personal_info person_udt
);

-- Inserting data
INSERT INTO people (personal_info) VALUES
    (ROW( 'John', 'Doe', '1990-05-15',
'john.doe@example.com', '123 Main St')),
    (ROW( 'Jane', 'Smith', '1985-08-20',
'jane.smith@example.com', '456 Elm St'));
```

Create Object - Schema

```
- CREATE SCHEMA [ IF NOT EXISTS] schema_name
[AUTHORIZATION owner_name];

COMMENT ON SCHEMA schema_name IS 'Description of the
schema';
```

CREATE SCHEMA **schema_name**;	Create a new schema.
CREATE TABLE **schema_name**.table_name (...)	Create a table in a specific schema.
CREATE INDEX index_name ON **schema_name**.table_name (column_name);	Create an index in a specific schema.
CREATE SEQUENCE **schema_name**.sequence_name;	Create a sequence in a specific schema.
CREATE FUNCTION **schema_name**.function_name (...) RETURNS ... AS $$ ... $$ LANGUAGE plpgsql;	Create a function in a specific schema.

Create Object - Database

```
CREATE DATABASE database_name
    [ [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ LC_COLLATE [=] collate ]
           [ LC_CTYPE [=] ctype ]
           [ TABLESPACE [=] tablespace_name ]
           [ CONNECTION LIMIT [=] connlimit ] ];
```

```
CREATE DATABASE mydatabase
    WITH OWNER = myuser
    TEMPLATE = template0
    ENCODING = 'UTF8'
    LC_COLLATE = 'en_US.utf8'
    LC_CTYPE = 'en_US.utf8'
    TABLESPACE = mytablespace
    CONNECTION LIMIT = -1;
```

In PostgreSQL, a tablespace is a location on disk where the database stores its data files

```
CREATE TABLESPACE mytablespace
    OWNER myuser
    LOCATION
'/path/to/tablespace';
```

Create Object - Database

Optional Parameters:

- **OWNER [=] user_name:** Specifies the user who will be the owner of the database. If not specified, the user executing the **CREATE DATABASE** command becomes the owner.
- **TEMPLATE [=] template:** Specifies a template database from which the new database will be created. The default template is usually **template1**.
- **ENCODING [=] encoding:** Specifies the character encoding scheme for the new database. Common values include **UTF8**, **LATIN1**, etc.
- **LC_COLLATE [=] collate:** Specifies the collation order to be used in the new database. This parameter influences the sort order/Comparisons. (LC: LoCale)
- **LC_CTYPE [=] ctype:** Specifies the character classification (ctype) to be used in the new database. It defines how characters are classified based on their types, such as upper or lower case. In the example, it is set to **'en_US.utf8'**, indicating the English (United States) character classification with UTF-8 encoding
- **TABLESPACE [=] tablespace_name:** Specifies the tablespace where the database's data files will be stored.
- **CONNECTION LIMIT [=] connlimit:** Specifies the maximum number of concurrent connections to the database. **-1** means unlimited.

Create Object - Database

Templates in PostgreSQL are databases that serve as a template for creating new databases

```
CREATE DATABASE template_new TEMPLATE template0/ template1/MyDB
```

template0:

- **template0** is a minimal template database.
- It is a "clean" template that does not contain any user-defined objects, data, or configurations.
- It serves as a baseline template for creating other databases.
- Changes made to **template0** will not affect the behavior of new databases created from it.
- It is always available and cannot be dropped or modified.

template1:

- **template1** is a general-purpose template database.
- It is a copy of **template0** and includes basic database objects and configurations.
- Users can customize **template1** by adding additional objects, schemas, or configurations that they want to be included in all new databases created from it.
- Unlike **template0**, changes made to **template1** will be reflected in new databases created from it.
- While it can be modified, it's often recommended to keep **template1** in a pristine state to avoid unintended changes to new databases.

Create Object - Table

