

Trabajo Práctico de PostgreSQL con Docker y DBeaver

Alumno: Yacob, Guillermo

Materia: Gestión de Bases de Datos

Profesor: Telesco, Lucas

Instituto Superior de Formación Docente y Técnica n° 166

Introducción

En el presente trabajo simularé una situación real de levantar una base de datos en un servidor Linux, al ser una situación simulada en este caso trabajaré sobre Windows.

En primera instancia se instaló Docker desde su página oficial, seguidamente a esto también se tuvo que instalar el WSL(Windows Subsystem for Linux) desde un enlace que provee el mismo Docker que lleva a una página oficial de Microsoft, ya que Docker es nativo de Linux y esto es necesario para que pueda funcionar en Windows. También se instaló DBeaver como herramienta de gestión de bases de datos de alto nivel desde su web oficial. Cabe destacar que tanto Docker como DBeaver son de uso gratuito.

- Sitio oficial de Docker → <https://docs.docker.com/desktop/windows/install/>
- Windows Subsystem for Linux (WSL) → <https://docs.microsoft.com/en-us/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package>
- Sitio oficial de DBeaver → <https://dbeaver.io/download/>

Docker nos permite virtualizar de manera más óptima que una máquina virtual, esto lo logra creando *contenedores* que compartirán recursos con el sistema operativo sin necesidad de levantar un sistema operativo para dicha aplicación.

Para utilizarlo, tenemos que entender también el concepto de *imagen*, que no es más que un equivalente a un instalador que genera un contenedor. Para crear un contenedor y así poder usar un programa o aplicación con Docker necesitaremos de una imagen, la cual la podemos generar nosotros mismos a partir de otra imagen previa o bajarla de una fuente confiable como DockerHub.

- Sitio oficial de DockerHub → <https://hub.docker.com/>

Si queremos trabajar desde Windows o Mac con Docker, primero tenemos que iniciarlo (cosa que no es necesaria en Linux). Una vez ya iniciado, para trabajar con PostgreSQL tenemos dos opciones: o bien le indicamos a Docker que queremos bajar la imagen oficial de PostgreSQL con el comando `docker pull postgres` o bien nos salteamos éste paso y vamos directo a la creación del contenedor, ya que durante el proceso de creación del contenedor si no encuentra la imagen que necesita para crear éste contenedor Docker mismo se encargará de bajarla.

Creando el contenedor

Para crear el contenedor también tenemos dos opciones:

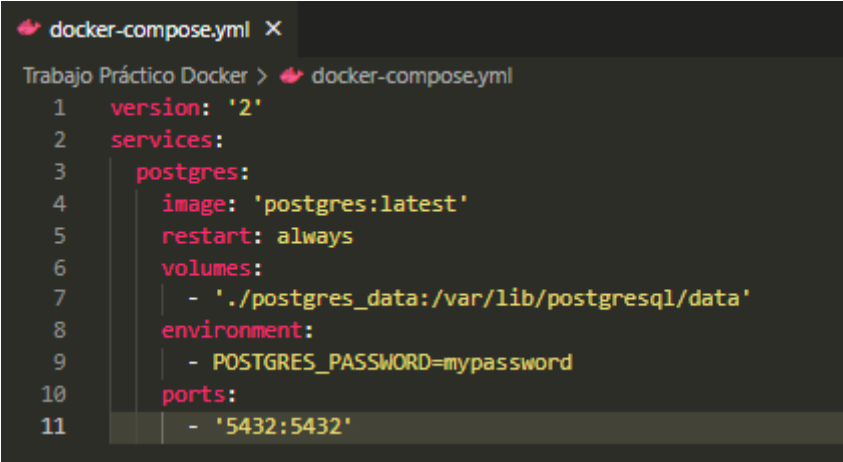
1. una sería con el siguiente comando:

```
docker run --name some-postgres -e POSTGRES_PASSWORD=mypassword -p 5432:5432 -d postgres
```

En este comando podemos ver que estamos estableciendo un password y un puerto, esto es necesario para que podamos acceder al contenedor para trabajar con postgres.

2. Otra opción (que es la que se usará, ya que es la más aplicable a una situación real) es la de utilizar *docker compose*. Para lo cual se necesita un archivo llamado *docker-compose.yml* que contendrá todas las configuraciones que necesitemos para adaptar el contenedor a nuestras necesidades.

