**Automating PostgreSQL Table Counts with Jenkins**

**1. Introduction**

In this tutorial, you will learn how to:

* Set up two PostgreSQL databases using **Docker Compose**
* Create tables and insert data using **SQL scripts**
* Configure **Jenkins credentials** securely (instead of hardcoding passwords)
* Write a **Jenkins pipeline** that connects to PostgreSQL and prints table counts

**2. Prerequisites**

Before starting, make sure you have the following installed:

* **Docker** and **Docker Compose**
* **Jenkins** (with access to install plugins)
* **PostgreSQL client (psql)** on your machine
* Basic knowledge of Linux shell commands

**3. Step 1: Set Up PostgreSQL with Docker**

We will run **two PostgreSQL databases (db1 and db2)** using Docker Compose.

Create a file named docker-compose.yml with the following content:

* services:

db1:

image: postgres:14

environment:

- POSTGRES\_USER=${POSTGRES\_USER}

- POSTGRES\_PASSWORD=${POSTGRES\_PASSWORD}

- POSTGRES\_DB=${POSTGRES\_DB}

ports:

- "5432:5432"

volumes:

- db1\_data:/var/lib/postgresql/data

db2:

image: postgres:14

environment:

- POSTGRES\_USER=${POSTGRES\_USER}

- POSTGRES\_PASSWORD=${POSTGRES\_PASSWORD}

- POSTGRES\_DB=${POSTGRES\_DB}

ports:

- "5433:5432"

volumes:

- db2\_data:/var/lib/postgresql/data

volumes:

db1\_data:

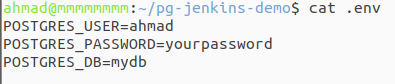
db2\_data:

Run the containers:

* docker compose up -d



📌 Note: The credentials (POSTGRES\_USER, POSTGRES\_PASSWORD, POSTGRES\_DB) are stored in a .env file or environment variables. This avoids putting them directly in the YAML file.



**4. Step 2: Create Database Schema and Seed Data**

We need two tables: schools and students.

**4.1 Create Schema (01\_schema.sql)**

DROP TABLE IF EXISTS students;

DROP TABLE IF EXISTS schools;

CREATE TABLE schools (

id SERIAL PRIMARY KEY,

school\_name TEXT NOT NULL UNIQUE,

country TEXT NOT NULL,

number\_of\_students INT NOT NULL DEFAULT 0

);

CREATE TABLE students (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

joined\_at TIMESTAMP NOT NULL DEFAULT NOW(),

grade VARCHAR(2) NOT NULL,

school\_name TEXT NOT NULL,

phone\_number TEXT NOT NULL

);

CREATE INDEX idx\_students\_school\_name ON students (school\_name);

**4.2 Insert Sample Data (02\_seed.sql)**

INSERT INTO schools (school\_name, country, number\_of\_students)

SELECT

'School ' || gs,

(ARRAY['Jordan','Saudi Arabia','UAE','Egypt','Iraq'])[1 + floor(random()\*5)::int],

100 + floor(random()\*1901)::int

FROM generate\_series(1, 200) AS gs;

INSERT INTO students (name, joined\_at, grade, school\_name, phone\_number)

SELECT

'Student ' || gs,

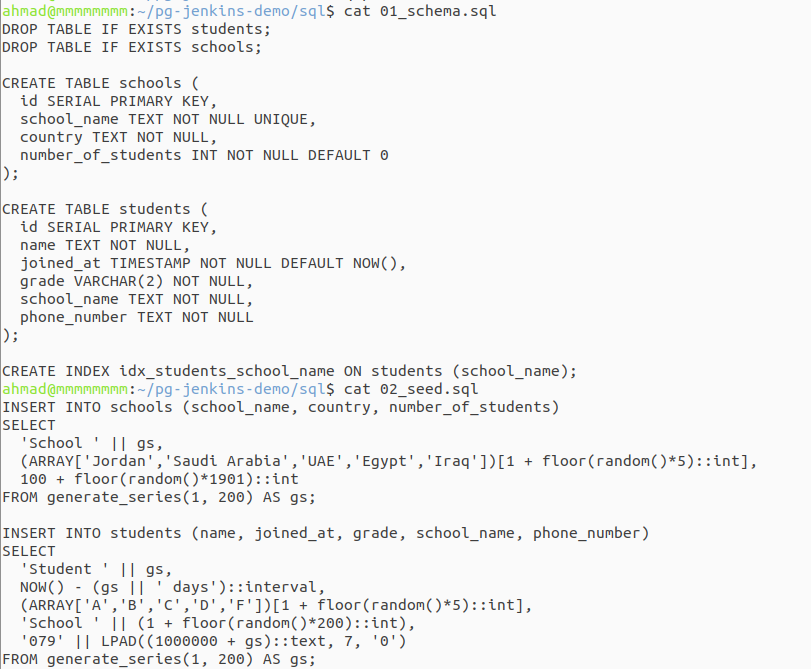
NOW() - (gs || ' days')::interval,

(ARRAY['A','B','C','D','F'])[1 + floor(random()\*5)::int],

'School ' || (1 + floor(random()\*200)::int),

'079' || LPAD((1000000 + gs)::text, 7, '0')

