### Evidencias

El alumno deberá entregar: **CRISTÓBAL SUÁREZ ABAD**

- Sentencias SQL completas y resultado obtenido
- Evidencia con consultas de verificación

# Actividad 1 — Gestión de Usuarios y Roles (2 puntos)

1. Crea una base de datos llamada **sistema_ventas** y conéctate a ella.

CREATE DATABASE sistema_ventas;

```
postgres=# CREATE DATABASE sistema_ventas;
CREATE DATABASE
postgres=#
```

Nos posicionamos en ella para seguir con la tarea:

\c sistema_ventas;

```
postgres=# \c sistema_ventas;
You are now connected to database "sistema_ventas" as user "postgres".
sistema_ventas=#
```

2. Crea los siguientes usuarios:

| Usuario | Requisitos |
|---|---|
| **supervisor_ventas** | Contraseña `Sup3r#2025`, puede crear roles, no puede crear BD, hereda privilegios |
| **asistente_ventas** | Contraseña `Asist$2025`, límite de 2 sesiones, no puede crear roles ni BD |
| **inspector** | Contraseña `Inspect#2025`, no hereda privilegios, solo 1 sesión |

**supervisor_ventas:**

```
CREATE USER
supervisor_ventas
WITH PASSWORD
'Sup3r#2025'
    CREATEROLE
    NOCREATEDB
    INHERIT;
```

```
sistema_ventas=# CREATE USER supervisor_ventas
WITH PASSWORD 'Sup3r#2025'
     CREATEROLE
     NOCREATEDB
     INHERIT;
CREATE ROLE
sistema_ventas=#
```

**asistente_ventas:**

CREATE USER asistente_ventas

WITH PASSWORD 'Asist$2025'

CONNECTION LIMIT 2

NOCREATEDB

NOCREATEROLE;

```
CREATE ROLE
sistema_ventas=# CREATE USER asistente_ventas
WITH PASSWORD 'Asist$2025'
         CONNECTION LIMIT 2
     NOCREATEDB
     NOCREATEROLE;
CREATE ROLE
sistema_ventas=#
```

inspector:

CREATE USER inspector

WITH PASSWORD 'Inspect#2025'

NOINHERIT

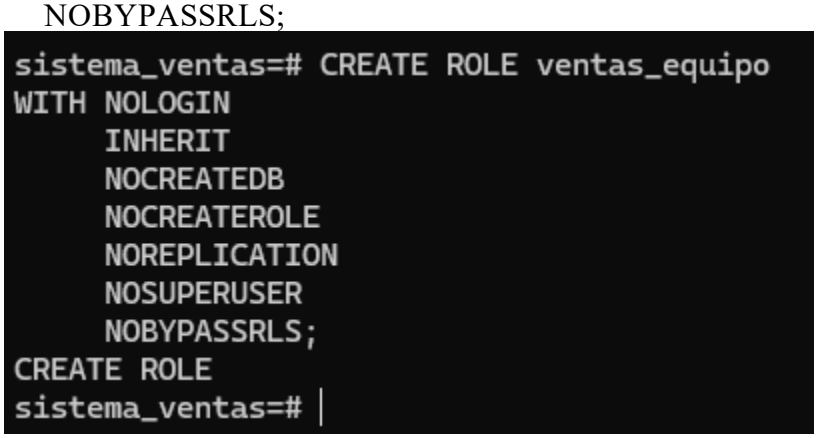
CONNECTION LIMIT 1;

```
sistema_ventas=# CREATE USER inspector
WITH PASSWORD 'Inspect#2025'
         NOINHERIT
         CONNECTION LIMIT 1;
CREATE ROLE
sistema_ventas=#
```

3. Crea un rol **ventas_equipo** con los siguientes requisitos:

- No inicia sesión
- Hereda permisos
- No crea BD ni roles
- No es superusuario
- No tiene permisos de replicación
- No puede omitir políticas

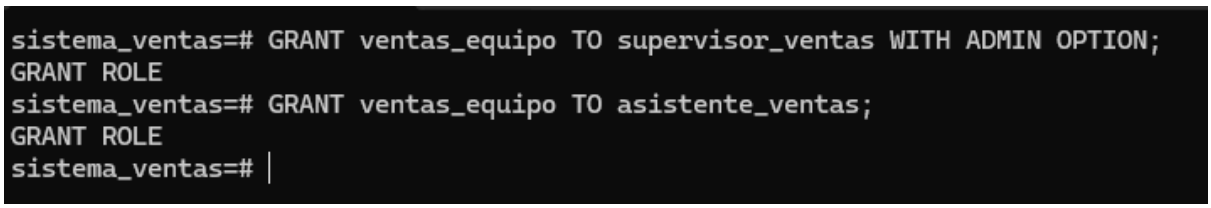
```
CREATE ROLE ventas_equipo
WITH NOLOGIN
   INHERIT
   NOCREATEDB
   NOCREATEROLE
   NOREPLICATION
   NOSUPERUSER
   NOBYPASSRLS;
```




4. Asocia **supervisor_ventas** y **asistente_ventas** al rol **ventas_equipo**.
   El supervisor debe poder administrar permisos del rol; el asistente no.

```
GRANT ventas_equipo TO supervisor_ventas WITH ADMIN OPTION;

GRANT ventas_equipo TO asistente_ventas;
```