Configuramos el archivo *docker-compose.yml* según nuestras necesidades. En este caso usaremos la última *versión* (latest) porque no estamos realizando esto en una producción real, en ese caso tendríamos que trabajar con una versión que sea estable y con la que no hayamos tenido problemas. También vemos que en el apartado de *volumes*, la ruta está haciendo referencia con el punto a nuestro directorio actual, es por eso que debemos tener en cuenta a la hora de levantar el *docker-compose.yml* en dónde estaremos posicionados, ésta ruta será la que dará persistencia a los datos de la base de datos, por fuera del contenedor, en nuestro disco en la carpeta *postgres_data*. A continuación se setean el *password* de la base de datos y el *puerto* de entrada y salida (de izquierda a derecha, respectivamente). Aclaración, para realizar esto es importante que no tengamos otro programa o aplicación trabajando en el mismo puerto de entrada, de ser el caso se deberá poner otro puerto.



```
docker-compose.yml X
Trabajo Práctico Docker > docker-compose.yml
1  version: '2'
2  services:
3    postgres:
4      image: 'postgres:latest'
5      restart: always
6      volumes:
7        - './postgres_data:/var/lib/postgresql/data'
8      environment:
9        - POSTGRES_PASSWORD=mypassword
10     ports:
11       - '5432:5432'
```

A continuación, con Docker ya iniciado, abrimos una terminal o un PowerShell y nos posicionamos en la carpeta en la que queremos trabajar y en la cual le vamos a dar persistencia a nuestra base de datos. Vemos que tenemos solo un directorio y nuestro archivo *docker-compose.yml*.

```
PS D:\Análisis y Programación\3er año\Gestión de Bases de Datos\Trabajo Práctico Docker> ls

Directorio: D:\Análisis y Programación\3er año\Gestión de Bases de Datos\Trabajo Práctico Docker
```

Mode	LastWriteTime	Length	Name
d-----	28/10/2021 20:48		Documentación
-a----	29/10/2021 19:11	228	docker-compose.yml

Ejecutamos el comando *docker-compose up*, en caso de no contar con la imagen para la creación del contenedor ésta se descargará y luego levantará el contenedor según la configuración de nuestro archivo *docker-compose.yml*.

```
PS D:\Análisis y Programación\3er año\Gestión de Bases de Datos\Trabajo Práctico Docker> docker-compose up -d
Creating trabajoprcticodocker_postgres_1 ... done
PS D:\Análisis y Programación\3er año\Gestión de Bases de Datos\Trabajo Práctico Docker> ls

Directorio: D:\Análisis y Programación\3er año\Gestión de Bases de Datos\Trabajo Práctico Docker
```

Mode	LastWriteTime	Length	Name
d-----	28/10/2021 20:48		Documentación
da----	29/10/2021 19:24		postgres_data
-a----	29/10/2021 19:11	228	docker-compose.yml

Vemos también que, como le hemos indicado, se nos ha creado la carpeta *postgres_data*, la cual nos permitirá la persistencia de nuestros datos.

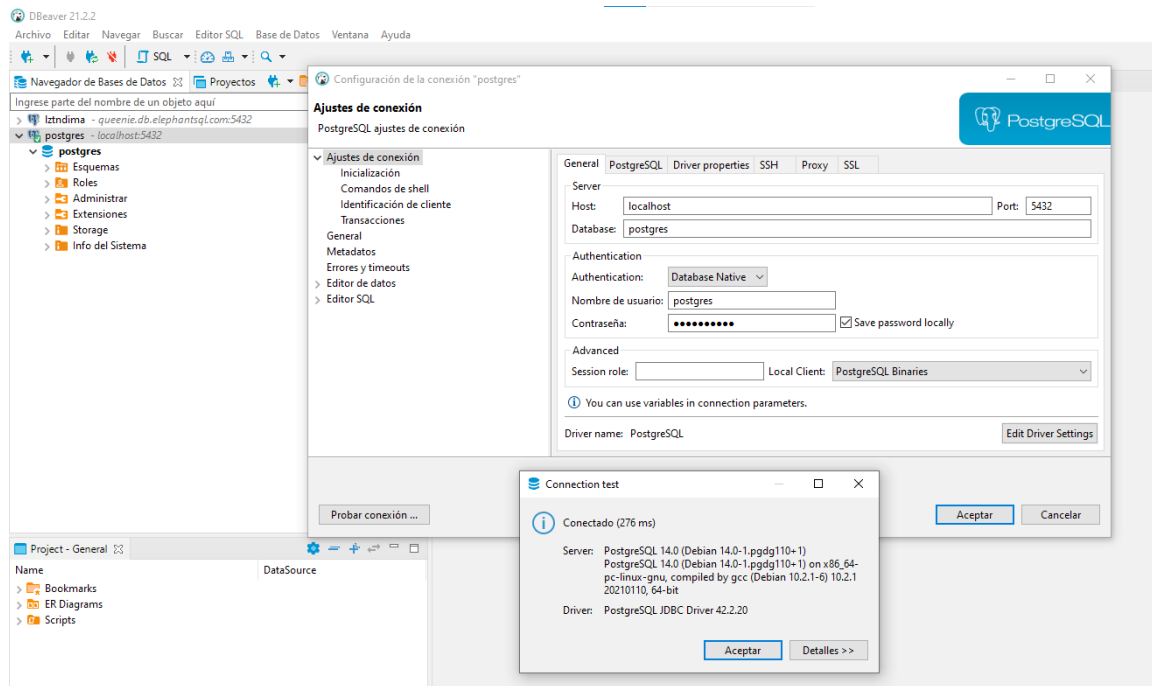
Con el comando *docker ps -a* listamos todos los contenedores que tenemos y vemos su estado, si está corriendo (*status up*) o no (*status exited 0*), mientras que si se produce un error en el contenedor también nos lo informaría (*status exited 1*). En este caso vemos que ya tenemos nuestro contenedor postgres y está corriendo en el puerto 5432 utilizando el protocolo TCP.

```
PS D:\Análisis y Programación\3er año\Gestión de Bases de Datos\Trabajo Práctico Docker> docker ps -a
```