FROM generate\_series(1, 200) AS gs;



Load schema and data into PostgreSQL:

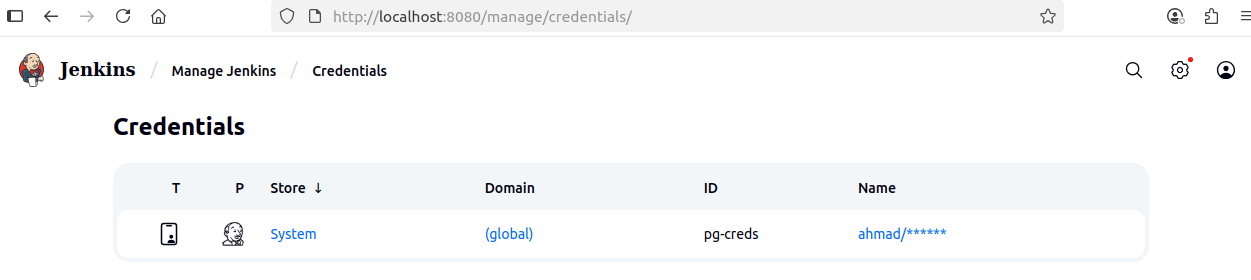
psql -h localhost -p 5432 -U ahmad -d mydb -f 01\_schema.sql

psql -h localhost -p 5432 -U ahmad -d mydb -f 02\_seed.sql

**5. Step 3: Configure Jenkins Credentials**

Instead of hardcoding the username and password, we store them securely in Jenkins.

1. Go to **Jenkins Dashboard → Manage Jenkins → Credentials**
2. Add new credentials:
   * **Kind**: Username with password
   * **ID**: pg-creds
   * **Username**: ahmad
   * **Password**: ahmad
3. Save



📌 Jenkins will now inject these credentials into the pipeline securely.

**6. Step 4: Create Jenkins Pipeline**

Create a new **Pipeline Job** in Jenkins and use the following pipeline script:

pipeline {

agent any

environment {

DB\_NAME = "mydb"

DB\_HOST = "localhost"

DB\_PORT = "5432"

}

stages {

stage('Print Table Counts') {

steps {

withCredentials([usernamePassword(credentialsId: 'pg-creds', usernameVariable: 'DB\_USER', passwordVariable: 'DB\_PASS')]) {

sh '''

echo "Schools count:"

PGPASSWORD=$DB\_PASS psql -h $DB\_HOST -p $DB\_PORT -U $DB\_USER -d $DB\_NAME -t -c "SELECT COUNT(\*) FROM schools;"

echo "Students count:"

PGPASSWORD=$DB\_PASS psql -h $DB\_HOST -p $DB\_PORT -U $DB\_USER -d $DB\_NAME -t -c "SELECT COUNT(\*) FROM students;"

'''

}

}

}

}

}



**How It Works:**

* **withCredentials** → fetches credentials from Jenkins securely
* **PGPASSWORD=$DB\_PASS** → sets password temporarily
* **psql -t -c "SELECT COUNT(\*)"** → runs SQL query and prints result

**7. Step 5: Run the Pipeline & Verify**

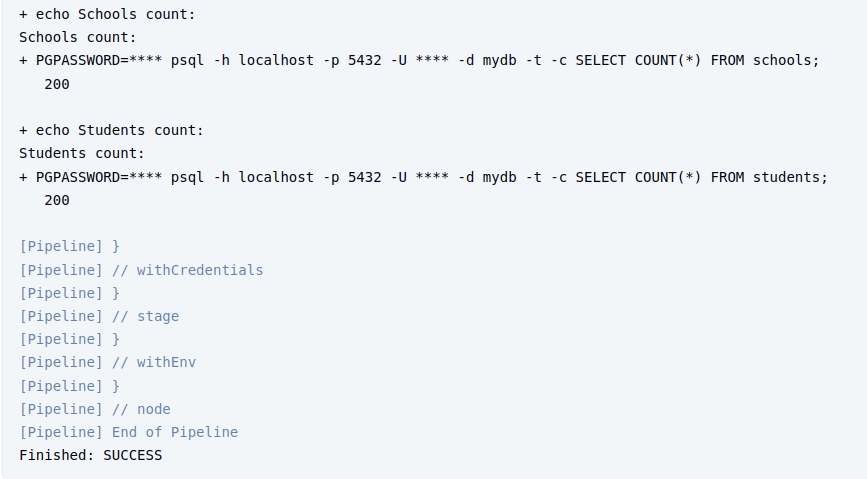
1. Build the pipeline job in Jenkins.
2. Expected output in console:

Schools count:

200

Students count:

200



✅ If you see the counts, your pipeline is working correctly!

**8. Best Practices & Security Notes**

* **Never hardcode passwords** in scripts or pipelines.
* Use **Jenkins credentials** to protect sensitive information.
* Keep different databases in **separate containers** for isolation.
* Use **indexes** (like idx\_students\_school\_name) for faster queries.

**9. Wrap-Up**

🎉 Congratulations! You just:

* Deployed PostgreSQL with Docker
* Created and seeded database tables
* Configured Jenkins credentials securely
* Wrote a pipeline to query and display table counts

This workflow is a **foundation for database automation** in real-world projects.