

# PRÁCTICA 4.2

Modificación de permisos y acceso a la información. PostgreSQL.

## DATOS:

- **Ciclo formativo:** Grado Superior 2º ASIR A
- **Módulo:** Administración de Sistemas Gestores de Bases de Datos
- **Unidad de trabajo:** UD04 – Configuración de una SGBD
- **Nombre y apellidos:** MC Pareja Ferreira

## ÍNDICE

<b>APARTADO 1. CREACIÓN DE TABLAS E INSERCCIÓN DE DATOS .....</b>	<b>3</b>
<b>APARTADO 2. CREACIÓN DE USUARIOS Y ROLES .....</b>	<b>4</b>
<b>APARTADO 3. COMPROBACIONES .....</b>	<b>6</b>
Primera conexión a la base de datos con los nuevos usuarios .....	6
Consulta de filas .....	6
Modificación de filas.....	7
Eliminación de filas .....	8
<b>APARTADO 4. CREACIÓN DE VISTAS Y MÁS ASIGNACIONES DE PERMISOS .....</b>	<b>9</b>

## APARTADO 1. CREACIÓN DE TABLAS E INSERCCIÓN DE DATOS

Para esta práctica hemos preparados cuatro tablas:

- **Usuarios:** id, nombre de usuario, email y contraseña.
- **Productos:** id, nombre, descripción y precio.
- **Pedidos:** id, id del usuario y fecha del pedido.
- **Detalles del pedido:** id, id del pedido, id del producto y cantidad de ese producto.



```
9/2/2023 11:02:05      129 msec
Date      Rows affected      Duration

Copy Copy to Query Editor

CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL
);

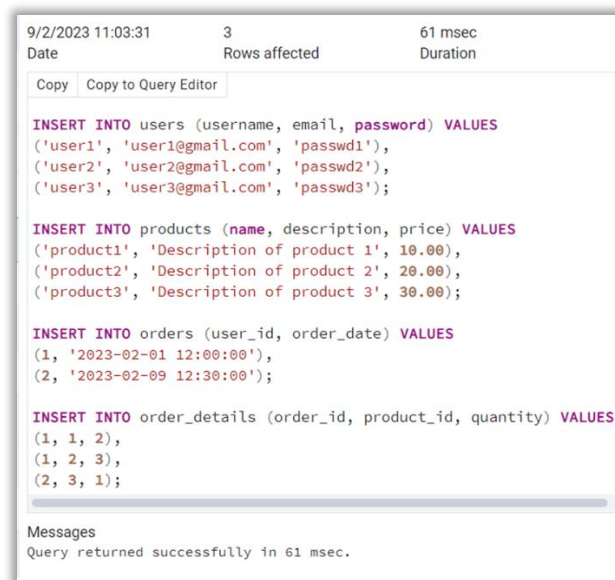
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  description TEXT NOT NULL,
  price NUMERIC(10, 2) NOT NULL
);

CREATE TABLE orders (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  order_date TIMESTAMP NOT NULL
);

CREATE TABLE order_details (
  id SERIAL PRIMARY KEY,
  order_id INTEGER REFERENCES orders(id),
  product_id INTEGER REFERENCES products(id),
  quantity INTEGER NOT NULL
);

Messages
Query returned successfully in 129 msec.
```

