Installing and Using PostgreSQL

Learning Objectives

After completing this lab session you should be able to

- Install and configure PostgreSQL database
- Setup the database for the sample application
- Configure roles and users for the sample application
- Connect PostgresSQL database from Go code and check the user and role configuration

Installation and Configuration of PostgreSQL

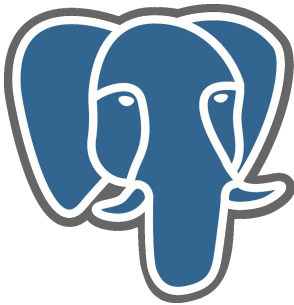
Install PostgreSQL from its website

https://www.postgresql.org/download/

By default Postgres creates a **postgres** user and that's the only user who can connect to the server

For convenience you can create another Postgres account with your username but first, you need to log in to the Postgres account

Optionally you can install **PgAdmin4** for visual sql editor



Installation and Configuration of PostgreSQL

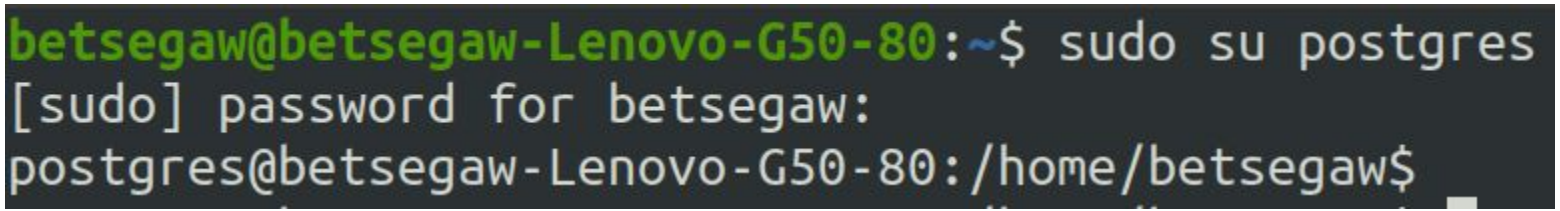
Official PostgreSQL tutorial is available at

https://www.postgresql.org/docs/12/tutorial.html

Connecting to and Using PostgreSQL

Login to PostgreSQL on Linux

```
sudo su postgres
```



PostgreSQL Interactive Terminal (Shell)

Type **`psql`** command to run the interactive terminal

By default the **`psql`** command connects you to a default database called postgres

```
postgres@betsegaw-Lenovo-G50-80:/home/betsegaw$ psql
psql (12.1 (Ubuntu 12.1-1.pgdg19.10+1))
Type "help" for help.

postgres=#
```

You can now use any PostgreSQL command

Change password of postgres users

You can change the password of the default user - **postgres**

```
ALTER USER postgres WITH PASSWORD 'P@$$w0rDd';
```

Listing existing users

Type **`\du`** to list all existing PostgreSQL users

```
postgres=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}

postgres=#
```

Create database

Run **`CREATE DATABASE restaurantdb;`** command to create a database named **restaurantdb**

```
postgres=# CREATE DATABASE restaurantdb;
CREATE DATABASE
```

You can list databases using **`\l`** command

```
postgres=# \l
                                  List of databases
    Name     |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------+----------+----------+-------------+-------------+-----------------------
 postgres    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 restaurantdb | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
```

Connect to a database

To connect to a particular database in one of the following commands

```
psql -d database -U  user -W
```

Example: `psql -d restaurantdb -U postgres -W`

If you want to connect to a database that resides on another host

```
psql -h host -d database -U user -W
```

If you want to use SSL mode for the connection

```
psql -U user -h host "dbname=db sslmode=require"
```

Switch database

Once you are connected to a database, you can switch the connection to a new database under a user specified by user

```
\c dbname username
```

Example: `\c template1 postgres`

Drop database

Run the `dropdb <dbname>;` command to drop a database

OR run as follows from `psql` terminal

```
CREATE DATABASE <dbname>;
```

Common Commands

`\?`: List all the commands

`\l`: List databases

`\conninfo`: Display information about current connection

`\c <Database Name>`: Connect to a database

`\d <Table Name>`: Show table definition including triggers

Common Commands

`\dt`: List all tables

`\d+ <Table Name>`: More detailed table definition including description and physical disk size