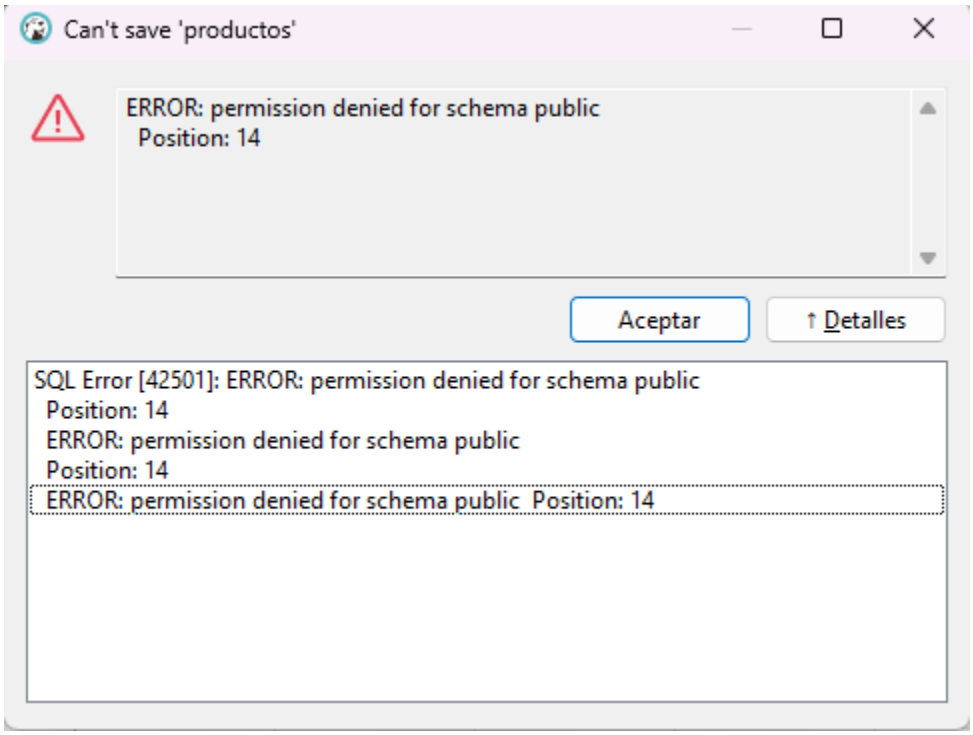
5. Demuestra la pertenencia a los roles con consultas.

6. Con el usuario ***asistente*_ventas**, crea una tabla llamada **productos** y luego intenta eliminar el usuario desde otro perfil.

Desde DBeaver nos creamos una conexión con las credenciales del usuario.

```
CREATE TABLE public.productos (
        id varchar NOT NULL,
        nombre varchar NULL,
        CONSTRAINT productos_pk PRIMARY KEY (id)
);
```

De principio no podemos crear la tabla, porque el usuario no tiene permisos para crear tablas en el esquema público.

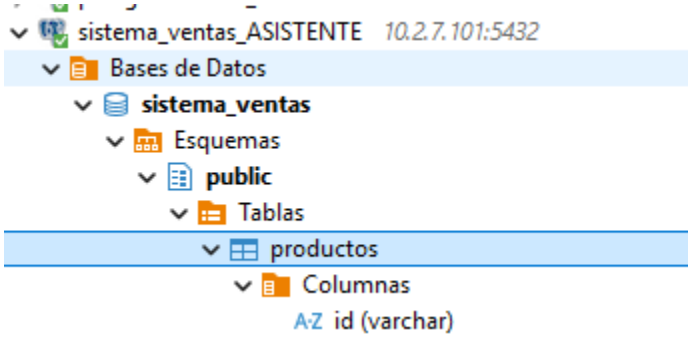



Para solucionarlo.

**GRANT CREATE ON SCHEMA public TO asistente_ventas;**

```
sistema_ventas=# GRANT CREATE ON SCHEMA public TO asistente_ventas;
GRANT
sistema_ventas=#
```

Ahora si Podemos:




```
sistema_ventas=# \dt
               List of relations
 Schema |    Name    | Type  |      Owner
--------+------------+-------+------------------
 public | productos | table | asistente_ventas
(1 row)

sistema_ventas=#
```

Intentamos eliminarlo:

DROP ROLE asistente_ventas;

- ¿Qué sucede y por qué?

No podemos: Algunos elementos (objetos) dependen de él. Es dueño de algunas tablas

```
sistema_ventas=# DROP ROLE asistente_ventas;
ERROR:  role "asistente_ventas" cannot be dropped because some objects depend on it
DETALLE:  privileges for schema public
owner of table productos
sistema_ventas=#
```

- ¿Qué pasos serían necesarios para eliminarlo correctamente?

Quitarle de todo:

REVOKE ALL ON ALL TABLES IN SCHEMA public FROM asistente_ventas;

REVOKE ALL ON ALL SEQUENCES IN SCHEMA public FROM asistente_ventas;

REVOKE ALL ON ALL FUNCTIONS IN SCHEMA public FROM asistente_ventas;

REVOKE ALL ON SCHEMA public FROM asistente_ventas;

```
sistema_ventas=# REVOKE ALL ON ALL TABLES IN SCHEMA public FROM asistente_ventas;
REVOKE
sistema_ventas=# REVOKE ALL ON ALL SEQUENCES IN SCHEMA public FROM asistente_ventas;
REVOKE
sistema_ventas=# REVOKE ALL ON ALL FUNCTIONS IN SCHEMA public FROM asistente_ventas;
REVOKE
sistema_ventas=# REVOKE ALL ON SCHEMA public FROM asistente_ventas;
REVOKE
sistema_ventas=#
```

Nos puede seguir danto error porque es dueño de una tabla

```
sistema_ventas=# DROP ROLE asistente_ventas;
ERROR:  role "asistente_ventas" cannot be dropped because some objects depend on it
DETALLE:  owner of table productos
```

Para ello cambiamos el propietario de la tabla:

ALTER TABLE public.productos OWNER TO supervisor_ventas;

Ahora si nos deja borrar al usuario.

```
sistema_ventas=# ALTER TABLE public.productos OWNER TO supervisor_ventas;
ALTER TABLE
sistema_ventas=# DROP ROLE asistente_ventas;
DROP ROLE
sistema_ventas=#
```

- ¿Qué riesgos implicaría forzar su eliminación?

