Postgres setup

Table of contents

1. Task requirement: Podman.Postgres

2. Environment details:

- Os:- Ubuntu 22.04.3 LTS
- Podman- 3.4.4
- Psql (PostgreSQL) 14.9

3. System configuration:

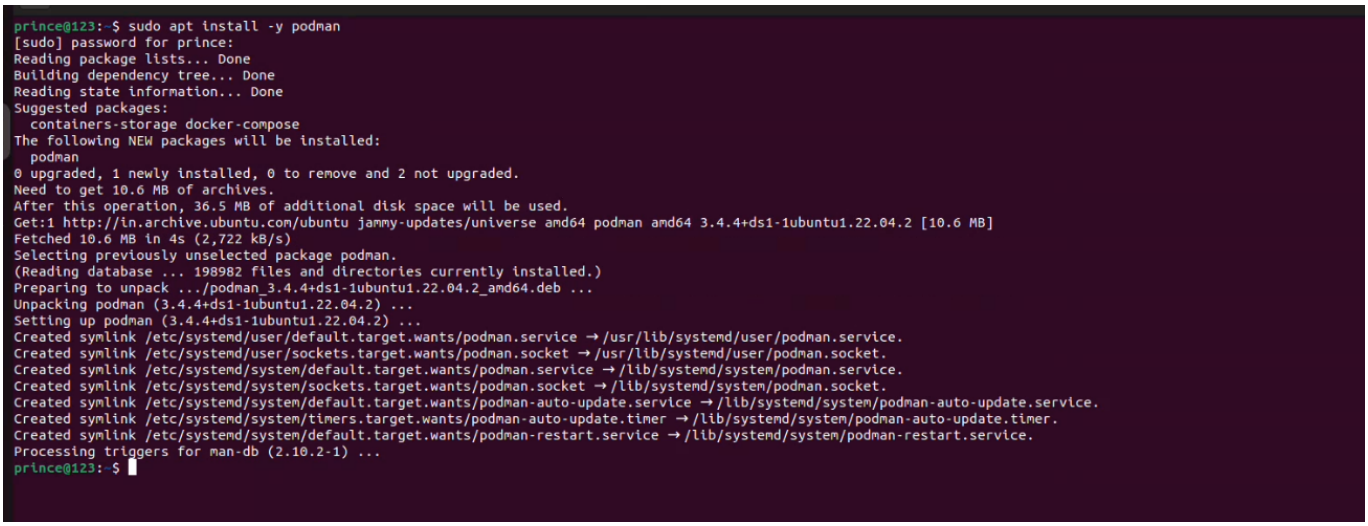
- CPU - 4
- Storage -16 GB

4. Definition of tools:

- Podman is an open-source container management tool used to create, run, and manage containers on Linux systems.
- PostgreSQL ek open-source relational database management system (RDBMS) hai jo data storage aur management ke liye istemal hota hai.

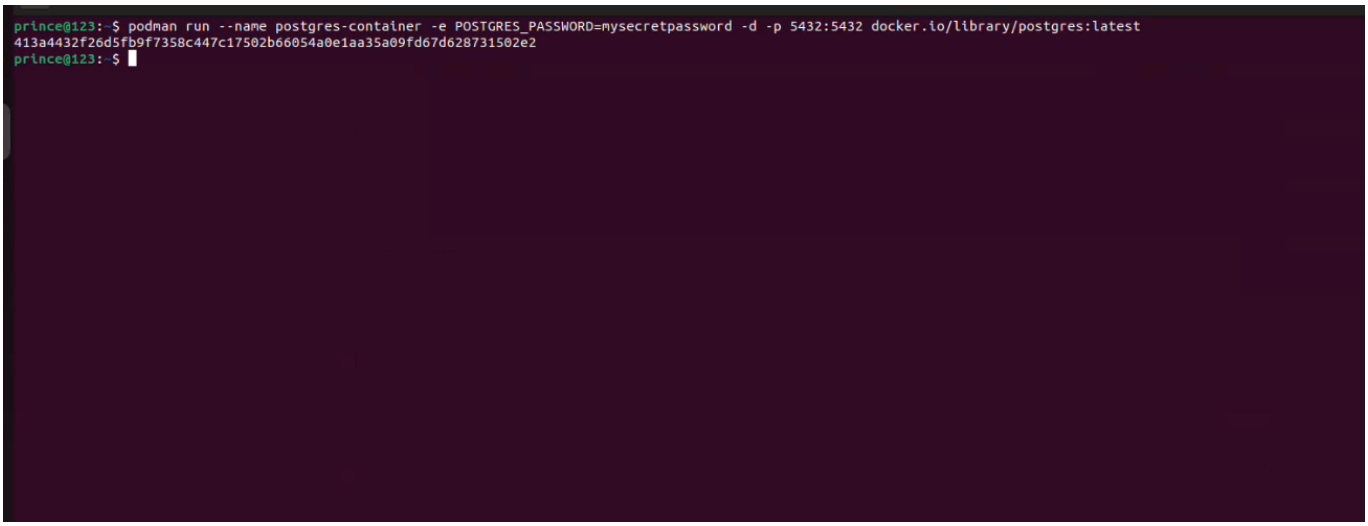
1. Create a postgres container with podman

```
sudo apt install -y podman
```




```
podman version
```

