EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: BEGIN, COMMIT Y ROLLBACK.
- EXERCISE 2: TRANSACCIONES.
- EXERCISE 3: CONSISTENCIA Y AISLAMIENTO.
- EXERCISE 4: PRIVILEGIOS Y USUARIOS.

EXERCISE 1: BEGIN, COMMIT Y ROLLBACK

Es fundamental conocer el concepto de transacción en los sistemas gestores de bases de datos, como PostgreSQL. Una transacción empaqueta varios pasos en una operación, de forma que se completen todos, o ninguno de ellos. Los estados intermedios entre los pasos no son visibles para otras transacciones ocurridas en el mismo momento.

En el caso de que ocurra algún fallo que impida que se complete la transacción, ninguno de los pasos se ejecuta, y éstos no afectan a los objetos de la base de datos.

Los pasos dentro de una transacción son varias sentencias SQL, las cuales deben completarse todas para que queden registradas. Para comenzar una, utilizamos el comando BEGIN. Luego, para indicar al sistema que han terminado correctamente todas las sentencias SQL, se emplea el comando COMMIT. En algunas ocasiones, tenemos que desechar uno de los pasos que se están realizando, entonces, para cancelar la transacción comenzada, se usa el comando ROLLBACK.

COMANDO BEGIN

Cuando lo utilizamos, **el sistema permite que se ejecuten todas las sentencias SQL que necesitemos**, y las registra en un fichero. A continuación, revisaremos un ejemplo donde se comienza una transacción, en la que deben completarse satisfactoriamente todas las sentencias.

```
1 BEGIN;
2 UPDATE cuentas SET balance = balance - 100.00 WHERE n_cuenta =
3 0127365;
4 UPDATE cuentas SET balance = balance + 100.00 WHERE n_cuenta =
5 0795417;
```

COMANDO COMMIT

Al ejecutarlo, **estamos confirmando que todas las sentencias son correctas**. Es decir que, mientras no se haya implementado, las sentencias no quedarán registradas. Por ejemplo: **si cerramos la conexión antes de ejecutar este comando, no se verá afectada ninguna de las relaciones de la base de datos**. A continuación, veremos un ejemplo en el cual se comienza una transacción, se ejecutan una serie de pasos, y confirmamos que todas las sentencias están correctas.

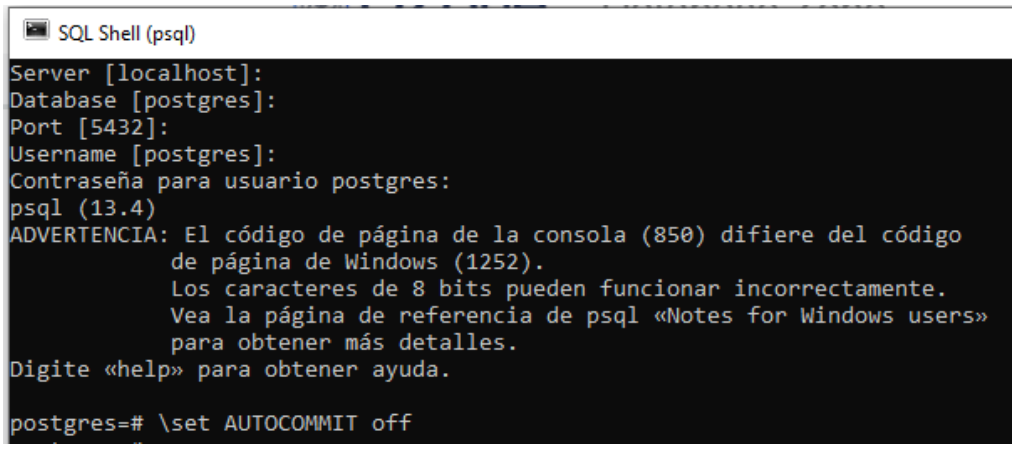
```
BEGIN;
INSERT INTO cuentas (n_cuenta, nombre, balance) VALUES (0679259,
'Pepe', 200);
UPDATE cuentas SET balance = balance - 137.00 WHERE nombre = 'Pepe';
UPDATE cuentas SET balance = balance + 137.00 WHERE nombre = 'Juan';
SELECT nombre, balance FROM cuentas WHERE nombre = 'Pepe' AND nombre =
'Juan';
COMMIT;
```