No podemos “logearnos” con sus credenciales, porque ya no existe.

Pero la tabla que creó sigue existiendo: Nos conectamos desde la cuenta de supervisor _ventas

```
sistema_ventas=# \dt
              List of relations
 Schema |   Name    | Type  |       Owner
--------+-----------+-------+-------------------
 public | productos | table | supervisor_ventas
(1 row)

sistema_ventas=#
```

# Actividad 2 — Vistas y Accesos Controlados (2 puntos)

1. Crea las tablas e inserta datos:

LO VAMOS A HACER DESDE EL USUARIO “postgres”.

**clientes(id, nombre, telefono, email, saldo, vip boolean)**

```
CREATE TABLE clientes (
        id SERIAL PRIMARY KEY,
        nombre VARCHAR(100),
        telefono VARCHAR(20),
        email VARCHAR(100),
        saldo NUMERIC(10, 2) DEFAULT 0.00,
        vip BOOLEAN DEFAULT FALSE
);
```

```
sistema_ventas=# CREATE TABLE clientes (
        id SERIAL PRIMARY KEY,
        nombre VARCHAR(100),
        telefono VARCHAR(20),
        email VARCHAR(100),
        saldo NUMERIC(10, 2) DEFAULT 0.00,
        vip BOOLEAN DEFAULT FALSE
);
CREATE TABLE
sistema_ventas=# \DT
```

```
sistema_ventas=# \dt
              List of relations
 Schema |   Name    | Type  |       Owner
--------+-----------+-------+-------------------
 public | clientes  | table | postgres
 public | productos | table | supervisor_ventas
(2 rows)

sistema_ventas=#
```

**ventas(id, id_cliente, fecha, total, estado)**

**Para poner foreing key: lo del INTEGER REFERENCES**
https://www.postgresql.org/docs/current/tutorial-fk.html

```
CREATE TABLE ventas (
        id SERIAL PRIMARY KEY,
        id_cliente INTEGER REFERENCES clientes(id),
        fecha DATE DEFAULT CURRENT_DATE,
        total NUMERIC(10, 2),
        estado VARCHAR(50)
);
```

```
sistema_ventas=# CREATE TABLE ventas (
        id SERIAL PRIMARY KEY,
        id_cliente INTEGER REFERENCES clientes(id),
        fecha DATE DEFAULT CURRENT_DATE,
        total NUMERIC(10, 2),
        estado VARCHAR(50)
);
CREATE TABLE
sistema_ventas=# \dt
               List of relations
 Schema |   Name    | Type  |       Owner
--------+-----------+-------+-------------------
 public | clientes  | table | postgres
 public | productos | table | supervisor_ventas
 public | ventas    | table | postgres
(3 rows)

sistema_ventas=#
```

2. Inserta al menos 3 clientes y 3 ventas de ejemplo.

INSERT INTO clientes (id, nombre, telefono, email, saldo, vip) VALUES
('12345677', 'Perro Sanchez', '600111222', 'perrete@psoe.com', -9999999.00, TRUE),
('12345678', 'Donaldinho Trompino', '600333444', 'trump@usa.com', 5000000.00, FALSE),
('12345679', 'Vladimiro Putinesco', '600555666', 'ervladi@ruski.com', 1.00, FALSE);

```
sistema_ventas=# INSERT INTO clientes (id, nombre, telefono, email, saldo, vip) VALUES
('12345677', 'Perro Sanchez', '600111222', 'perrete@psoe.com', -9999999.00, TRUE),
('12345678', 'Donaldinho Trompino', '600333444', 'trump@usa.com', 5000000.00, FALSE),
('12345679', 'Vladimiro Putinesco', '600555666', 'ervladi@ruski.com', 1.00, FALSE);
INSERT 0 3
sistema_ventas=# SELECT * FROM clientes;
```

```
sistema_ventas=# SELECT * FROM clientes;
    id    |       nombre        | telefono  |       email       |    saldo    | vip
----------+---------------------+-----------+-------------------+-------------+-----
 12345677 | Perro Sanchez       | 600111222 | perrete@psoe.com  | -9999999.00 | t
 12345678 | Donaldinho Trompino | 600333444 | trump@usa.com     |  5000000.00 | f
 12345679 | Vladimiro Putinesco | 600555666 | ervladi@ruski.com |        1.00 | f
(3 rows)

sistema_ventas=#
```

INSERT INTO ventas (id, id_cliente, total, estado) VALUES

(1, 12345677, 300.00, 'Entregado'),

(2, 12345678, 50.50, 'Pendiente'),

(3, 12345677, 150.00, 'Entregado'),

(4, 12345679, 400.00, 'Entregado');

```
sistema_ventas=# INSERT INTO ventas (id, id_cliente, total, estado) VALUES
 (1, 12345677, 300.00, 'Entregado'),
 (2, 12345678, 50.50, 'Pendiente'),
 (3, 12345677, 150.00, 'Entregado'),
 (4, 12345679, 400.00, 'Entregado');
INSERT 0 4
sistema_ventas=#
```

```
sistema_ventas=# SELECT * FROM ventas;
 id | id_cliente |   fecha    | total  |  estado
----+------------+------------+--------+-----------
  1 |   12345677 | 2025-11-28 | 300.00 | Entregado
  2 |   12345678 | 2025-11-28 |  50.50 | Pendiente
  3 |   12345677 | 2025-11-28 | 150.00 | Entregado
  4 |   12345679 | 2025-11-28 | 400.00 | Entregado
(4 rows)

sistema_ventas=#
```

