SQL DCL Task 1

Task 1. Figure out what security precautions are already used in your 'dvd\_rental' database. Prepare description

**The first security measure to check is the User Access Control:**

There is a script in PostgreSQL which if run, displays all the rolnames that are existing within a database. The proper query is this:  
**SELECT** ***rolname*** **FROM** ***pg\_roles***;  
In our case in the dvdrental the following roles are existing.

* pg\_database\_owner
* pg\_read\_all\_data
* pg\_write\_all\_data
* pg\_monitor
* pg\_read\_all\_settings
* pg\_read\_all\_stats
* pg\_stat\_scan\_tables
* pg\_read\_server\_files
* pg\_write\_server\_files
* pg\_execute\_server\_program
* pg\_signal\_backend
* pg\_checkpoint
* pg\_use\_reserved\_connections
* pg\_create\_subscription
* postgres

It is true that all of these roles exists, however they did not have any accesses granted. According to the sources each have (only the role name differs):

But from the names of these roles, if there would be a proper access granting, the structure of accesses can be useful.

**CREATE** **ROLE** pg\_execute\_server\_program **WITH**

NOSUPERUSER

NOCREATEDB

NOCREATEROLE

**INHERIT**

NOLOGIN

NOREPLICATION

NOBYPASSRLS

**CONNECTION** **LIMIT** -1;

The meaning of this code is the following:  
NOSUPERUSER: The role does not have superuser privileges. Superusers can bypass all access permission checks in the database.

NOCREATEDB: The role cannot create new databases.

NOCREATEROLE: The role cannot create new roles.

INHERIT: The role will inherit permissions of roles it is a member of.

NOLOGIN: The role cannot be used to log in to the database. It can be used to group and manage permissions for other roles that do have login privileges.

NOREPLICATION: The role cannot initiate streaming replication or put the system in or out of backup mode.

NOBYPASSRLS: The role must obey row-level security policies. If BYPASSRLS is specified, the role bypasses any row-level security policies.

CONNECTION LIMIT -1: There is no limit on the number of concurrent connections this role can make.

Also I have gone through all the tables with the following script and each time the result was NULL, so basically there is no privileges granted to these tables:  
**SELECT** ***relacl*** **FROM** ***pg\_class*** **WHERE** ***relname*** = **'inventory'**;

But, there is a script:

**SELECT** grantee,

table\_schema,

table\_name,

privilege\_type

**FROM** information\_schema.table\_privileges

**ORDER** **BY** grantee, table\_schema, table\_name;

By using this, it gives the output of all privileges for each table for each grantee that has privileges. And from that I could recognize that there exist 3 types, “PUBLIC” “Postgres” and “pg\_read\_all\_stats” this role have access to the followings:

A blue and black text box

Description automatically generated

SSL:

Data encryption, it can be turned on or off. In this database this is **OFF.**

**show** ssl;

**Conclusion**

In conclusion, this database is having a high risk in terms of data loss and security, because there are no proper division of table accesses to the users that may use them.  
In my way of thinking, I would certainly create roles before giving access to these database to anyone (as I learned from the current material..).