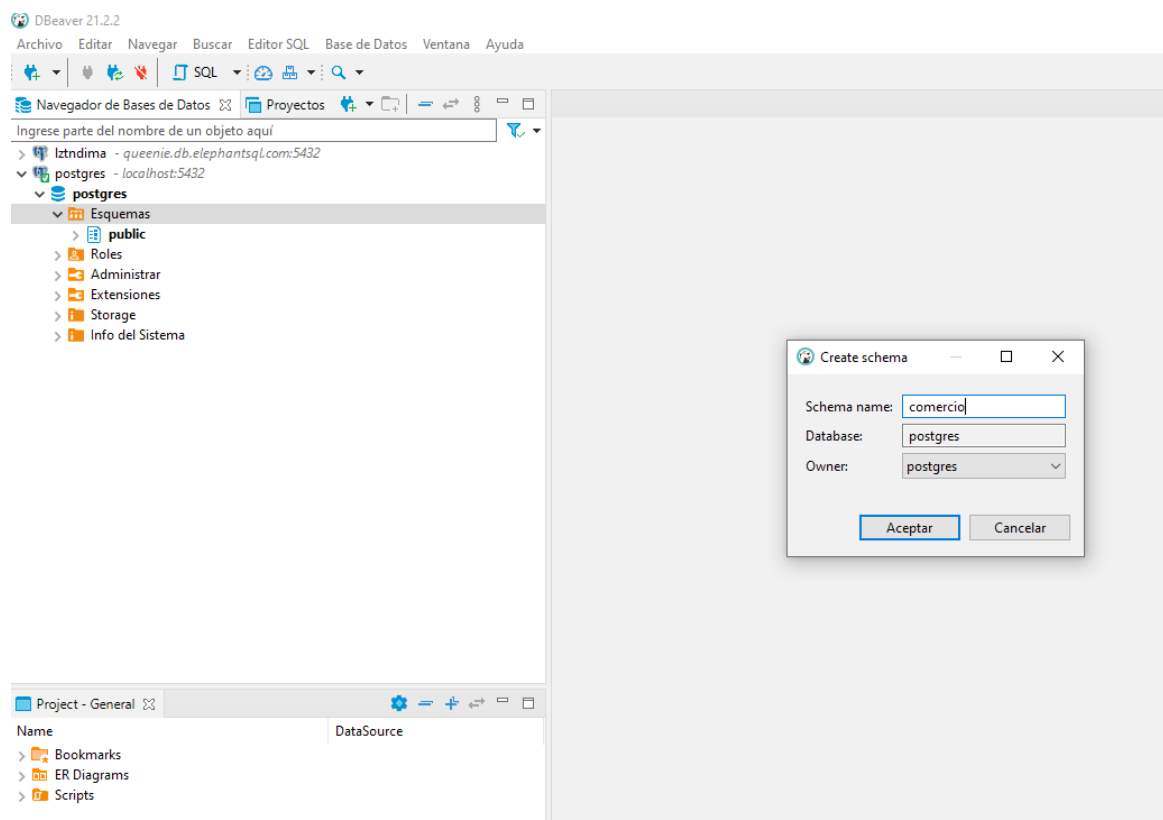
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bdfe6810fce3	postgres:latest	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:5432->5432/tcp	trabajoprcticodocker_postgres_1

Trabajando con DBeaver

En nuestro gestor DBeaver generamos una *nueva conexión* haciendo click en el ícono de enchufe de arriba a la izquierda en la barra de herramientas, seleccionamos PostgreSQL, completamos con nuestros datos y vemos que se ha establecido la conexión correctamente.