COMANDO ROLLBACK

Con éste **podemos desechar las transacciones que se hayan ejecutado**. Por lo tanto, después de haber realizado y confirmado una transacción, PostgreSQL nos permite anularla, de forma que no se modifiquen los datos de nuestra base de datos. A continuación, un ejemplo donde vamos a anular la transacción confirmada anteriormente.

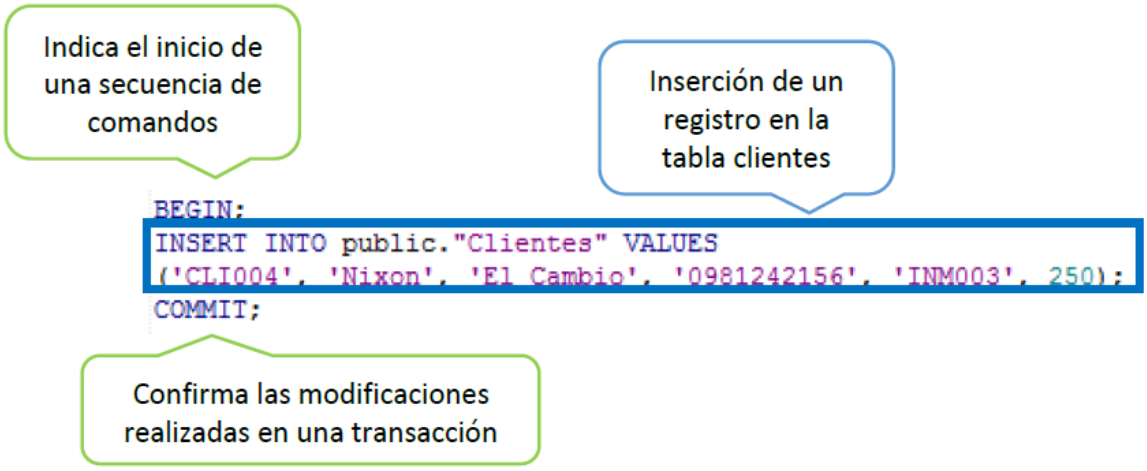
```
BEGIN;
"SENTENCIAS SQL"
COMMIT;
ROLLBACK;
```

Para poder **utilizar estos comandos mencionados: BEGIN, COMMIT y ROLLBACK, debemos desactivar el AUTOCOMMIT desde psql**. Esta opción es a nivel de cliente, y por defecto, está activada. Es así como toda sentencia ejecutada queda confirmada, y también registrada en la base de datos.



EXERCISE 2: TRANSACCIONES

Para comprender la propiedad de atomicidad, se presentan algunos ejemplos: el primero de éstos hace uso del comando COMMIT, el cual se utiliza para confirmar como permanentes, las modificaciones realizadas en una transacción:



Para comprobar que los comandos anteriores se ejecutaron correctamente, y se almacenaron en la base de datos, se hace una consulta en la tabla clientes:

```
1 SELECT * FROM "Clientes";
```