`\l`: List databases

`\x`: Pretty-format query results instead of the not-so-useful ASCII tables

Common Commands

`\du`: List users

`\du <username>`: List a username if present

`\dn`: List available schema

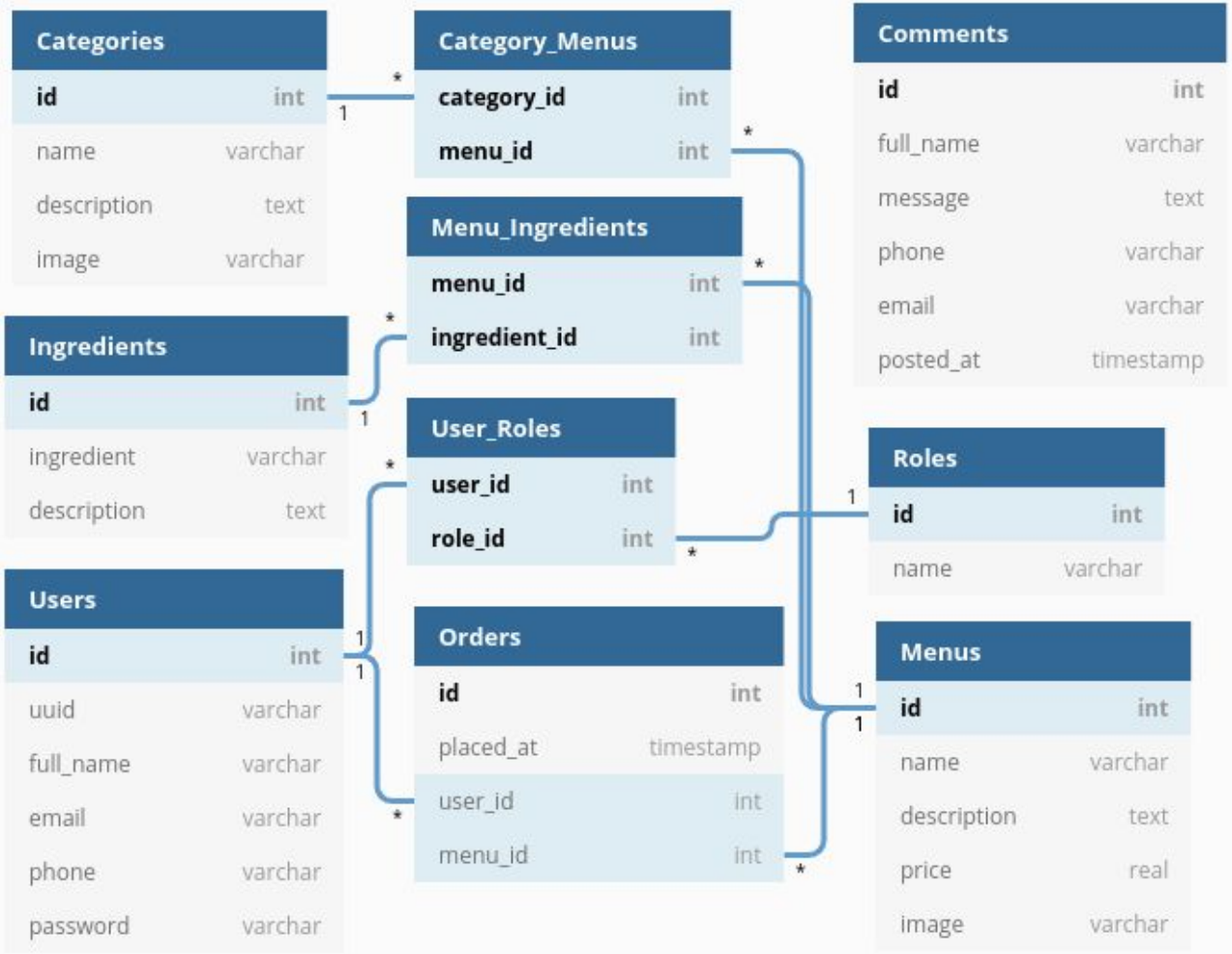
`\e`: Edit command in your own editor

`\q`: Quit/Exit

DB Structure

Sample Application table relationships

You can go to **https://dbdiagram.io/** to create similar diagrams



Database Table Definitions of the Sample App

```
1   create table categories (
2       id  integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
3       name varchar(255) NOT NULL,
4       description text,
5       image varchar(255)
6   );
7
8   create table menus (
9       id integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
10      name varchar(255) NOT NULL,
11      price numeric NOT NULL DEFAULT 0,
12      description text,
13      image varchar(255)
14  );
15
```