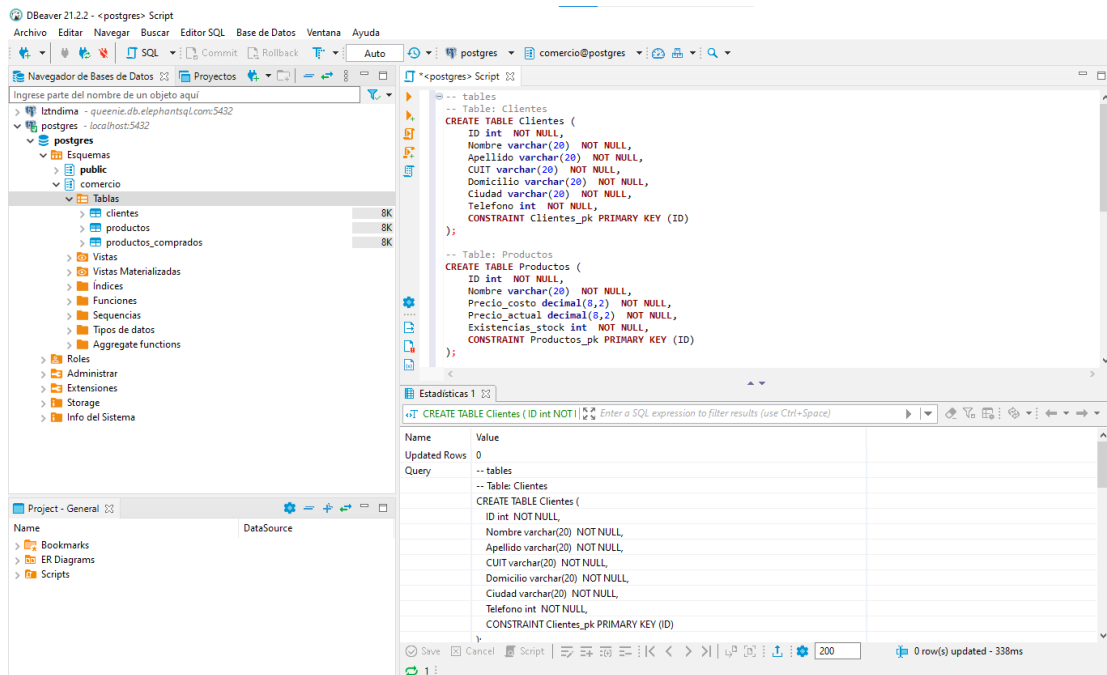


Creamos un *nuevo esquema* (o *schema*), nos paramos dentro de nuestra base de datos en la carpeta esquema, damos click derecho y en generar un *nuevo esquema*.