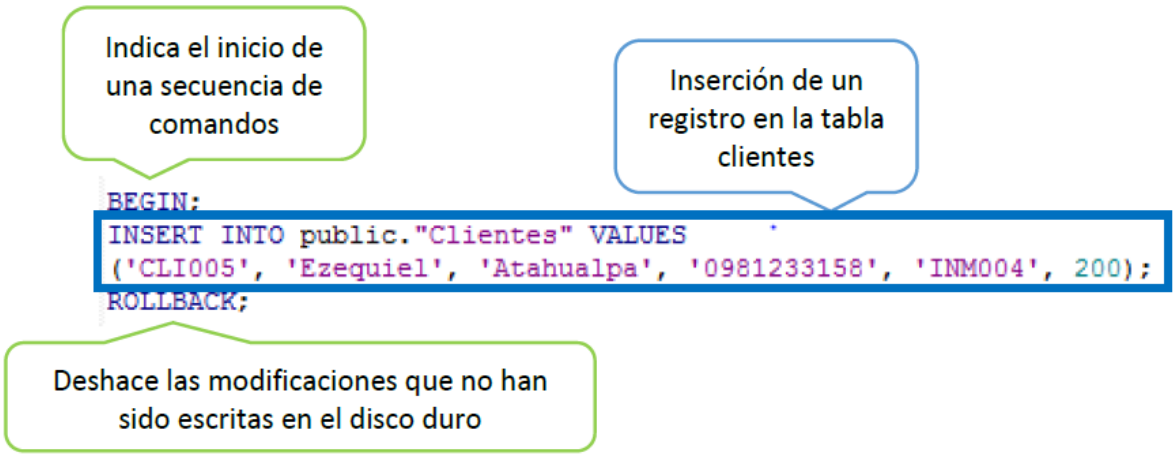
Su resultado es:

	id_cli character varying	nombre_cli character varying	direccion_cli character varying	telefono_cli character varying	inmueble_preferido_cli character varying	importe_maximo_cli double precision
1	CLI001	Ufredo Romero	Santa Rosa	2890910	INM001	300
2	CLI002	Diego Romero	Santa Rosa	2890911	INM002	150
3	CLI003	Jazmin Quirola	Machala	2890912	INM002	190
4	CLI004	Nixon	El Cambio	0981242156	INM003	250

Con esto se puede evidenciar que el registro de la transacción se ha guardado en la base de datos inmobiliaria de forma correcta.

Otra forma de comprobar la propiedad de atomicidad es la que se indica a continuación. En ésta se hace uso del comando ROLLBACK, el cual permite deshacer todas las modificaciones que se han realizado a la base de datos, pero que no han sido escritas en el disco duro por la sentencia COMMIT.

PROPIEDAD DE ATOMICIDAD.



Para comprobar que los comandos anteriores se ejecutaron, y que no se almacenaron en la base de datos, se hace una consulta en la tabla clientes:

```
1 SELECT * FROM "Clientes";
```

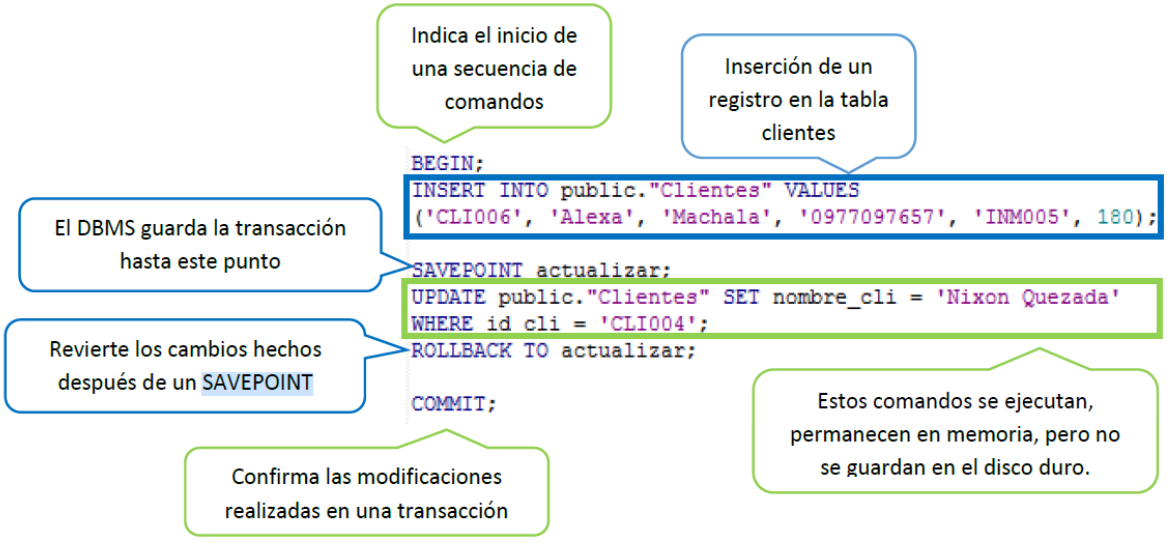
Su resultado es:

	id_cli character varying	nombre_cli character varying	direccion_cli character varying	telefono_cli character varying	inmueble_preferido_cli character varying	importe_maximo_cli double precision
1	CLI001	Ufredo Romero	Santa Rosa	2890910	INM001	300
2	CLI002	Diego Romero	Santa Rosa	2890911	INM002	150
3	CLI003	Jazmin Quirola	Machala	2890912	INM002	190
4	CLI004	Nixon	El Cambio	0981242156	INM003	250

Con ello, se puede evidenciar que el registro de la transacción no se ha guardado en la base de datos inmobiliaria.

Otros de los comandos que se pueden usar para la comprobación de la propiedad de atomicidad, son: SAVEPOINT y ROLLBACK TO. EL primero le indica al DBMS la ubicación de un punto de retorno en una transacción, en caso de que ésta sea cancelada. Mientras que el segundo, revierte los cambios hechos después de un SAVEPOINT.

COMPROBACIÓN DE ATOMICIDAD.



Para comprobar que los comandos anteriores se ejecutaron, y que éstos cumplen con las definiciones, se hace una consulta en la tabla clientes de la base de datos inmobiliaria:

```
SELECT * FROM "Clientes";
```

Su resultado es:

	id_cli character varying	nombre_cli character varying	direccion_cli character varying	telefono_cli character varying	inmueble_preferido_cli character varying	importe_maximo_cli double precision
1	CLI001	Ufredo Romero	Santa Rosa	2890910	INM001	300
2	CLI002	Diego Romero	Santa Rosa	2890911	INM002	150
3	CLI003	Jazmin Quirola	Machala	2890912	INM002	190
4	CLI004	Nixon	El Cambio	0981242156	INM003	250
5	CLI006	Alexa	Machala	0977097657	INM005	180

Con esto se evidencia que las instrucciones que se encuentran dentro del bloque SAVEPOINT <actualizar>, y ROLLACK TO <actualizar>, no han efectuado ningún cambio, pues no han sido escritas en el disco duro; y, en cambio, la información que se ubica fuera del bloque SAVEPOINT <actualizar> y ROLLACK TO <actualizar>, si ha sido almacenada en la base de datos.

Reglas ACID: comprobar la propiedad de consistencia, utilizando la tabla “Contrato”.

EXERCISE 3: CONSISTENCIA Y AISLAMIENTO

Para comprender la propiedad de consistencia, a continuación, se presentan algunos ejemplos, los cuales enfatizan sobre la integridad de los datos que lleva a cabo postgreSQL.

```
BEGIN;
INSERT INTO public. "Clientes" VALUES
('CLI004', 'Nixon', 'El Cambio', '0981242156', 'INM003', 250);
COMMIT;
```