- Crea las siguientes vistas:

| Vista | Contenido | Quién accede |
|---|---|---|
| **vista_admin_clientes** | Todos los datos + suma total gastado por cliente | supervisor_ventas |
| **vista_asistente_clientes** | Nombre, teléfono, saldo, vip (sin email) | asistente_ventas |
| **vista_inspector_anonima** | Solo totales por cliente sin información personal | inspector |

**vista_admin_clientes:**

```
CREATE VIEW vista_admin_clientes AS
 SELECT c.*,
 COALESCE(SUM(v.total),0.00)
 AS total_gastado FROM clientes c LEFT JOIN ventas v ON c.id = v.id_cliente
 GROUP BY c.id
 ORDER BY c.id;
```

```
sistema_ventas=# SELECT c.*,
 COALESCE(SUM(v.total),0.00)
 AS total_gastado FROM clientes c LEFT JOIN ventas v ON c.id = v.id_cliente
 GROUP BY c.id
 ORDER BY c.id;
    id    |       nombre        | telefono  |       email       |    saldo    | vip | total_gastado
----------+---------------------+-----------+-------------------+-------------+-----+---------------
 12345677 | Perro Sanchez       | 600111222 | perrete@psoe.com  | -9999999.00 | t   |        450.00
 12345678 | Donaldinho Trompino | 600333444 | trump@usa.com     |  5000000.00 | f   |         50.50
 12345679 | Vladimiro Putinesco | 600555666 | ervladi@ruski.com |        1.00 | f   |        400.00
(3 rows)
```

```
sistema_ventas=# CREATE VIEW vista_admin_clientes AS
 SELECT c.*,
 COALESCE(SUM(v.total),0.00)
 AS total_gastado FROM clientes c LEFT JOIN ventas v ON c.id = v.id_cliente
 GROUP BY c.id
 ORDER BY c.id;
CREATE VIEW
sistema_ventas=#
```

**vista_asistente_clientes:**

```
CREATE VIEW vista_asistente_clientes AS
        SELECT nombre,
        telefono,
        saldo,
        vip FROM clientes;
```

```
sistema_ventas=# CREATE VIEW vista_asistente_clientes AS
        SELECT nombre,
        telefono,
        saldo,
        vip FROM clientes;
CREATE VIEW
sistema_ventas=#
```

```
sistema_ventas=#        SELECT nombre,
        telefono,
        saldo,
        vip FROM clientes;
       nombre        | telefono  |    saldo    | vip
---------------------+-----------+-------------+-----
 Perro Sanchez       | 600111222 | -9999999.00 | t
 Donaldinho Trompino | 600333444 |  5000000.00 | f
 Vladimiro Putinesco | 600555666 |        1.00 | f
(3 rows)

sistema_ventas=#
```

**vista_inspector_anonima:**

CREATE VIEW vista_inspector_anonima AS
SELECT id_cliente,
SUM(total) AS total_compras FROM ventas GROUP BY id_cliente;

```
sistema_ventas=# CREATE VIEW vista_inspector_anonima AS
        SELECT id_cliente,
        SUM(total) AS total_compras FROM ventas GROUP BY id_cliente;
CREATE VIEW
sistema_ventas=#
```

```
sistema_ventas=#        SELECT id_cliente,
        SUM(total) AS total_compras FROM ventas GROUP BY id_cliente;
 id_cliente | total_compras
------------+---------------
   12345677 |        450.00
   12345679 |        400.00
   12345678 |         50.50
(3 rows)
```

3. Revoca permisos directos sobre las tablas base a todos los usuarios excepto el superusuario.
   Concede permisos solamente a través de vistas.

QUITAR PERMISOS:

**REVOKE ALL ON clientes FROM supervisor_ventas;**
**REVOKE ALL ON ventas FROM supervisor_ventas;**
**REVOKE ALL ON clientes FROM inspector;**
**REVOKE ALL ON ventas FROM inspector;**

**asistente_ventas UNA VEZ QUE SE BORRÓ NO HA SIDO CREADO DE NUEVO. DE TODAS MANERAS:**

**REVOKE ALL ON clientes FROM asistente_ventas;**
**REVOKE ALL ON ventas FROM asistente_ventas;**

```
sistema_ventas=# REVOKE ALL ON clientes FROM supervisor_ventas;
REVOKE
sistema_ventas=# REVOKE ALL ON pedidos FROM supervisor_ventas;
ERROR:  relation "pedidos" does not exist
sistema_ventas=# \DT
invalid command \DT
Try \? for help.
sistema_ventas=# \dt
              List of relations
 Schema |   Name    | Type  |       Owner
--------+-----------+-------+-------------------
 public | clientes  | table | postgres
 public | productos | table | supervisor_ventas
 public | ventas    | table | postgres
(3 rows)

sistema_ventas=# REVOKE ALL ON ventas FROM supervisor_ventas;
REVOKE
sistema_ventas=# REVOKE ALL ON clientes FROM inspector;
REVOKE
sistema_ventas=# REVOKE ALL ON ventas FROM inspector;
REVOKE
sistema_ventas=# REVOKE ALL ON clientes FROM asistente_ventas;
REVOKE
sistema_ventas=# REVOKE ALL ON ventas FROM asistente_ventas;
REVOKE
sistema_ventas=#
```