También insertaremos algunos datos de prueba para realizar futuras comprobaciones con los usuarios y los roles.



```
9/2/2023 11:03:31      3      61 msec
Date      Rows affected      Duration

Copy Copy to Query Editor

INSERT INTO users (username, email, password) VALUES
('user1', 'user1@gmail.com', 'passwd1'),
('user2', 'user2@gmail.com', 'passwd2'),
('user3', 'user3@gmail.com', 'passwd3');

INSERT INTO products (name, description, price) VALUES
('product1', 'Description of product 1', 10.00),
('product2', 'Description of product 2', 20.00),
('product3', 'Description of product 3', 30.00);

INSERT INTO orders (user_id, order_date) VALUES
(1, '2023-02-01 12:00:00'),
(2, '2023-02-09 12:30:00');

INSERT INTO order_details (order_id, product_id, quantity) VALUES
(1, 1, 2),
(1, 2, 3),
(2, 3, 1);

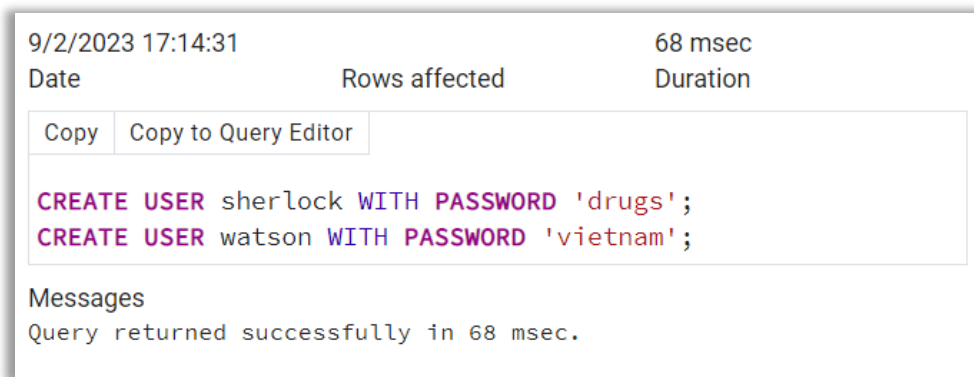
Messages
Query returned successfully in 61 msec.
```

## APARTADO 2. CREACIÓN DE USUARIOS Y ROLES

En PostgreSQL, la sentencia **CREATE ROLE** se utiliza para crear tanto roles como usuarios, ya que se tratan de conceptos muy similares.

Un usuario es un tipo especial de rol con permisos de LOGIN en la base de datos. Además de **CREATE ROLE**, en PostgreSQL 15 existe la posibilidad de usar el alias **CREATE USER**, que genera un rol con permisos de inicio de sesión por defecto. Esto significa que, una vez creado, el usuario puede conectarse a la base de datos y acceder a los objetos de la base de datos con los permisos asignados al rol.

Para esta práctica, trabajaremos con dos usuarios: Sherlock y Watson.



```
9/2/2023 17:14:31          68 msec
Date                    Rows affected  Duration

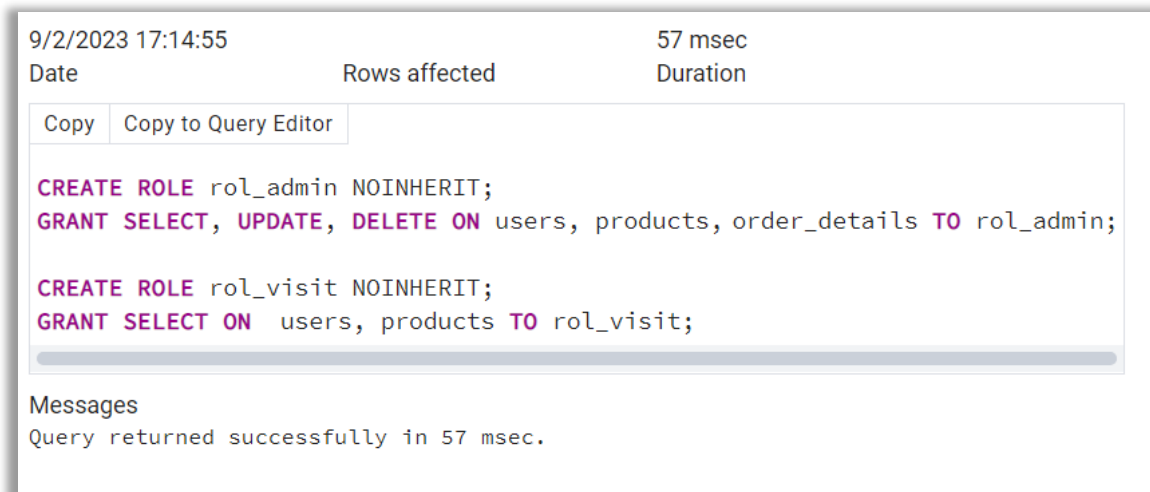
Copy  Copy to Query Editor

CREATE USER sherlock WITH PASSWORD 'drugs';
CREATE USER watson WITH PASSWORD 'vietnam';

Messages
Query returned successfully in 68 msec.
```