PROPIEDAD DE CONSISTENCIA

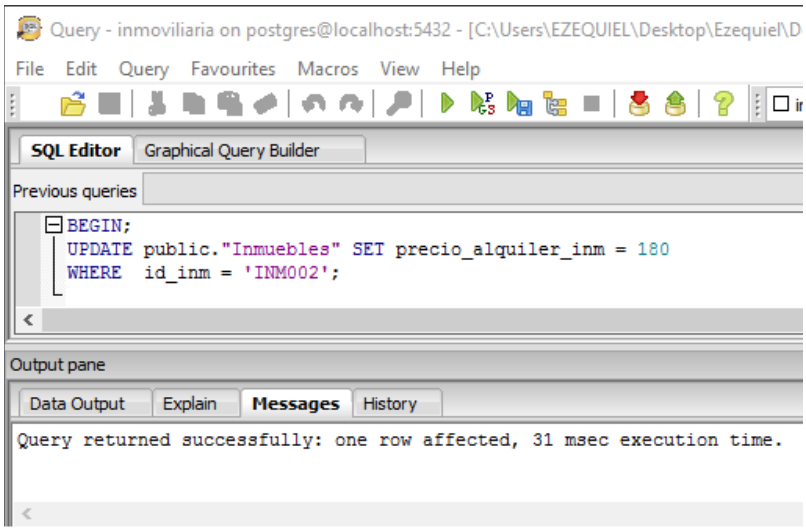
Al ejecutar la transacción anterior, se produce el siguiente resultado:

```
ERROR:  llave duplicada viola restricción de unicidad «Clientes_pkey»
DETAIL:  Ya existe la llave (id_cli)=(CLI004).
********** Error **********
```

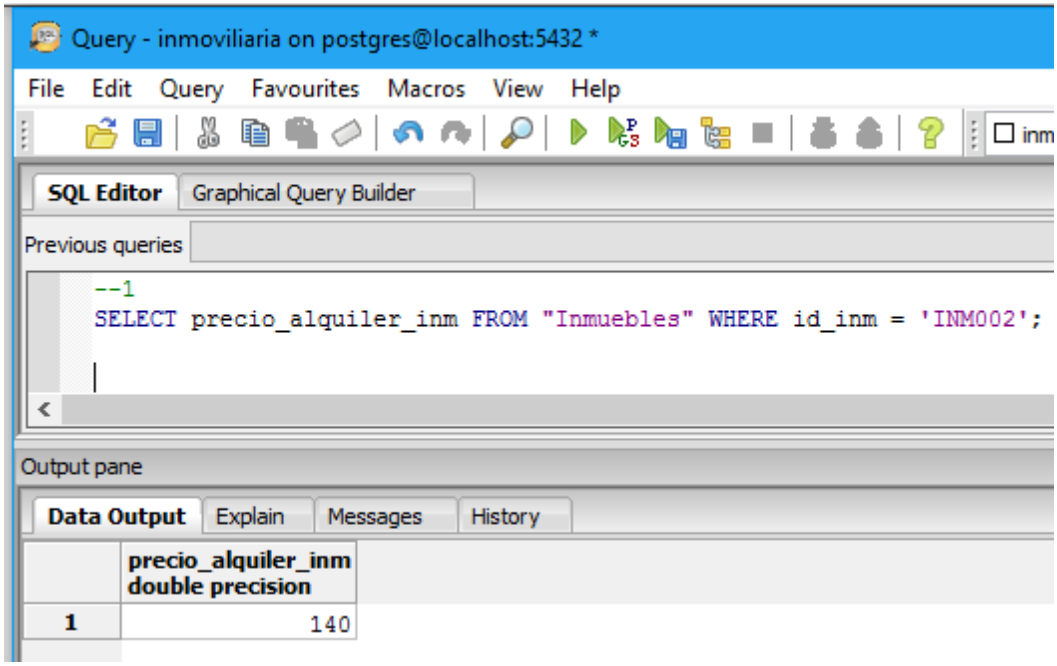
Esto ocurre debido a que ya existe un registro que contiene el valor de CLI004 como clave primaria. Con ello, se evidencia que postgreSQL es un DBMS que controla la integridad de los datos, y por lo tanto, cumple con la propiedad de consistencia.

Reglas ACID: verificaremos la propiedad de aislamiento (los cambios en una transacción no terminada no se ven en otra sesión), utilizando la tabla “Contrato”.