```
CREATE TABLE [IF NOT EXISTS] table_name (
    column_name1 data_type [column_constraint1],
    column_name2 data_type [column_constraint2],
    ...
    table_constraint1,
    table_constraint2,
    ...
);

-- Optional: Create an INDEX
CREATE [UNIQUE] INDEX index_name
ON table_name (column1 [, column2, ...]);
```

Create Object - Table Constrints

Constraint Type	Description	Sample
PRIMARY KEY	Ensures that a column or a group of columns is unique and not null.	sql CREATE TABLE example_table (id serial PRIMARY KEY, name VARCHAR(50));
UNIQUE	Ensures that values in a column or a group of columns are unique.	sql CREATE TABLE example_table (email VARCHAR(100) UNIQUE, phone VARCHAR(20));
CHECK	Enforces a condition that must be true for each row in the table.	sql CREATE TABLE example_table (age INT CHECK (age >= 18), salary DECIMAL);
FOREIGN KEY	Establishes a link between data in two tables by enforcing a referential integrity.	sql CREATE TABLE orders (order_id serial PRIMARY KEY, customer_id INT REFERENCES customers(customer_id));
DEFAULT	Sets a default value for a column.	sql CREATE TABLE example_table (status VARCHAR(20) DEFAULT 'active');
NOT NULL	Ensures that a column does not accept null values.	sql CREATE TABLE example_table (username VARCHAR(50) NOT NULL, email VARCHAR(100));
CHECK (with expression)	Provides a more complex condition using expressions.	sql CREATE TABLE employees (salary DECIMAL CHECK (salary >= 0), hire_date DATE CHECK (hire_date <= CURRENT_DATE));

Create Object – A Sample DB : Job Seeker

```
-- Define custom ENUM for user_type
CREATE TYPE user_role AS ENUM ('job_seeker', 'employer');

-- Create a sequence for generating custom IDs
CREATE SEQUENCE custom_id_seq;

-- Users Table (Common for both job seekers and employers)
CREATE TABLE users (
    user_id INT DEFAULT nextval('custom_id_seq') PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    user_type user_role NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Create Object – A Sample DB : Job Seeker

```
-- Job Seekers Table
CREATE TABLE job_seekers (
    seeker_id INT PRIMARY KEY REFERENCES users(user_id),
    resume JSON NOT NULL,
    additional_info HSTORE,  -- Storing additional information
in HStore
    FOREIGN KEY (seeker_id) REFERENCES users(user_id)
);

-- Employers Table
CREATE TABLE employers (
    employer_id INT PRIMARY KEY REFERENCES users(user_id),
    company_name VARCHAR(100),
    company_info HSTORE,  -- Storing company information in
HStore
    FOREIGN KEY (employer_id) REFERENCES users(user_id)
);
```

Create Object – A Sample DB : Job Seeker

```sql
-- Job Listings Table
CREATE TABLE job_listings (
    job_id INT DEFAULT nextval('custom_id_seq') PRIMARY
KEY,
    employer_id INT REFERENCES employers(employer_id),
    title VARCHAR(100) NOT NULL,
    description TEXT NOT NULL,
    location VARCHAR(100),
    posted_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) CHECK (status IN ('open', 'closed'))
DEFAULT 'open'
);
```

Create Object – A Sample DB : Job Seeker

```sql
-- Applications Table
CREATE TABLE applications (
    application_id INT DEFAULT nextval('custom_id_seq')
PRIMARY KEY,
    job_id INT REFERENCES job_listings(job_id),
    seeker_id INT REFERENCES job_seekers(seeker_id),
    application_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) CHECK (status IN ('pending',
'accepted', 'rejected')) DEFAULT 'pending'
);

-- Creating an index for faster searches on job listings
CREATE INDEX idx_job_listings ON job_listings(title,
location);
```

Workshop

- Please Connect into Postgres using PSQL command line utility
- Run Commands in *Section 1 Basics of DDL and Table Creation.md* one by one