Por el contrario, cuando un rol carece de permisos de inicio de sesión, se considera un rol en el sentido más literal de la palabra, y se usará para agrupar usuarios y otorgar permisos de manera común en lugar de asignarlos a usuarios individuales.

Para esta práctica, trabajaremos con dos roles: rol\_admin y rol\_visit.



```
9/2/2023 17:14:55          57 msec
Date                    Rows affected  Duration

Copy  Copy to Query Editor

CREATE ROLE rol_admin NOINHERIT;
GRANT SELECT, UPDATE, DELETE ON users, products, order_details TO rol_admin;

CREATE ROLE rol_visit NOINHERIT;
GRANT SELECT ON users, products TO rol_visit;

Messages
Query returned successfully in 57 msec.
```

A través de la sentencia **GRANT**, hemos otorgado los permisos necesarios al rol\_admin para que sus miembros puedan ver, actualizar y borrar las filas de las cuatro tablas existentes en la base de datos; mientras que los miembros de rol\_visit, solo podrán ver las filas de dos de las cuatro tablas.

Para la asignación de miembros a un rol, también usaremos la sentencia **GRANT** de la siguiente manera:

9/2/2023 17:15:43

40 msec

Date	Rows affected	Duration
------	---------------	----------

Copy

Copy to Query Editor

```
GRANT rol_admin TO sherlock;
GRANT rol_visit TO watson;
```

Messages

Query returned successfully in 40 msec.

Si nos dirigimos al menú lateral, observaremos que la lista de usuarios/roles ha crecido.

pg\_write\_server\_files

postgres

rol\_admin

rol\_visit

sherlock

watson

Tablespaces

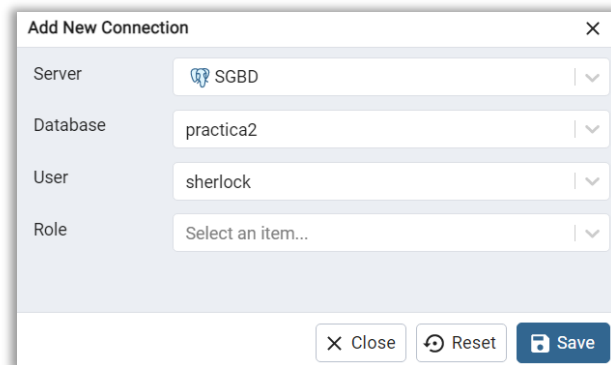
GRA

Que

## APARTADO 3. COMPROBACIONES

### Primera conexión a la base de datos con los nuevos usuarios

A la hora de añadir nuevas conexiones a una base de datos, se ofrece la posibilidad de asignar un rol al usuario dueño de la conexión. Sin embargo, debemos saber que esta asignación durará lo que dure la conexión, por lo que si queremos que la asignación sea de forma permanente deberemos seguir los pasos explicados en el apartado anterior.

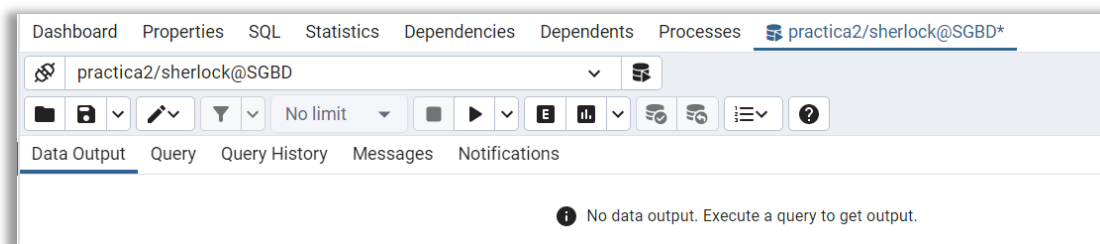


The 'Add New Connection' dialog box contains the following fields:

- Server: SGBD
- Database: practica2
- User: sherlock
- Role: Select an item...