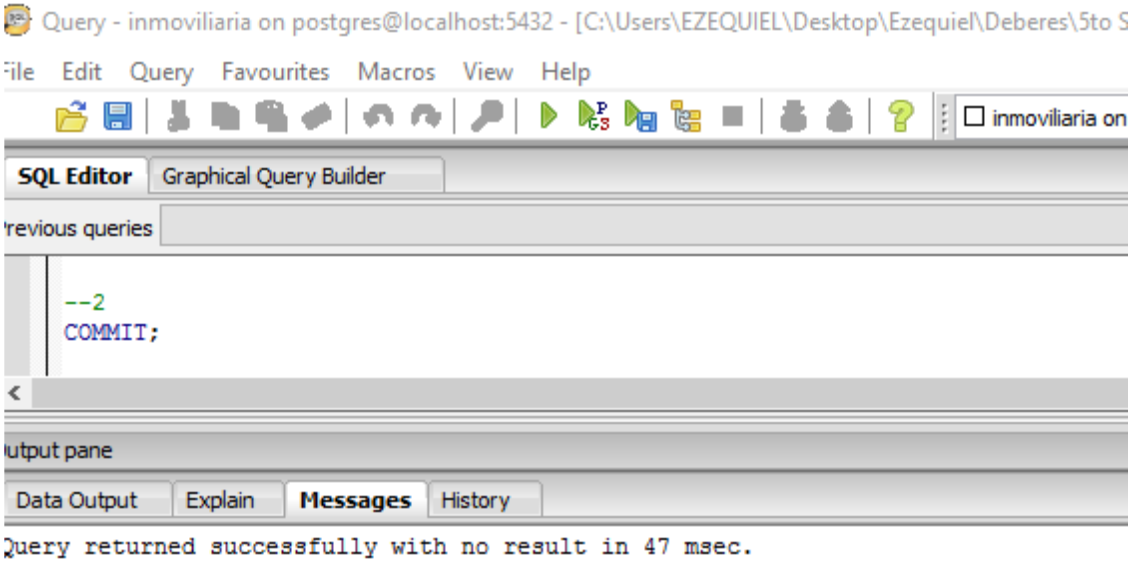
Para comprobar que postgreSQL cumple con la propiedad de aislamiento, se abren dos ventanas para ejecutar consultas SQL. En la primera se llevarán a cabo los siguientes comandos, con lo que se indica que la consulta fue exitosa:



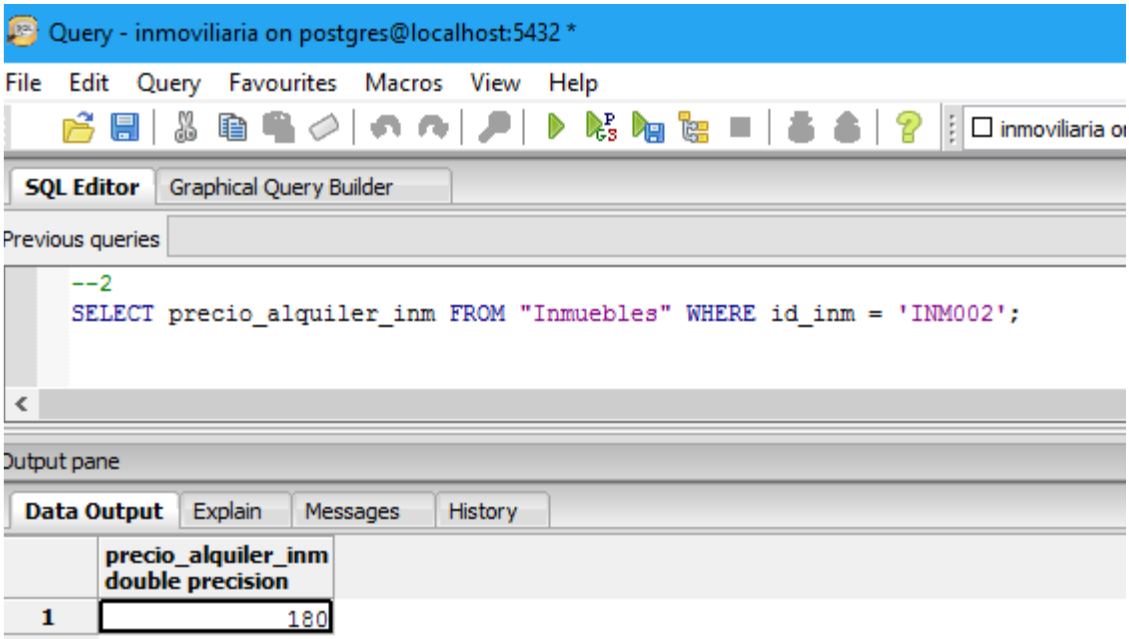
Si otro usuario deseara visualizar cierta información de la base de datos inmobiliaria, específicamente el precio de alquiler del inmueble cuyo código es INM002, se muestra el valor que estaba registrado en la base, y no el que se modificó en la ventana 1.