**CONCEDER PERMISOS:**

**GRANT SELECT ON vista_admin_clientes TO supervisor_ventas;**

**GRANT SELECT ON vista_asistente_clientes TO asistente_ventas;**

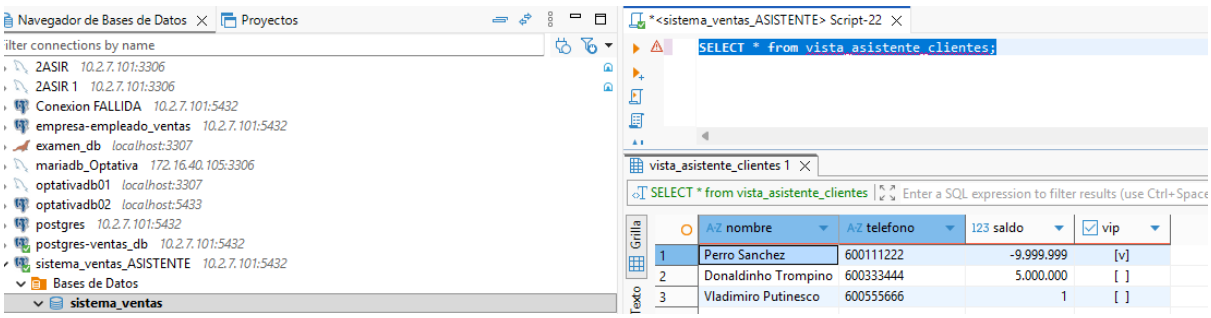
**GRANT SELECT ON vista_inspector_anonima TO inspector;**

```
sistema_ventas=# GRANT SELECT ON vista_admin_clientes TO supervisor_ventas;
GRANT
sistema_ventas=# GRANT SELECT ON vista_asistente_clientes TO asistente_ventas;
GRANT
sistema_ventas=# GRANT SELECT ON vista_inspector_anonima TO inspector;
GRANT
sistema_ventas=#
```

4. Demuestra las diferencias al consultar con cada usuario.

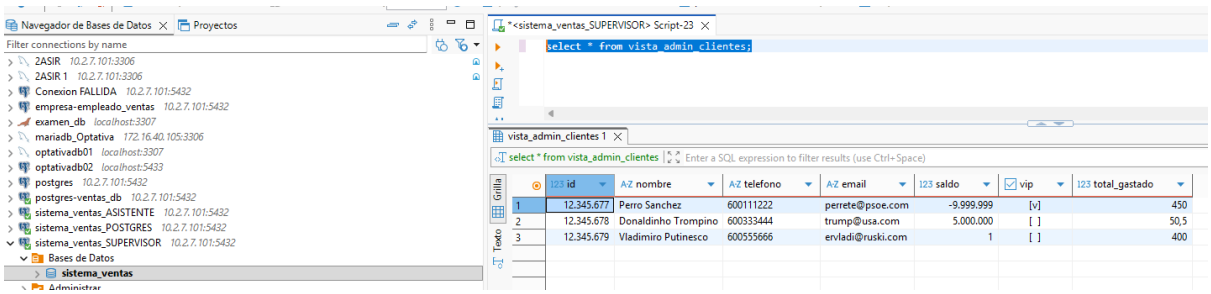
Desde asistente_ventas

**SELECT * from** vista_asistente_clientes;




Desde supervisor_ventas:

**select * from** vista_admin_clientes;

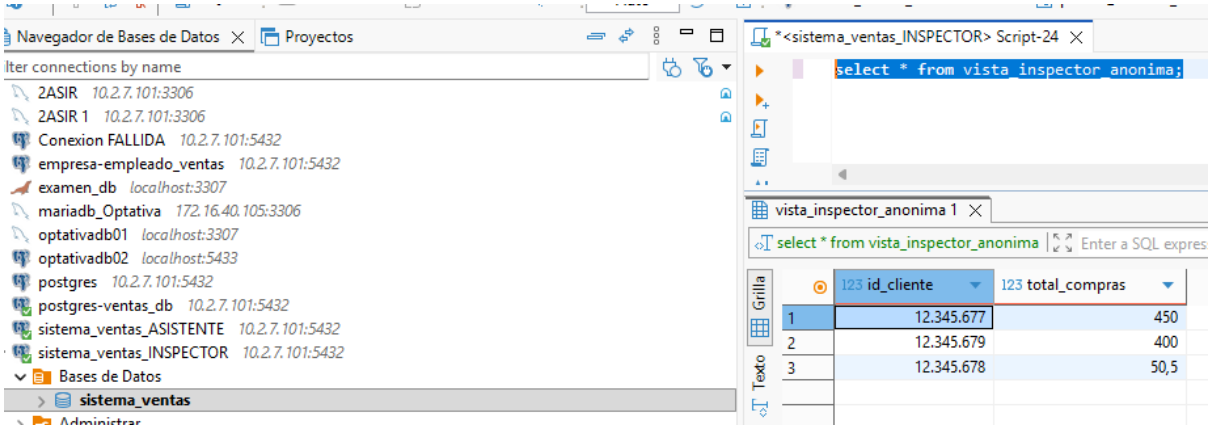



Desde inspector:

Puede que haya un problema con el limites de conexiones de inspector al usar DBEAVER:

```
sistema_ventas=# ALTER USER inspector CONNECTION LIMIT 8;
ALTER ROLE
sistema_ventas=#
```

**select * from** vista_inspector_anonima;




# Actividad 3 — Auditoría, Seguridad y Políticas (2 puntos)

1. Crea un rol adicional llamado **solo_lectura_global** y:

- Dale acceso de *solo lectura* a todas las tablas y vistas existentes.