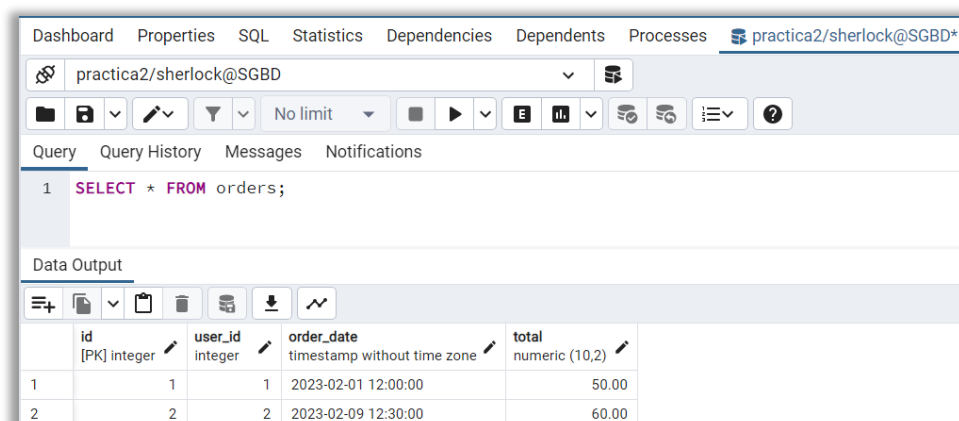
Buttons at the bottom: Close, Reset, Save.

En nuestro caso, al realizar la asignación permanente, no será necesario especificar rol.



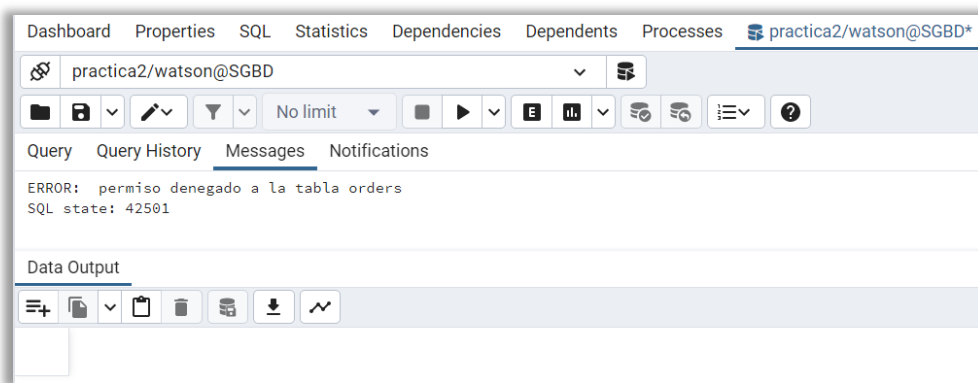
### Consulta de filas

Podemos comprobar como el usuario Sherlock perteneciente al rol\_admin puede consultar filas de cualquier tabla; mientras que el usuario Watson solo puede consultar filas en dos de ellas.