Database Table Definitions of the Sample App

```
16  create table menu_category (
17      menu_id integer REFERENCES menus(id) ON UPDATE CASCADE ON DELETE CASCADE,
18      category_id integer REFERENCES categories(id) ON UPDATE CASCADE,
19      CONSTRAINT menu_category_pky PRIMARY KEY (menu_id, category_id)
20  );
21
22  create table ingredients (
23      id integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
24      name varchar(255) NOT NULL,
25      description varchar(255)
26  );
```

Database Table Definitions of the Sample App

```
28  create table menu_ingredients (
29      menu_id integer REFERENCES menus(id) ON UPDATE CASCADE ON DELETE CASCADE,
30      ingredient_id integer REFERENCES ingredients(id) ON UPDATE CASCADE,
31      CONSTRAINT menu_ingredient_pkey PRIMARY KEY (menu_id, ingredient_id)
32  );
33
34  create table users (
35      id  integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
36      uuid varchar(64) NOT NULL UNIQUE,
37      full_name varchar(255),
38      email varchar(255) NOT NULL UNIQUE,
39      phone varchar(255) NOT NULL UNIQUE,
40      password varchar(255) NOT NULL
41  );
42
```

Database Table Definitions of the Sample App

```
43  create table roles (
44      id integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
45      name varchar(255) NOT NULL UNIQUE
46  );
47
48  create table user_roles (
49      user_id integer REFERENCES users(id) ON UPDATE CASCADE ON DELETE CASCADE,
50      role_id integer REFERENCES roles(id) ON UPDATE CASCADE,
51      CONSTRAINT user_role_pkey PRIMARY KEY (user_id, role_id)
52  );
53
```

Database Table Definitions of the Sample App

```
54  create table orders (
55      id integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
56      placed_at timestamp NOT NULL,
57      user_id integer REFERENCES users(id) ON UPDATE CASCADE ON DELETE CASCADE,
58      menu_id integer REFERENCES menus(id) ON UPDATE CASCADE
59  );
60
61  create table comments (
62      id integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
63      full_name varchar(255),
64      message text NOT NULL,
65      phone varchar(255),
66      email varchar(255),
67      posted_at timestamp NOT NULL
68  );
```

Database Table Definitions of the Sample App

You can put all the table definitions in a single `.sql` file and run the following command to execute it

```
psql -f restaurant.sql -d restaurantdb
```

Make sure to create the database "`restaurantdb`" before executing the above command

Managing Users and Roles in PostgreSQL

Below are good practices for fine-grained access control

- Use the master user to create roles per application or use case, like `readonly` and `readwrite`
- Add permissions to allow these roles to access various database objects. For example, the `readonly` role can only run `SELECT` queries
- Grant the roles the least possible permissions required for the functionality

Managing Users and Roles in PostgreSQL

Below are good practices for fine-grained access control

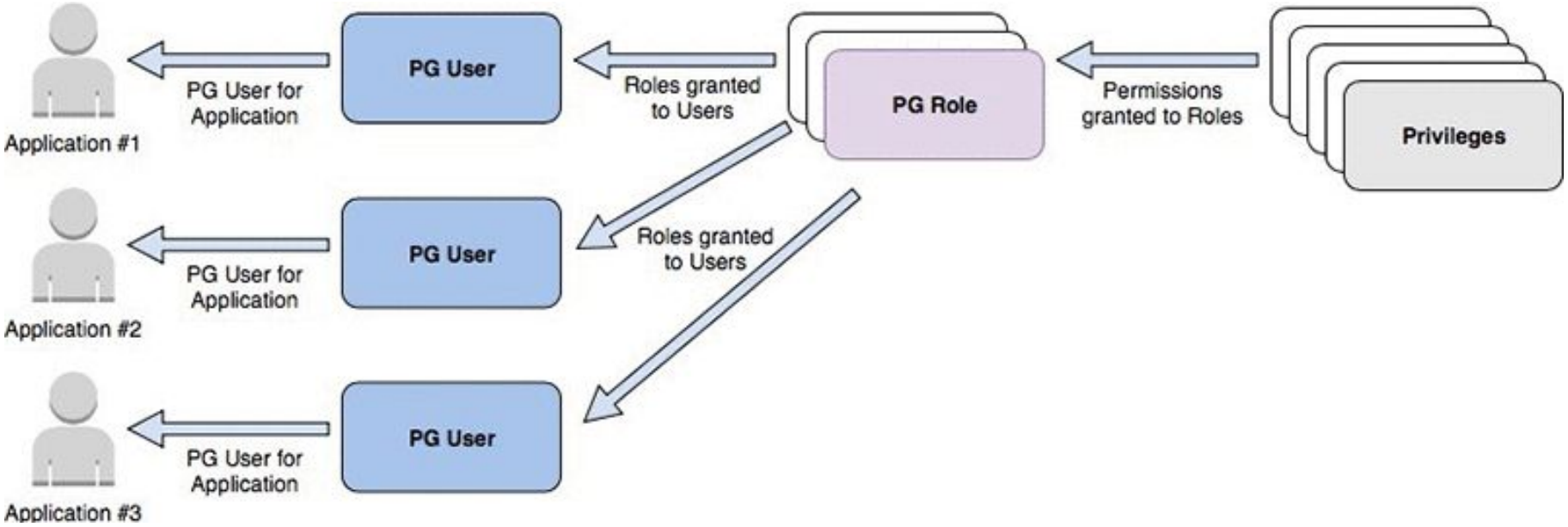
Create new users for each application or distinct functionality, like **`app_user`** and **`reporting_user`**