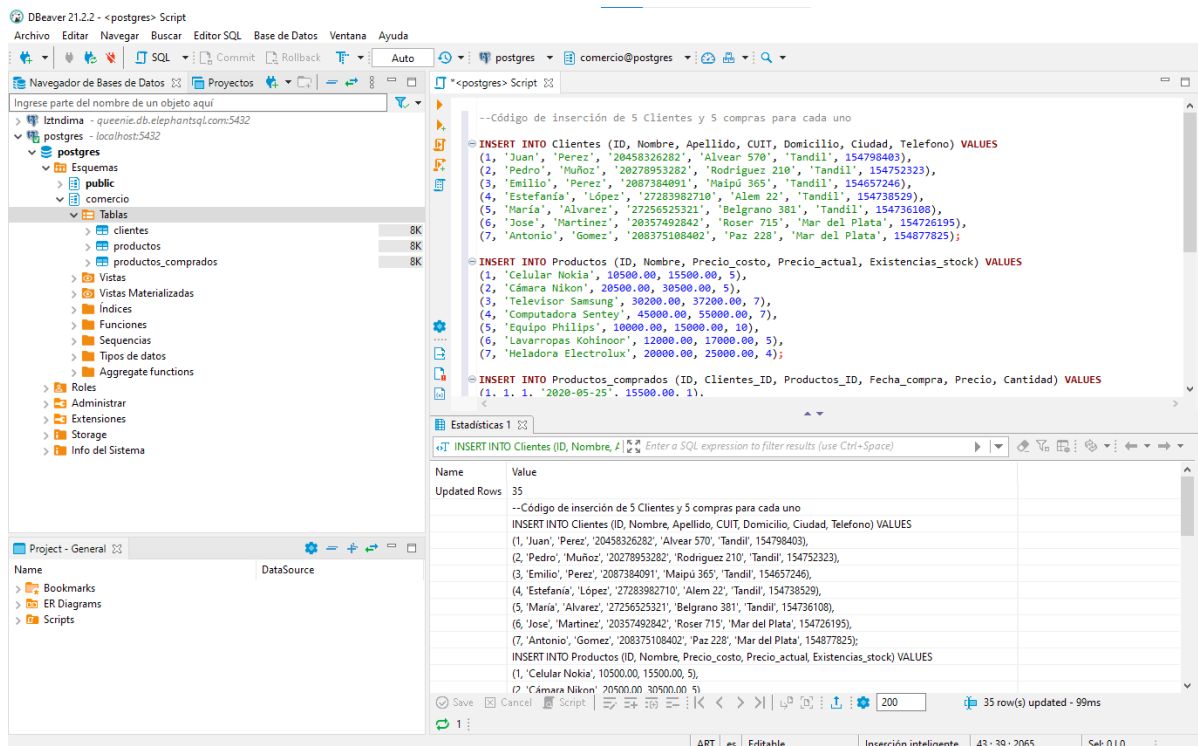


En este punto del trabajo se trabajará con los scripts que serán adjuntados en la entrega del mismo.

Creación de las tablas



Inserción de los datos



Actualizaciones de datos

Actualización de cambio de nombre de un cliente.

The screenshot shows a PostgreSQL script editor with the following SQL code:

```
--Mostramos el dato por pantalla
select Nombre
from Clientes
where ID = 1;

--Codigo de actualización donde un usuario cambia de Nombre
UPDATE Clientes
SET Nombre = 'Andrés'
WHERE ID = 1;

--Mostramos el dato modificado por pantalla
select Nombre
from Clientes
where ID = 1;
```

Below the script editor, the results of the first query are displayed in a table:

Grilla	Nombre
1	Juan

The screenshot shows the same PostgreSQL script editor with the same SQL code. The results of the second query are displayed in a table:

Grilla	Nombre
1	Andrés

Actualización donde un comprador agrega un nuevo producto.

The screenshot shows the PostgreSQL Script Editor with the following SQL code:

```
--Mostramos el dato por pantalla
select Cantidad
from Productos_comprados
where ID = 1;

--Codigo de actualizacion donde un comprador agrega 1 nuevo producto
UPDATE Productos_comprados
SET Cantidad = Cantidad + 1
WHERE ID = 1;

--Mostramos el dato modificado por pantalla
select Cantidad
from Productos_comprados
where ID = 1;
```

Below the script editor, the data grid for 'productos_comprados 1' is displayed. The grid has two columns: 'cantidad' and 'ID'. The first row shows a value of 123 for 'cantidad' and 1 for 'ID'.

cantidad	ID
123	1

The screenshot shows the PostgreSQL Script Editor with the same SQL code as the previous image. The data grid for 'productos_comprados 1' is updated, showing a value of 124 for 'cantidad' and 1 for 'ID'.

cantidad	ID
124	1

Actualización del stock de un producto.

The screenshot shows a PostgreSQL script editor with the following SQL code:

```
--Mostramos el dato por pantalla
select Nombre, Existencias_stock
from Productos
where ID = 3;

--Codigo de actualizacion donde se agregan 10 unidades de un producto mas al stock

UPDATE Productos
SET Existencias_stock = Existencias_stock + 10
WHERE ID = 3;

--Mostramos el dato modificado por pantalla
select Nombre, Existencias_stock
from Productos
where ID = 3;
```

Below the script editor, a table view displays the results of the query. The table has two columns: 'Nombre' and 'Existencias_stock'. The first row shows 'Televisor Samsung' with a stock value of 7.

Grilla	Nombre	Existencias_stock
1	Televisor Samsung	7

The screenshot shows the same PostgreSQL script editor with the same SQL code as the first image. The table view below shows the results of the query after the update. The stock value for 'Televisor Samsung' has increased from 7 to 17.

Grilla	Nombre	Existencias_stock
1	Televisor Samsung	17

Actualización del precio de costo de un producto.