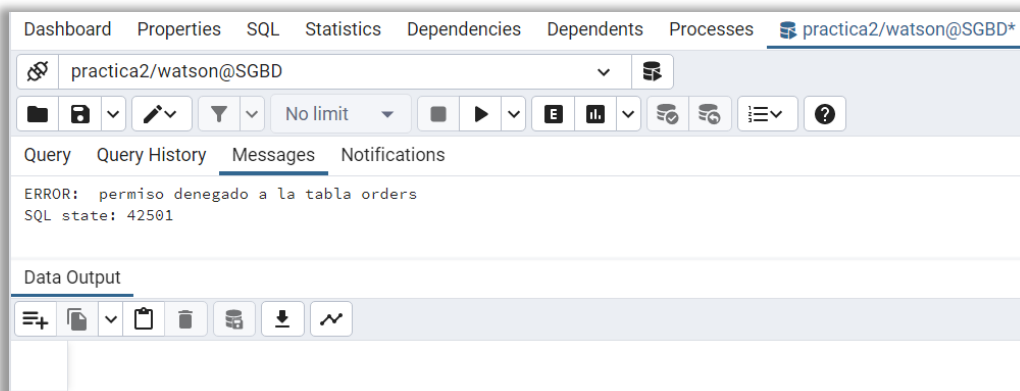
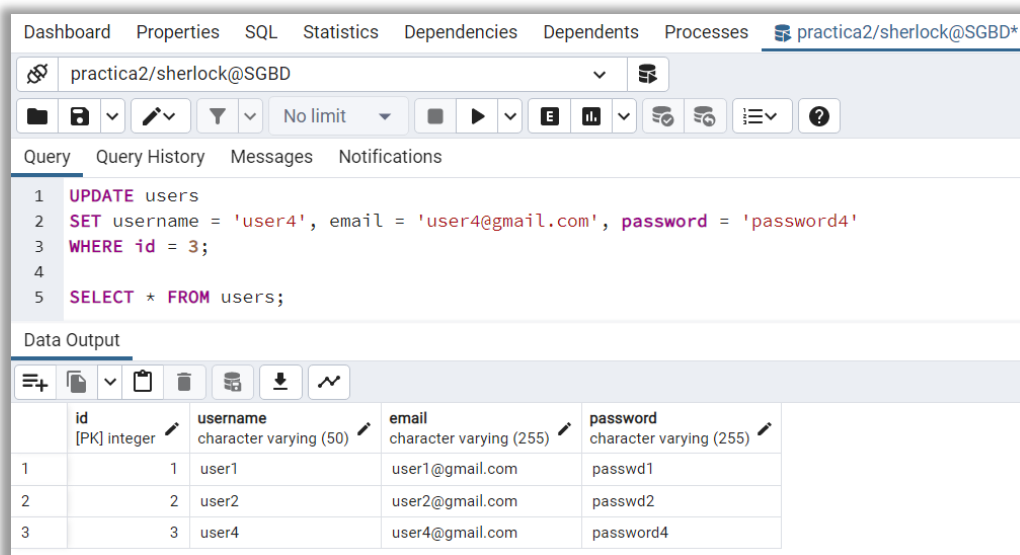
The screenshot shows the database client interface with the 'practica2/sherlock@SGBD' connection selected. The 'Query' tab is active, displaying the query: `SELECT * FROM orders;`. The 'Data Output' tab is also active, displaying the results of the query in a table format.

	id [PK] integer	user_id integer	order_date timestamp without time zone	total numeric (10,2)
1	1	1	2023-02-01 12:00:00	50.00
2	2	2	2023-02-09 12:30:00	60.00



## Modificación de filas

Podemos comprobar como el usuario Sherlock perteneciente al rol\_admin puede modificar filas de cualquier tabla; mientras que el usuario Watson no tiene permisos de modificación en ninguna tabla.



## Eliminación de filas

Podemos comprobar como el usuario Sherlock perteneciente al rol\_admin puede eliminar filas de cualquier tabla; mientras que el usuario Watson no tiene permiso para eliminar filas en ninguna de ellas.