Assign the applicable roles to these users to quickly grant them the same permissions as the role. For example, grant the **`readwrite`** role to **`app_user`** and grant the **`readonly`** role to **`reporting_user`**

At any time, you can remove the role from the user in order to revoke the permissions

Managing Users and Roles in PostgreSQL

Below is summary of good practices for fine-grained access control



Managing Users and Roles in PostgreSQL

Users, groups, and roles

Users, groups, and roles are the same thing in PostgreSQL, with the only difference being that users have permission to log in by default

The `CREATE USER` and `CREATE GROUP` statements are actually aliases for the `CREATE ROLE` statement

Managing Users and Roles in PostgreSQL

Schema

In PostgreSQL, a **schema** is a namespace that contains named database objects such as tables, views, indexes, data types, functions, and operators

To access an object of a schema, you qualify its name with the schema name as a prefix

```
Schema_name.object_name
```

Example: `\dt public.categories`

Managing Users and Roles in PostgreSQL

Public schema and public role

When a new database is created, PostgreSQL by default creates a **schema** named `public` and grants access on this schema to a backend **role** named `public`

All new users and roles are by default granted this `public` **role**, and therefore can create objects in the public schema

When a user tries to create a new table without specifying the schema name, the table gets created in the `public` schema

Managing Users and Roles in PostgreSQL

Public schema and public role

By default, all users have access to create objects in the public schema

This becomes a problem if you are trying to create a read-only user. Even if you restrict all privileges, the permissions inherited via the public role allow the user to create objects in the public schema

To fix this, you should revoke the default create permission on the public schema from the public role

```
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
```

Managing Users and Roles in PostgreSQL

Public schema and public role

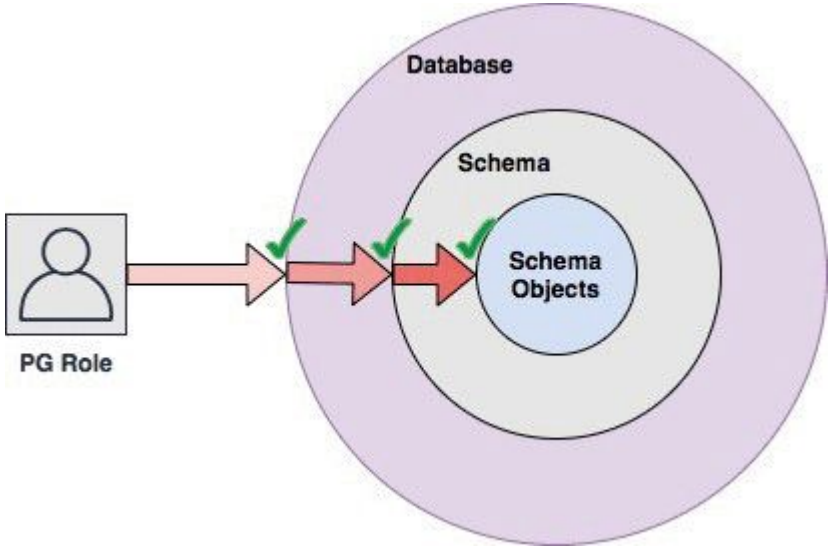
The following statement revokes the public role's ability to connect to the database

```
REVOKE ALL ON DATABASE restaurantdb FROM PUBLIC;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Permissions must be granted at the database, schema, and schema object level



Managing Users and Roles in PostgreSQL

Creating database roles

Read-Only role

The first step is to create a new role named **readonly** using the following SQL statement