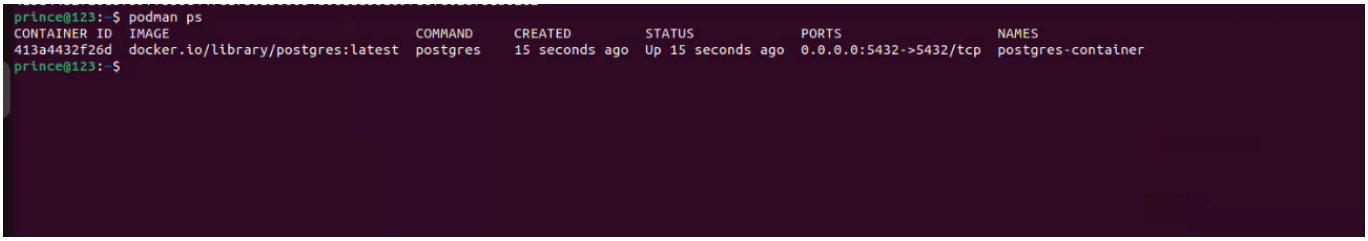

```
podman run --name postgres-container -e POSTGRES_PASSWORD=mysecretpassword
-d -p 5432:5432 docker.io/library/postgres:latest
```



- podman run: Ye command ek container ko create aur run karne ke liye use hoti hai.
- -d: Ye option container ko background mode me run karne ke liye use hota hai, matlab container run hoga aur aapka terminal available rahega.
- --name my-postgres: Is option se aap apne container ko "my-postgres" naam se identify kar sakte hain.
- -e POSTGRES_PASSWORD=mysecretpassword: Is option se aap environment variable set kar rahe hain, jiska naam "POSTGRES_PASSWORD" hai aur value "mysecretpassword" hai. Ye password PostgreSQL database access ke liye use hoga.
- -v /mydata:/var/lib/postgresql/data: Is option se aap ek volume mount kar rahe hain. /mydata host system ke path ko /var/lib/postgresql/data container ke path se map karega. Isse aap apne database data ko host system me store kar sakte hain, taki wo persist kare, aur aap container ko delete karne ke baad bhi data safe rahe.
- 'postgres:14: Ye image name hai, jise aap container se use kar rahe hain. "postgres:14" PostgreSQL ke version 14 ke official Docker image ko represent karta hai. Isse container PostgreSQL database server ke sath run karega.

Verify that the PostgreSQL container is running:

```
podman ps
```



2.create users,databases,tables,extensions on the same.

```
podman exec -it postgres-container psql -U postgres
```

(a) Create Users

```
CREATE USER noida WITH PASSWORD 'noida1';
CREATE USER delhi WITH PASSWORD 'delhi1';
CREATE USER gurugram WITH PASSWORD 'gurugram1';
```

```
postgres=# CREATE USER noida WITH PASSWORD 'noida1';
CREATE USER delhi WITH PASSWORD 'delhi1';
CREATE USER gurugram WITH PASSWORD 'gurugram1';
CREATE ROLE
CREATE ROLE
CREATE ROLE
postgres=#
```

(b) Databases

```
CREATE DATABASE my_database;
```

```
postgres=# CREATE DATABASE my_database;
CREATE DATABASE
postgres=#
```

(c) Tables

```
\c my_database;
```

```
     CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE
);
```

```
my_database=#     CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE
);

CREATE TABLE
my_database=#
```

- CREATE TABLE: This keyword tells the database to create a new table.
- my_table: This is the name of the table that is being created.
- (id SERIAL NOT NULL PRIMARY KEY, name VARCHAR(255) NOT NULL): This is the definition of the table, which includes the names and data types of the columns in the table.