Al escribir el comando COMMIT en la ventana 1, y ejecutarlo, recién se estará guardando el registro en el disco duro.



Ahora, si se vuelve a generar la misma consulta en la ventana 2, se podrá visualizar que los cambios han sido almacenados en el disco duro.



EXERCISE 4: PRIVILEGIOS Y USUARIOS

GRUPOS DE USUARIOS

Esta propiedad nos permite agrupar a varios usuarios, con el objetivo de asignar privilegios de manera general para optimizar tiempo. Luego, podemos crear usuarios de manera independiente, y enlazarlos con algún grupo.

Sintaxis del grupo:

```
CREATE GROUP [nombregrupo]
```

Sintaxis del usuario:

```
CREATE USER [nombreusuario] WITH PASSWORD 'password' IN GROUP
[nombregrupo]
```

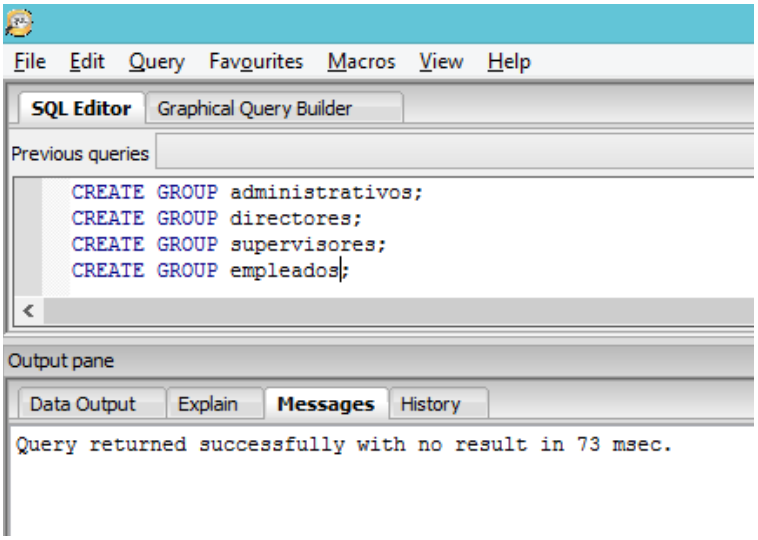
PRIVILEGIOS

Con la asignación de privilegios al usuario, se da la autorización a que éste, o un grupo, realice cualquier acción sobre una tabla específica. Dichas acciones pueden ser otorgadas con el comando “GRANT”, o a su vez, eliminadas con el comando “REVOKE”.

Su sintaxis es:

```
GRANT [SELECT, INSERT UPDATE, DELETE, ALL] ON [nombretabla] TO
[nombreusuario o nombreGrupo]
```

Bajo estos conceptos, aplicaremos lo mismo a la base de datos llamada inmobiliaria. Primero creamos los 4 grupos principales: Administrativo, Director, Supervisor, Empleado.



Luego, nos guiaremos en la siguiente matriz de trazabilidad, donde observamos qué acciones se han asignado a los respectivos grupos, en las diferentes tablas de nuestra base de datos.

Tabla Matriz de Trazabilidad de los Usuarios.