CREATE ROLE solo_lectura_global NOLOGIN;

```
ALTER ROLE
sistema_ventas=# CREATE ROLE solo_lectura_global NOLOGIN;
CREATE ROLE
sistema_ventas=#
```

**GRANT SELECT ON ALL TABLES IN SCHEMA public TO solo_lectura_global;**
Para poder acceder a los objetos también hay que darle este permiso:

**GRANT USAGE ON SCHEMA public TO solo_lectura_global;**

```
CREATE ROLE
sistema_ventas=# GRANT SELECT ON ALL TABLES IN SCHEMA public TO solo_lectura_global;
GRANT
sistema_ventas=# GRANT USAGE ON SCHEMA public TO solo_lectura_global;
GRANT
sistema_ventas=#
```

- Haz que el usuario inspector herede este rol.

¿Te acuerdas que inspector tiene noinherit?

ALTER USER inspector INHERIT;

GRANT solo_lectura_global TO inspector;

```
GRANT
sistema_ventas=# GRANT solo_lectura_global TO inspector;
GRANT ROLE
sistema_ventas=#
```

- Asegura que cualquier objeto futuro creado en el esquema sea accesible en lectura por este rol.

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

```
sistema_ventas=# ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO solo_lectura_glob
al;
ALTER DEFAULT PRIVILEGES
sistema_ventas=#
```

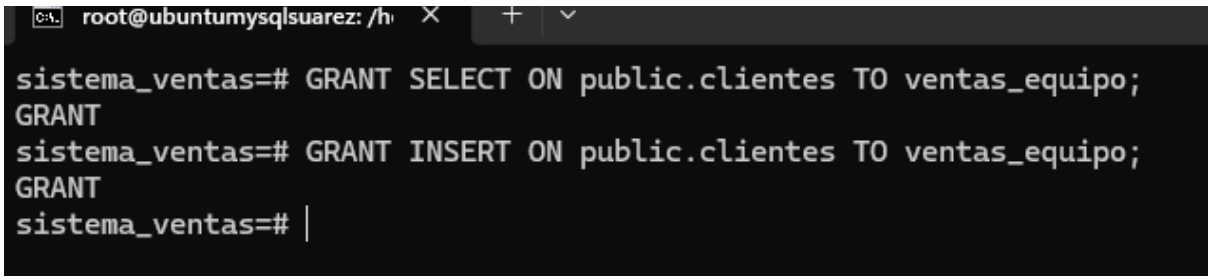
# Actividad 4 — Práctica de Revocación y Cambios de Privilegios (2 puntos)

1. Sobre la tabla clientes:

- Concede a **ventas_equipo** permisos de INSERT y SELECT.

GRANT SELECT ON public.clientes TO ventas_equipo;

GRANT INSERT ON public.clientes TO ventas_equipo;




- Concede a **supervisor_ventas** permiso de DELETE.

GRANT DELETE ON public.clientes TO **supervisor_ventas**;

```
sistema_ventas=# GRANT DELETE ON public.clientes TO supervisor_ventas;
GRANT
sistema_ventas=#
```

- Revoca explícitamente DELETE a **asistente_ventas**.

REVOKE DELETE ON public.clientes FROM **asistente_ventas**;

```
sistema_ventas=# REVOKE DELETE ON public.clientes FROM asistente_ventas;
REVOKE
sistema_ventas=#
```

2. Luego:

- Revoca TODOS los permisos del rol ventas_equipo sobre clientes.

REVOKE ALL ON clientes FROM ventas_equipo;

```
REVOKE
sistema_ventas=# REVOKE ALL ON clientes FROM ventas_equipo;
REVOKE
sistema_ventas=#
```

- Vuelve a conceder solo SELECT.

GRANT SELECT ON public.clientes TO ventas_equipo;

```
sistema_ventas=# GRANT SELECT ON public.clientes TO ventas_equipo;
GRANT
sistema_ventas=#
```

# Actividad 5 — Informe de Seguridad (2 puntos)

1. Modifica autenticación en `pg_hba.conf` para usar **scram-sha-256**.
   Activa el log y registra intentos de conexión fallidos.

nano /etc/postgresql/16/main/pg_hba.conf

```
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
host    all             segurisimo      0.0.0.0/0               scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
host    all             all             0.0.0.0/0               scram-sha-256
```

nano /etc/postgresql/16/main/postgresql.conf

Activamos loggin_collector

```
# This is used when logging to stderr:
logging_collector = on                  # Enable capturing of stderr, jsonlog,
                                        # and csvlog into log files. Required
```

Y después

```
#log_checkpoints = on
log_connections = on
log_disconnections = on
```

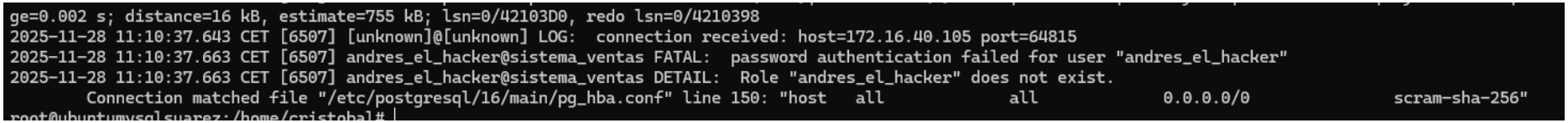
Algunos compañeros han dicho que han tenido que “descomentar” las siguientes líneas para que les funcione el registro. A mi no me ha hecho falta.

```
# These are only used if logging_collector is on:
#log_directory = 'log'                  # directory where log files are written,
                                        # can be absolute or relative to PGDATA
#log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'        # log file name pattern,
                                        # can include strftime() escapes
```

Entrega:

- Captura de logs con fallos:

Usuario sin credenciales intenta “logearse”.