Here is a more detailed explanation of each part of the statement:

- id SERIAL NOT NULL PRIMARY KEY: This column will store the unique identifier for each row in the table. The SERIAL data type means that the database will automatically generate a unique integer value for each new row that is inserted into the table. The NOT NULL constraint means that this column cannot be empty. The PRIMARY KEY constraint means that this column uniquely identifies each row in the table.
- name VARCHAR(255) NOT NULL: This column will store the name of each row in the table. The VARCHAR(255) data type means that this column can store up to 255 characters of text. The NOT NULL constraint means that this column cannot be empty.

(d) extensions

```
CREATE EXTENSION pg_trgm;
```

EXTENSION Ye SQL statement PostgreSQL database mein ek extension ko create karne ya activate karne ke liye istemal hota hai. Extension ek prakar ke additional modules ya functions hote hain jo PostgreSQL database functionality ko extend karte hain.

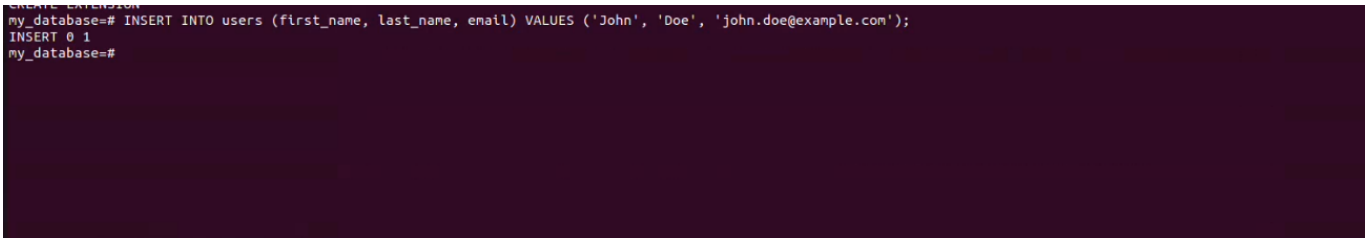
pg_trgm Ye extension PostgreSQL mein full-text search aur trigram similarity capabilities provide karta hai. Full-text search ka use karke, aap apne data mein text ko search kar sakte hain. Trigram similarity ka use karke, aap apne data mein text ke similarity ko calculate kar sakte hain. In capabilities ka use karne ke kai liye hai. For example, aap ine capabilities ka use karke ek website mein contents ko search kar sakte hain, ek database mein data ko search kar sakte hain, ya ek document collection mein documents ko search kar sakte hain.

3.Perform crud operations.

CRUD (Create, Read, Update, Delete)

(a)Create

```
INSERT INTO users (first_name, last_name, email) VALUES ('John', 'Doe',
'john.doe@example.com');
```



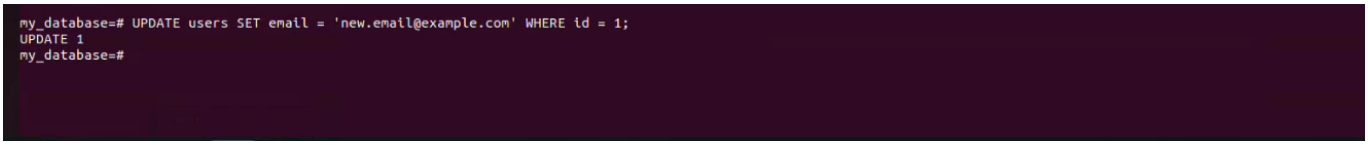
- id: A serial column that is the primary key of the table. This means that each row in the table will have a unique id value.
- name: A VARCHAR column that stores the name of the hospital.
- address: A VARCHAR column that stores the address of the hospital.
- phone: A VARCHAR column that stores the phone number of the hospital.
- NOT NULL: constraint on all of the columns means that each column must have a value. No rows will be inserted into the table if any of the columns are empty.
- VARCHAR : is a variable-length string data type in SQL. It means that it can store strings of any length, up to the maximum length specified when the column is created. It is a good choice for storing strings of variable length, such as names, addresses, and phone numbers.