	Administrativo	Director	Supervisor	Empleado
Personas				
Directores				
Empleados				
Pariente				
Oficinas				
Visitas				
Clientes				
Inspección				
Inmuebles				
Inmuebles - Factura				
Factura				
Periódico				
Publicidad				
Propietario				
Contrato				
Pago				

SELECT

ALL

Por lo tanto, quedaría así:

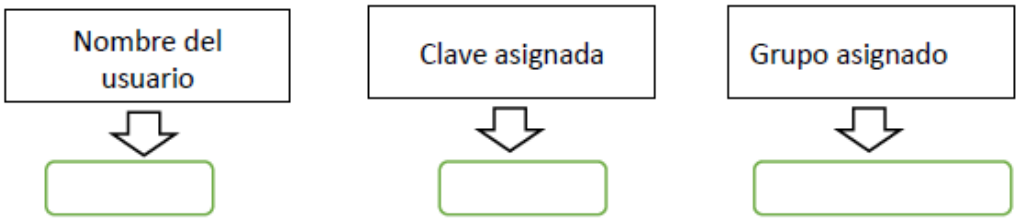
```
GRANT ALL ON "Personas" TO GROUP administrativos;
GRANT ALL ON "Directores" TO GROUP administrativos;
GRANT ALL ON "Empleados" TO GROUP administrativos;
GRANT ALL ON "Pariente" TO GROUP administrativos;
GRANT ALL ON "Oficinas" TO GROUP administrativos;
GRANT ALL ON "Visitas" TO GROUP administrativos;
GRANT ALL ON "Clientes" TO GROUP administrativos;
GRANT ALL ON "Inspeccion" TO GROUP administrativos;
GRANT ALL ON "Inmuebles" TO GROUP administrativos;
GRANT ALL ON "Inmueble_Factura" TO GROUP administrativos;
GRANT ALL ON "Facturas" TO GROUP administrativos;
GRANT ALL ON "Periodico" TO GROUP administrativos;
GRANT ALL ON "Publicidad" TO GROUP administrativos;
GRANT ALL ON "Propietario" TO GROUP administrativos;
GRANT ALL ON "Contrato" TO GROUP administrativos;
GRANT ALL ON "Pago" TO GROUP administrativos;
```

Output pane

Data Output | Explain | **Messages** | History

```
Query returned successfully with no result in 23 msec.
```

Una vez generados estos grupos de usuarios, procedemos a crear a los respectivos usuarios, basándonos en la sintaxis presentada anteriormente. Se creará uno por grupo, y se podrán añadir más dependiendo las necesidades de la empresa.




```
CREATE USER admin_jose WITH PASSWORD 'clave123' IN GROUP administrativos;
CREATE USER director_juan WITH PASSWORD 'abc123' IN GROUP directores;
CREATE USER superv_carlos WITH PASSWORD 'abc***' IN GROUP supervisores;
CREATE USER empl_luis WITH PASSWORD 'password' IN GROUP empleados;
```

Si ejecutamos la sentencia: SELECT * FROM pg_shadow, podremos ver como los usuarios han sido creados.

```
SELECT * from pg_shadow;
```

Output pane

Data Output | Explain | Messages | History

	usename name	usesysid oid	usecreatedb boolean	usesuper boolean	userepl boolean	usebypassrls boolean	passwd text	valuntil abstime	useconfig text[]
7	admin_jose	25163	f	f	f	f	md5f88b89f6053da685eb10dc9987f93a0f		
8	director_juan	25164	f	f	f	f	md574a9f7b3165b009439ae89794a363bf5		
9	superv_carlos	25165	f	f	f	f	md53d90de7a08b80e7d61388bcd5c4a3424		
10	empl_luis	25166	f	f	f	f	md510bee7d2eddfce65b4c517f4c149570f		

Y si queremos modificar, por ejemplo, la contraseña, solo es necesario ejecutar la sentencia ALTER USER:

```
1 ALTER USER admin_jose WITH PASSWORD 'nueva_password';
```