Dashboard Properties SQL Statistics Dependencies Dependents Processes [practica2/sherlock@SGBD\\*](#)

practica2/sherlock@SGBD

No limit

Query Query History Messages Notifications

```
1 DELETE FROM users WHERE username = 'user4';
2
3 SELECT * FROM users;
```

Data Output

	id [PK] integer	username character varying (50)	email character varying (255)	password character varying (255)
1	1	user1	user1@gmail.com	passwd1
2	2	user2	user2@gmail.com	passwd2

Dashboard Properties SQL Statistics Dependencies Dependents Processes [practica2/watson@SGBD\\*](#)

practica2/watson@SGBD

No limit

Query Query History Messages Notifications

ERROR: permiso denegado a la tabla orders  
SQL state: 42501

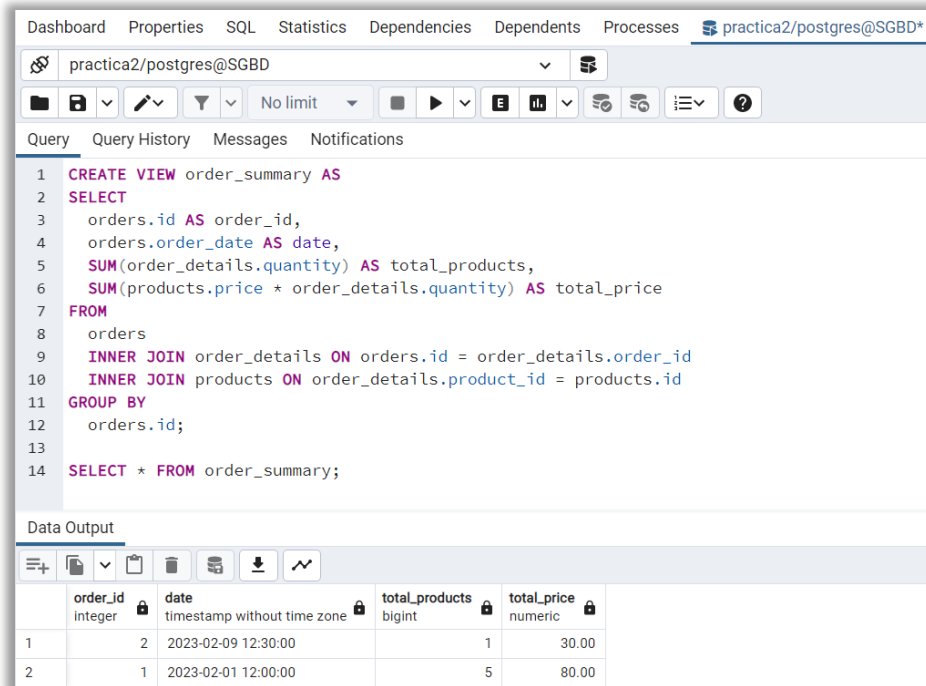
Data Output



## APARTADO 4. CREACIÓN DE VISTAS Y MÁS ASIGNACIONES DE PERMISOS

Para crear una vista en PostgreSQL, debemos ejecutar una consulta SELECT junto a una sentencia CREATE VIEW.

La primera vista recoge información de tres tablas para mostrar un resumen de la venta compuesto del id, fecha, total de productos adquiridos y precio total del pedido.