(b) Read

```
SELECT * FROM users;
```

(C) Update

```
UPDATE users SET email = 'new.email@example.com' WHERE id = 1;
```



(D) Delete

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

```
my_database=# DELETE FROM users WHERE id = 1;
DELETE 1
my_database=#
```

4. Create three users with a password.

```
CREATE ROLE user1 WITH LOGIN PASSWORD 'password1';
```

```
my_database=# CREATE ROLE user1 WITH LOGIN PASSWORD 'password1';
CREATE ROLE
my_database=#
```

```
CREATE ROLE user2 WITH LOGIN PASSWORD 'password2';
```

```
my_database=# CREATE ROLE user2 WITH LOGIN PASSWORD 'password2';
CREATE ROLE
my_database=#
```

```
CREATE ROLE user3 WITH LOGIN PASSWORD 'password3';
```

```
my_database=# CREATE ROLE user3 WITH LOGIN PASSWORD 'password3';
CREATE ROLE
my_database=#
```

```
\du
```

```
my_database=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 delhi     |
 gurugram  |
 noida     |
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS
 user1     |
 user2     |
 user3     |

my_database=#
```

- show user

5. Grant select permission for user1,select,insert,delete for user2 and all for user3.

- Granting selection permission to user1:

```
GRANT SELECT ON public.users TO user1;
```

```
my_database=# GRANT SELECT ON public.users TO user1;
GRANT
my_database=#
```

- To grant select, insert, and delete permissions to user2:

```
GRANT SELECT, INSERT, DELETE ON public.users TO user2;
```

```
my_database=# GRANT SELECT, INSERT, DELETE ON public.users TO user2;
GRANT
my_database=#
```

- Give all permissions to user3

```
GRANT ALL PRIVILEGES ON public.users TO user3;
```

```
my_database=# GRANT ALL PRIVILEGES ON public.users TO user3;
GRANT
my_database=#
```

6. Understanding the table structure,finding database size, table size etc.

INSERT INTO

```
INSERT INTO users (first_name, last_name, email)
VALUES
    ('John', 'Doe', 'john.doe@example.com'),
    ('Jane', 'Smith', 'jane.smith@example.com'),
    ('Alice', 'Johnson', 'alice.johnson@example.com'),
    ('Bob', 'Brown', 'bob.brown@example.com'),
    ('Eva', 'Davis', 'eva.davis@example.com'),
    ('Michael', 'Lee', 'michael.lee@example.com'),
    ('Samantha', 'Taylor', 'samantha.taylor@example.com'),
    ('William', 'Anderson', 'william.anderson@example.com'),
    ('Olivia', 'Moore', 'olivia.moore@example.com'),
    ('Daniel', 'Wilson', 'daniel.wilson@example.com');
```

```
my_database=# INSERT INTO users (first_name, last_name, email)
VALUES
    ('John', 'Doe', 'john.doe@example.com'),
    ('Jane', 'Smith', 'jane.smith@example.com'),
    ('Alice', 'Johnson', 'alice.johnson@example.com'),
    ('Bob', 'Brown', 'bob.brown@example.com'),
    ('Eva', 'Davis', 'eva.davis@example.com'),
    ('Michael', 'Lee', 'michael.lee@example.com'),
    ('Samantha', 'Taylor', 'samantha.taylor@example.com'),
    ('William', 'Anderson', 'william.anderson@example.com'),
    ('Olivia', 'Moore', 'olivia.moore@example.com'),
    ('Daniel', 'Wilson', 'daniel.wilson@example.com');
INSERT 0 10
my_database=#
```

(a) Table structure

```
\d users
```

```
my_database=# \d users
                                    Table "public.users"
   Column   |          Type          | Collation | Nullable |              Default
------------+------------------------+-----------+----------+-----------------------------------
 id         | integer                |           | not null | nextval('users_id_seq'::regclass)
 first_name | character varying(50)  |           |          |
 last_name  | character varying(50)  |           |          |
 email      | character varying(100) |           |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)
    "users_email_key" UNIQUE CONSTRAINT, btree (email)

my_database=#
```

(b) finding database size

```
SELECT pg_size_pretty(pg_database_size(current_database()));
```

```
my_database=# SELECT pg_size_pretty(pg_database_size(current_database()));
 pg_size_pretty
----------------
 7700 kB
(1 row)

my_database=#
```

(c) table size

```
SELECT pg_size_pretty(pg_total_relation_size('users'));
```

```
my_database=# SELECT pg_size_pretty(pg_total_relation_size('users'));
 pg_size_pretty
----------------
 40 kB
(1 row)

my_database=#
```