The screenshot shows a PostgreSQL script editor with the following SQL code:

```
--Mostramos el dato por pantalla
select Nombre, Precio_costo
from Productos
where ID = 5;

--Codigo de actualizacion donde el costo de un producto aumente un 10%
UPDATE Productos
SET Precio_costo = Precio_costo + (Precio_costo * 0.1)
WHERE ID = 5;

--Mostramos el dato modificado por pantalla
select Nombre, Precio_costo
from Productos
where ID = 5;
```

Below the script editor, a table view shows the results of the query. The table has two columns: 'Nombre' and 'Precio_costo'. The first row shows 'Equipo Philips' with a 'Precio_costo' of 10.000.

Nombre	Precio_costo
Equipo Philips	10.000

The screenshot shows the same PostgreSQL script editor with the same SQL code as the first screenshot. Below the script editor, a table view shows the results of the query. The table has two columns: 'Nombre' and 'Precio_costo'. The first row shows 'Equipo Philips' with a 'Precio_costo' of 11.000.

Nombre	Precio_costo
Equipo Philips	11.000

Actualización del precio de venta actual de los productos, aplicando un descuento del 10% a los que están por encima del valor de 35000.

The screenshot shows a PostgreSQL script editor with the following SQL code:

```
--Mostramos los datos por pantalla
select Nombre, Precio_actual
from Productos
where Precio_actual >= 35000;

--Codigo de actualizacion donde todos los productos con un precio a la venta mayor o igual a 35000 pesos tengan un
UPDATE Productos
SET Precio_actual = Precio_actual - (Precio_actual * 0.1)
WHERE Precio_actual >= 35000;

--Mostramos los datos modificados por pantalla
select Nombre, Precio_actual
from Productos
where Precio_actual >= 35000;
```

Below the script editor, the query results are displayed in a table:

ABC	nombre	123	precio_actual
1	Computadora Sentey		55.000
2	Televisor Samsung		37.200

The interface also includes a sidebar with icons for various database operations and a panel on the right showing the selected table 'Computadora Sentey'.

The screenshot shows the same PostgreSQL script editor with the same SQL code as above. The query results are now updated to reflect the 10% discount applied to products with prices of 35000 or more:

ABC	nombre	123	precio_actual
1	Computadora Sentey		49.500

The 'Televisor Samsung' product is no longer visible in the results table as its price (37.200) is below the 35000 threshold.

Consultas

Top 5 de compras de mayor precio.

The screenshot shows the PostgreSQL Script editor with the following query:

```
--Cuales son las Top 5 compras de mayor precio  
  
SELECT ID, Clientes_ID, Productos_ID, (Precio * Cantidad) as Total_de_venta  
FROM Productos_comprados  
ORDER BY Total_de_venta DESC  
LIMIT 5;
```

Below the editor, the table view for 'productos_comprados 1' is displayed, showing the top 5 purchases by total price:

	id	clientes_id	productos_id	total_de_venta
1	21	7	3	148.800
2	5	2	4	110.000
3	13	5	3	74.400
4	3	1	3	74.400
5	8	3	3	74.400

Top 5 de compras de menor precio.

The screenshot shows the PostgreSQL Script editor with the following query:

```
--Cuales son las Top 5 compras de menor precio  
  
SELECT ID, Clientes_ID, Productos_ID, (Precio * Cantidad) as Total_de_venta  
FROM Productos_comprados  
ORDER BY Total_de_venta ASC  
LIMIT 5;
```

Below the editor, the table view for 'productos_comprados 1' is displayed, showing the top 5 purchases by total price in ascending order:

	id	clientes_id	productos_id	total_de_venta
1	15	5	5	15.000
2	20	7	7	25.000
3	17	6	7	25.000
4	6	2	5	30.000
5	12	5	2	30.500

Promedio de compra por cliente.

The screenshot shows the PostgreSQL Script editor with the following SQL query:

```
--Cual es el promedio de compras por cliente  
  
SELECT Clientes_ID, AVG(cantidad) AS Promedio_compras  
FROM Productos_comprados  
GROUP BY Clientes_ID;
```

The query results are displayed in a table with 7 rows:

	clientes_id	promedio_compras
1	3	2,5
2	5	1,25
3	4	1,666666667
4	6	1,666666667
5	2	1,666666667
6	7	1,666666667
7	1	2

On the right, a dictionary for 'clientes' is shown:

Value	Description
1	Perez
2	Muñoz
3	Perez
4	López

Total de compras por ciudad.