```
CREATE ROLE readonly;
```

This is a base role with no permissions and no password. It cannot be used to log in to the database

Managing Users and Roles in PostgreSQL

Creating database roles

Read-Only role

Grant this role permission to connect to your target database named "**restaurantdb**"

```
GRANT CONNECT ON DATABASE restaurantdb TO readonly;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read-Only role

The next step is to grant this role usage access to your schema

```
GRANT USAGE ON SCHEMA public TO readonly;
```

The next step is to grant the **readonly** role access to run select on the required tables

```
GRANT SELECT ON TABLE categories, menus TO readonly;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read-Only role

If the requirement is to grant access on all the tables and views in the schema, then you can use the following SQL

```
GRANT SELECT ON ALL TABLES IN SCHEMA public TO
readonly;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read-Only role

Note that any new tables that get added in the future will not be accessible by the **readonly** user. To help ensure that new tables and views are also accessible, run the following statement to grant permissions automatically

```
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT
ON TABLES TO readonly;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read/Write role

Creating a role

```
CREATE ROLE readwrite;
```

Grant this role permission to connect to your target database

```
GRANT CONNECT ON DATABASE restaurantdb TO readwrite;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read/Write role

Grant schema usage privilege

```
GRANT USAGE ON SCHEMA public TO readwrite;
```

If you want to allow this role to create new objects like tables in this schema, then use the following SQL instead of the one preceding

```
GRANT USAGE, CREATE ON SCHEMA public TO readwrite;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read/Write role

Grant access to the tables. The grant can be on individual tables or on all tables in the schema

```
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE
comments, orders TO readwrite;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN
SCHEMA public TO readwrite;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read/Write role

To automatically grant permissions on tables and views added in the future

```
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT
SELECT, INSERT, UPDATE, DELETE ON TABLES TO readwrite;
```

Managing Users and Roles in PostgreSQL

Creating database roles

Read/Write role

For read/write roles, there is normally a requirement to use sequences also

```
GRANT USAGE ON SEQUENCE myseq1, myseq2 TO readwrite;
```

You can also grant permission to all sequences

```
GRANT USAGE ON ALL SEQUENCES IN SCHEMA public TO readwrite;
```

Managing Users and Roles in PostgreSQL

Creating database users

Create the user and grant it one of the existing roles

```
CREATE USER app_user WITH PASSWORD 'P@$$w0RrdD';

GRANT readwrite TO app_user;
```

Users and roles for the sample application

CRUD privilege per role and pre user

Users	Roles	Permissions	On Tables
app_user	user_role	Read	categories, menus, ingredients, menu_categories, menu_ingredients, roles, users, user_roles
		Create, Read, Update, Delete (CRUD)	comments, orders
app_admin	admin_role	Create, Read, Update, Delete (CRUD)	All Tables

Users and roles for the sample application

Steps

Remove access to creating objects in the **public schema**

Remove the **public role**'s ability to connect to the database

Create the `user_role` and `admin_role` roles

Grant these roles permission to connect to the `restaurantdb` database

Grant schema usage privilege to the roles

Users and roles for the sample application

Steps

Grant the roles access to the tables as per the requirement shown in the previous table

Allow to automatically grant permissions on objects such as tables added in the future

Grant permissions to sequences

Create the `app_user` and `app_admin` users and assign them the corresponding roles

Users and roles for the sample application

Steps

Remove access to creating objects in the **public schema**

```
REVOKE CREATE ON SCHEMA public FROM public;
```

Remove the `public` **role**'s ability to connect to the database

```
REVOKE ALL ON DATABASE restaurantdb FROM public;
```

Users and roles for the sample application

Steps

Create the **user** and **admin** roles

```
CREATE ROLE user_role;

CREATE ROLE admin_role;
```

Grant these roles permission to **connect** to the **restaurantdb** database

```
GRANT CONNECT ON DATABASE restaurantdb TO user_role;

GRANT CONNECT ON DATABASE restaurantdb TO admin_role;
```