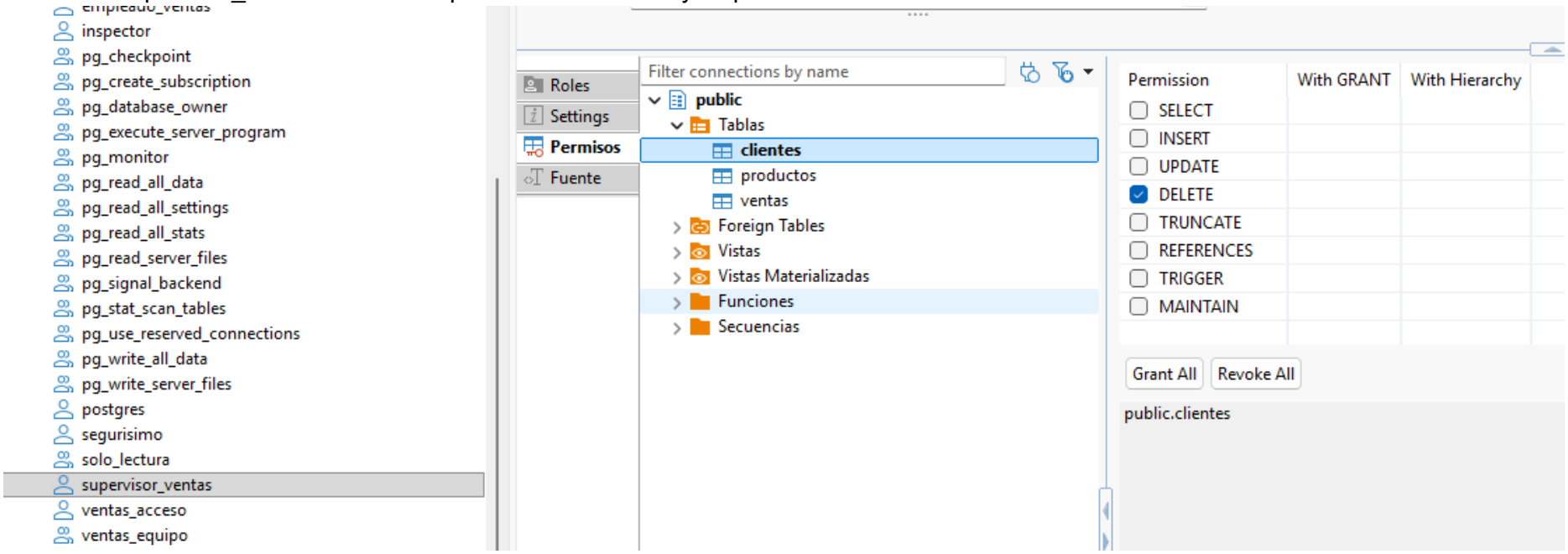
- Verifica los privilegios  de cada usuario

En la tabla clientes:

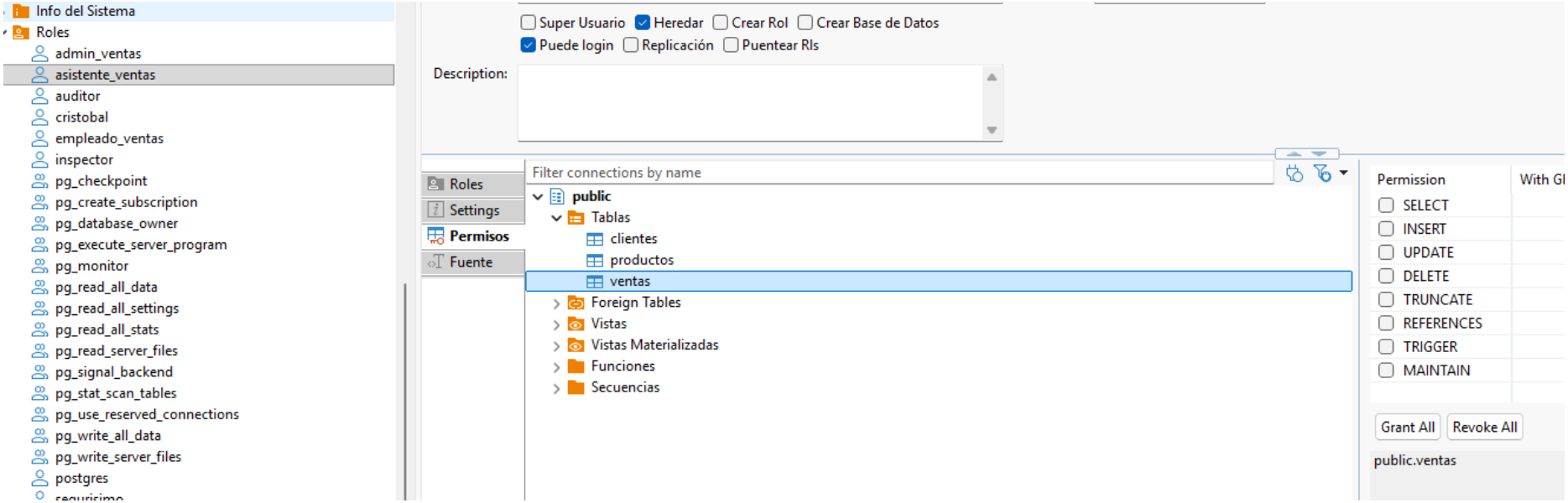

En la tabla ventas:

Desde DBeaver:

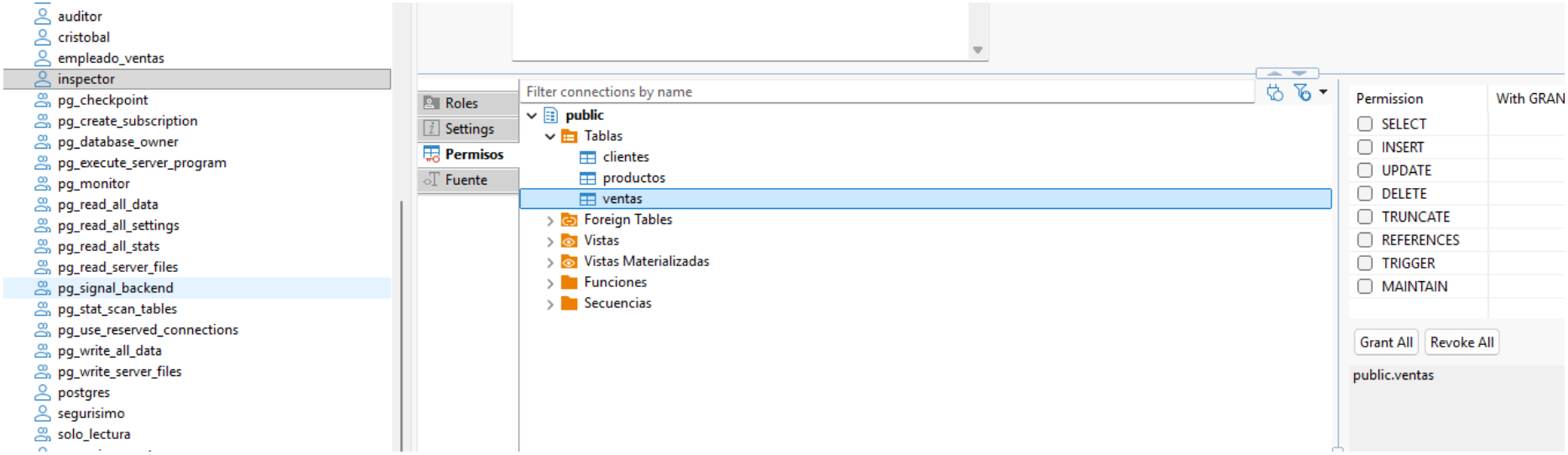
El usuario supervisor_ventas: Solo tiene permisos en clientes y es para “Borrar”delt



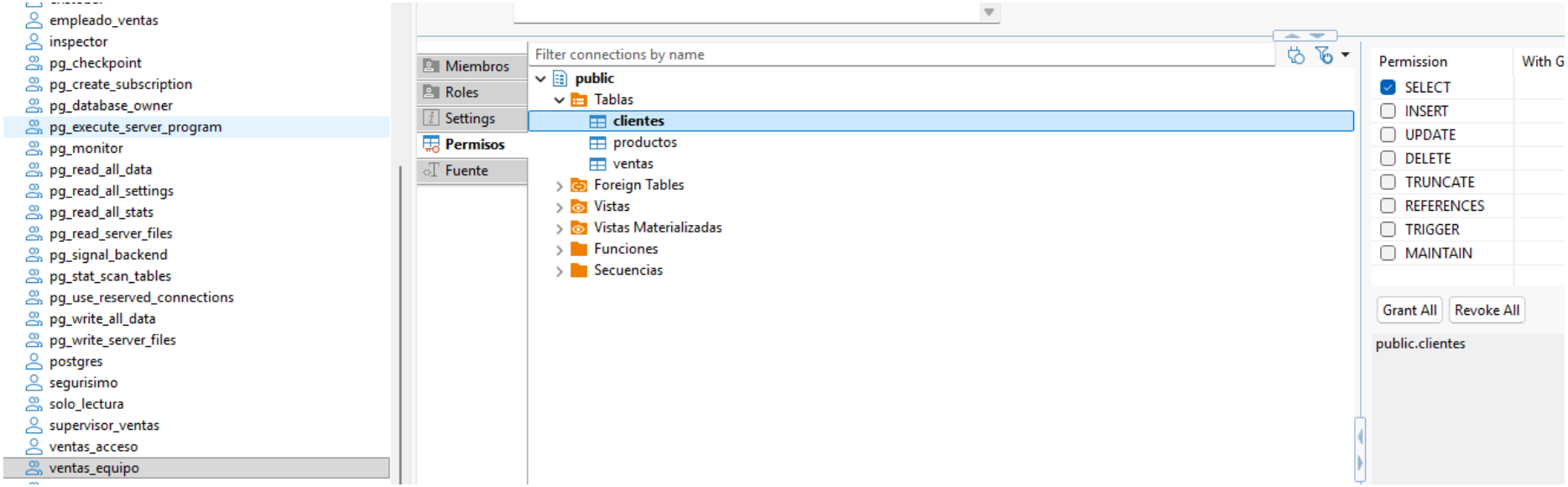
asistente_ventas no tiene ningún privilegio:



inspector: no tiene ningún privilegio.



Rol ventas_equipo: Puede hacer Select en “clientes”.



solo_lectura_global: Tiene Select en todas las tablas.