The screenshot shows the PostgreSQL Script editor with the following SQL query:

```
--Cual es el total de compras por ciudad  
  
SELECT Clientes.Ciudad, SUM(Productos_comprados.Cantidad) AS Total_de_compras  
FROM Clientes  
JOIN Productos_comprados ON Clientes.ID = Productos_comprados.Clientes_ID  
GROUP BY Clientes.Ciudad;
```

The query results are displayed in a table with 2 rows:

	ciudad	total_de_compras
1	Tandil	26
2	Mar del Plata	10

On the right, a dictionary for 'clientes' is shown:

Value	Description
Tandil	

Borrado de datos

Borrado de un cliente.

The screenshot shows a PostgreSQL client window with a script editor and a table view. The script contains the following SQL commands:

```
--Mostramos los datos por pantalla
select ID, Nombre, Apellido
from Clientes;

--Borrado de un cliente
DELETE FROM Clientes
WHERE ID = 3;

--Mostramos los datos nuevamente por pantalla
select ID, Nombre, Apellido
from Clientes;
```

The table view shows the 'clientes' table with the following data:

id	nombre	apellido
1	Pedro	Muñoz
2	Emilio	Perez
3	Estefanía	López
4	María	Alvarez
5	Jose	Martinez
6	Antonio	Gomez
7	Andrés	Perez

The screenshot shows the same PostgreSQL client window after executing the script. The 'DELETE FROM Clientes WHERE ID = 3;' statement has been executed, and the table view now shows the updated data:

id	nombre	apellido
1	Pedro	Muñoz
2	Emilio	Perez
3	Estefanía	López
4	María	Alvarez
5	Jose	Martinez
6	Antonio	Gomez
7	Andrés	Perez

Borrado de dos productos.

The screenshot shows the PostgreSQL Script editor with the following SQL script:

```
--Mostramos los datos por pantalla
select ID, Nombre
from Productos;

--Borrado de dos productos
DELETE FROM Productos
WHERE ID = 1 OR ID = 3;

--Mostramos los datos nuevamente por pantalla
select ID, Nombre
from Productos;
```

The results table, titled "productos 1 (2)", displays the initial data:

	id	nombre
1	1	Celular Nokia
2	2	Cámara Nikon
3	6	Lavarropas Kohinoor
4	7	Heladora Electrolux
5	5	Equipo Philips
6	4	Computadora Sentey
7	3	Televisor Samsung

The screenshot shows the PostgreSQL Script editor with the same SQL script as above. The results table, titled "productos 1 (2)", displays the data after deleting products with IDs 1 and 3:

	id	nombre
1	2	Cámara Nikon
2	6	Lavarropas Kohin
3	7	Heladora Electrol
4	5	Equipo Philips
5	4	Computadora Ser

Borrado de un producto comprado.

*<postgres> Script

```
--Mostramos los datos por pantalla
select ID, Clientes_ID, Productos_ID, Fecha_compra
from Productos_comprados;

--Borrado de un producto comprado
DELETE FROM Productos_comprados
WHERE ID = 17;

--Mostramos los datos nuevamente por pantalla
select ID, Clientes_ID, Productos_ID, Fecha_compra
from Productos_comprados;
```

productos_comprados 1 | productos_comprados 1 (2)

select ID, Clientes_ID, Productos_ID, Fecha_compra

id	clientes_id	productos_id	fecha_compra
1	2	1	2020-08-27
2	5	2	2020-09-07
3	6	2	2021-01-03
4	10	4	2020-09-26
5	12	5	2020-08-04
6	14	5	2021-01-03
7	15	5	2021-01-06
8	16	6	2021-10-17
9	17	6	2021-10-17
10	20	7	2021-10-19
11	9	4	[NULL]
12	18	6	[NULL]
13	10	7	[NULL]

20 row(s) fetched - 90ms

*<postgres> Script

```
--Mostramos los datos por pantalla
select ID, Clientes_ID, Productos_ID, Fecha_compra
from Productos_comprados;

--Borrado de un producto comprado
DELETE FROM Productos_comprados
WHERE ID = 17;

--Mostramos los datos nuevamente por pantalla
select ID, Clientes_ID, Productos_ID, Fecha_compra
from Productos_comprados;
```