Users and roles for the sample application

Steps

Grant schema usage privilege to the roles

```
GRANT USAGE ON SCHEMA public TO user_role;

GRANT USAGE ON SCHEMA public TO admin_role;
```

Users and roles for the sample application

Steps

Grant the roles access to the tables as per the requirement shown in the previous table

```
GRANT SELECT ON TABLE categories, menus, ingredients,
menu_categories, menu_ingredients, roles, users,
user_roles TO user_role;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE
comments, orders TO user_role;
```

Users and roles for the sample application

Steps

Grant the roles access to the tables as per the requirement shown in the previous table

```
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN
SCHEMA public TO admin_role;
```

Users and roles for the sample application

Steps

Allow to automatically grant permissions on objects such as tables added in the future

```
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT
ON TABLES TO user_role;

ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT
SELECT, INSERT, UPDATE, DELETE ON TABLES TO
admin_role;
```

Users and roles for the sample application

Steps

Grant permissions to sequences

```
GRANT USAGE ON ALL SEQUENCES IN SCHEMA public TO
user_role;
```

```
GRANT USAGE ON ALL SEQUENCES IN SCHEMA public TO
admin_role;
```

Users and roles for the sample application

Steps

Create the app_user and app_admin users and assign them the corresponding roles

```
CREATE USER app_user WITH PASSWORD 'P@$$w0rdD1';

GRANT user_role TO app_user;

CREATE USER app_admin WITH PASSWORD 'P@$$w0rdD2';

GRANT admin_role TO app_admin;
```

Connecting to PostgreSQL from Go code

To connect to PostgreSQL database you first need to download postgres driver. Type the following command inside your project directory

```
go get github.com/lib/pq
```

For this to work correctly, you need to have properly configured **`GOPATH`** environment variable (Refer to the first lab to check how to do)

Connecting to PostgreSQL from Go code

In your `main.go` file write the following import statements

Then declare the variable `db` that will be used to store the connection object

```
1   package main
2
3   import (
4       "database/sql"
5       "fmt"
6
7       _ "github.com/lib/pq"
8   )
9
10  var db *sql.DB
11
12 > func main() {…
43  }
```

Connecting to PostgreSQL from Go code

To connect to the database you can use either of the following approaches

```
db, err = sql.Open("postgres", "user=app_user
dbname=restaurantdb password='P@$$w0rdD1'
sslmode=disable"
```

OR

```
db, err := sql.Open("postgres",
"postgres://app_admin:P@$$w0rdD2@localhost/restaurantdb?s
slmode=disable")
```

Connecting to PostgreSQL from Go code

```
12  func main() {
13
14      db, err := sql.Open("postgres",
15          "postgres://app_admin:P@$$w0rdD2@localhost/restaurantdb?sslmode=disable")
16
17      if err != nil {
18          panic(err)
19      }
20
21      defer db.Close()
22
23      if err = db.Ping(); err != nil {
24          panic(err)
25      }
26
27      fmt.Println("Database connection established successfully")
28  }
```

Your main function can look like the code shown here

If you see the output of the last print statement, then you have successful connection

Check if you can also connect to the database using the `app_user` account

Run SQL statements from Go code

At the `psql` terminal run the following `SQL` code to insert data to the roles table

```
INSERT INTO roles (name) values ('manager');
```

Check if the data has been inserted by running the `SELECT` query at the psql terminal

```
SELECT * FROM roles;
```

Run SQL statements from Go code

Let's now run the `SELECT` query from Go code

Add the following code to your `main.go` file next to the `fmt.Println` statement

```
29      role := "Default"
30
31      err = db.QueryRow("SELECT name FROM roles WHERE id = 1").Scan(&role)
32      if err != nil {
33          panic(err)
34      }
35
36      fmt.Println(role)
```

Run your code and you should see the string `"manager"` printed

Run SQL statements from Go code

Let's now check with **INSERT** query from Go code

Add the following code to your **main.go**

If you run this code with the **app_user** privilege, it will panic as the **app_user** account do not have **CREATE** permission on the **roles** table (check it)

```
38      _, err = db.Exec("INSERT INTO roles (name) VALUES ('admin')")
39
40      if err != nil {
41          panic(err)
42      }
```

References

http://www.postgresqltutorial.com/

https://aws.amazon.com/blogs/database/managing-postgresql-users-and-roles/

https://www.postgresql.org/docs/current/