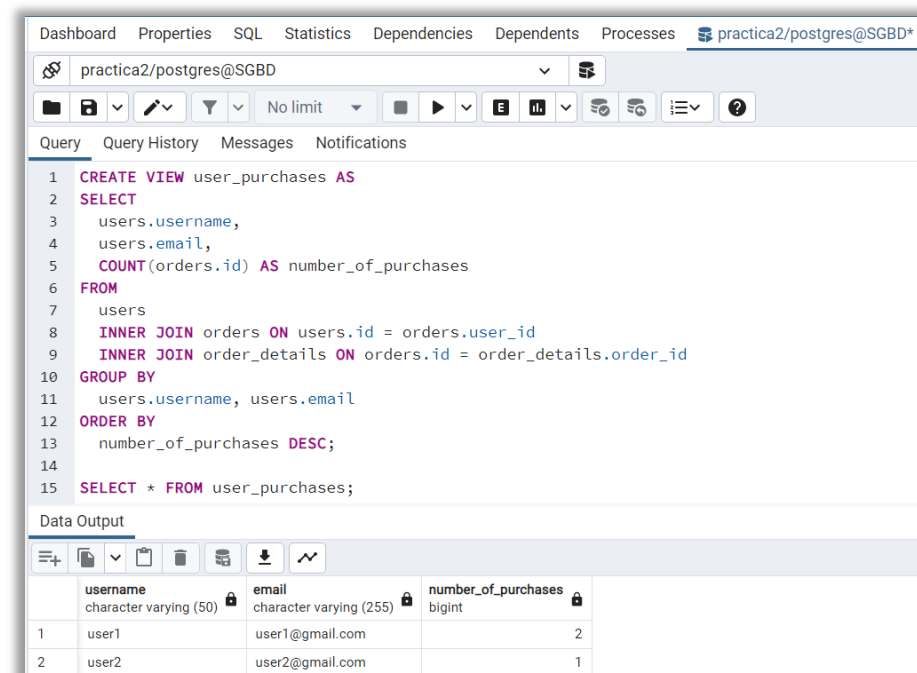
The screenshot shows the PostgreSQL IDE interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and Processes. The main window displays a SQL query to create a view named 'order\_summary' and then select from it. The query is as follows:

```
1 CREATE VIEW order_summary AS
2 SELECT
3     orders.id AS order_id,
4     orders.order_date AS date,
5     SUM(order_details.quantity) AS total_products,
6     SUM(products.price * order_details.quantity) AS total_price
7 FROM
8     orders
9     INNER JOIN order_details ON orders.id = order_details.order_id
10    INNER JOIN products ON order_details.product_id = products.id
11 GROUP BY
12     orders.id;
13
14 SELECT * FROM order_summary;
```

Below the query, the 'Data Output' section shows the results of the SELECT statement. The output is a table with the following columns: order\_id (integer), date (timestamp without time zone), total\_products (bigint), and total\_price (numeric).

	order_id	date	total_products	total_price
1	2	2023-02-09 12:30:00	1	30.00
2	1	2023-02-01 12:00:00	5	80.00

La segunda vista recoge información de tres tablas para mostrar el username de cada usuario, su email y el número de pedidos realizados.



The screenshot shows the PostgreSQL IDE interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and Processes. The main window displays a SQL query to create a view named 'user\_purchases' and then select from it. The query is as follows:

```
1 CREATE VIEW user_purchases AS
2 SELECT
3     users.username,
4     users.email,
5     COUNT(orders.id) AS number_of_purchases
6 FROM
7     users
8     INNER JOIN orders ON users.id = orders.user_id
9     INNER JOIN order_details ON orders.id = order_details.order_id
10 GROUP BY
11     users.username, users.email
12 ORDER BY
13     number_of_purchases DESC;
14
15 SELECT * FROM user_purchases;
```

Below the query, the 'Data Output' section shows the results of the SELECT statement. The output is a table with the following columns: username (character varying (50)), email (character varying (255)), and number\_of\_purchases (bigint).

	username	email	number_of_purchases
1	user1	user1@gmail.com	2
2	user2	user2@gmail.com	1

Finalmente, si queremos que nuestros usuarios puedan acceder a estas vistas, será necesario ejecutar la siguiente sentencia.

9/2/2023 20:42:19

49 msec

Date

Rows affected

Duration

Copy

Copy to Query Editor

GRANT SELECT ON order\_summary TO sherlock, watson;

Messages

Query returned successfully in 49 msec.

Dashboard

Properties

SQL

Statistics

Dependencies

Dependents

Processes

practica2/watson@SGBD\*

practica2/watson@SGBD

No limit

Query

Query History

Messages

Notifications

1

SELECT \* FROM order\_summary;

Data Output

	order_id integer	date timestamp without time zone	total_products bigint	total_price numeric
1	2	2023-02-09 12:30:00	1	30.00
2	1	2023-02-01 12:00:00	5	80.00