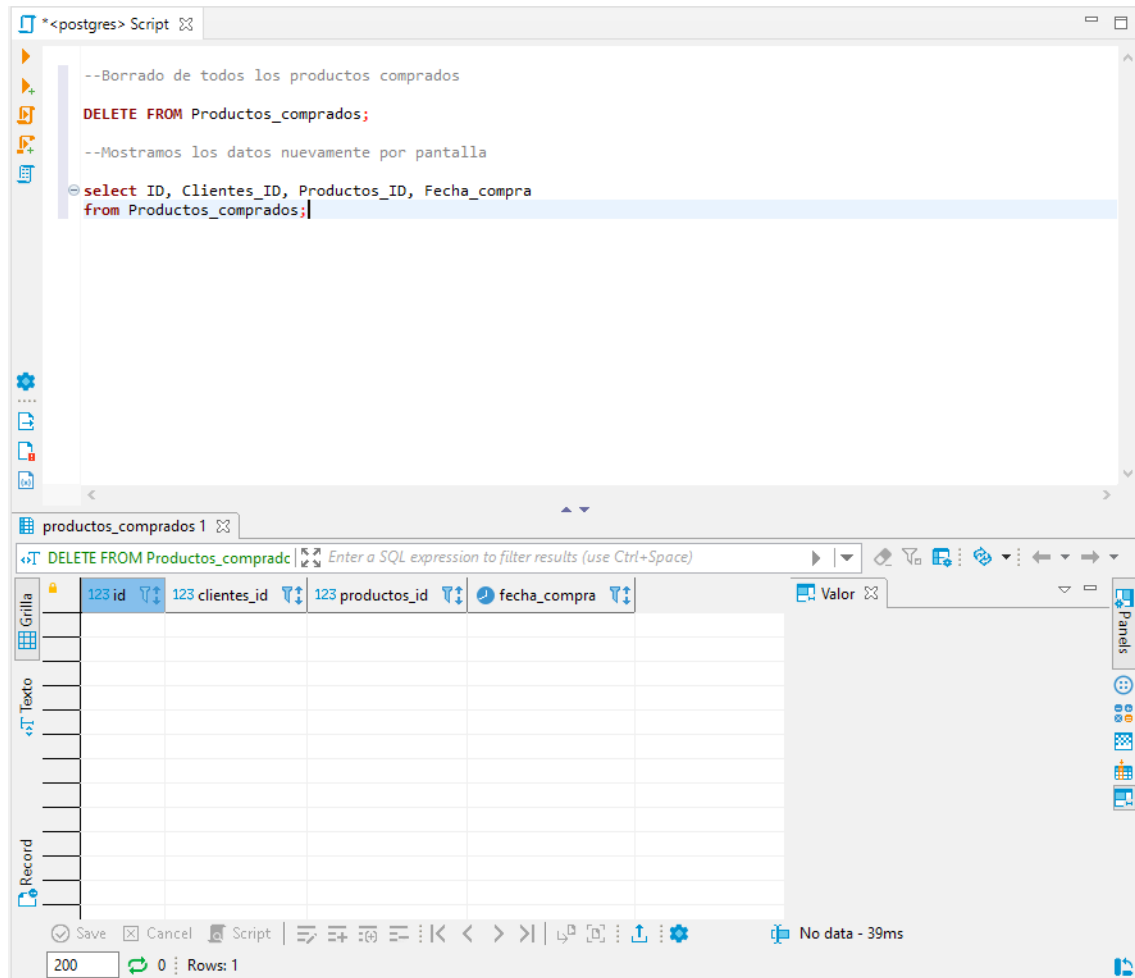
productos_comprados 1 | productos_comprados 1 (2)

select ID, Clientes_ID, Productos_ID, Fecha_compra

id	clientes_id	productos_id	fecha_compra
1	2	1	2020-08-27
2	5	2	2020-09-07
3	6	2	2021-01-03
4	10	4	2020-09-26
5	12	5	2020-08-04
6	14	5	2021-01-03
7	15	5	2021-01-06
8	16	6	2021-10-17
9	20	7	2021-10-19
10	9	4	[NULL]
11	18	6	[NULL]
12	19	7	[NULL]
13	1	1	[NULL]

20 row(s) fetched

Borrado de todos los productos comprados.



Finalización

Para finalizar, cerramos nuestro gestor DBeaver, y en PowerShell con el comando `docker ps -a` conseguimos el identificador del contenedor (*container id*) que vemos que está corriendo. A continuación ejecutamos el comando `docker stop [container id]` para detener el contenedor.

```
PS C:\Users\Yacko> docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED    STATUS    PORTS                               NAMES
6dfe6810fce3   postgres:latest "docker-entrypoint.s..." 2 days ago Up 2 hours 0.0.0.0:5432->5432/tcp      trabajoprcticodocker_postgres_1
PS C:\Users\Yacko> docker stop 6dfe6810fce3
6dfe6810fce3
PS C:\Users\Yacko>
```

En un sistema Windows o Mac podríamos cerrar directamente Docker y se detendrían los contenedores, pero esto no sucede en Linux, por lo tanto en una buena práctica la de detener contenedores que ya no vayamos a utilizar más. Para finalizar ya podemos cerrar Docker.