



Sustentación Proyecto Sportswear Store

Natalia Moreno

Valentina Bueno

Juan Andrés López

David Cajiao

GitHub - DCajiao/sportswear_store: Design, development and implementation of database and back-end for the management of an ecommerce
Design, development and implementation of database and back-end for the management of an ecommerce of a custom sportswear store called "Sportswear Stor

https://github.com/DCajiao/sportswear_store

Correcciones

[Evidencias de Correcciones](#)

¿Cómo se implementó la funcionalidad de Compra?

1. Operaciones CRUD en las tablas [Compra](#) [Contiene](#) [Envio](#) [Lugar](#) [Persona](#) [Producto](#)

2. Validaciones de correcto funcionamiento a nivel de la Base de datos (Objetos Almacenados)

El problema que queríamos evitar con nuestra base de datos era la falta de garantías sobre la integridad y consistencia de los datos almacenados. Esto podría manifestarse en inserciones o modificaciones que violan las restricciones de integridad, como claves primarias duplicadas o valores fuera del rango permitido. La importancia de corregir este problema radica en mantener la confiabilidad y precisión de la información, evitando inconsistencias que podrían afectar la toma de decisiones basadas en los datos.

La decisión de implementar nuestras funcionalidades desde la base de datos fue la más alineada con las mejores prácticas por varias razones.

- En primer lugar, al aprovechar objetos almacenados de SQL, podemos establecer reglas y restricciones a nivel de base de datos, lo que garantiza que los datos se adhieran a las políticas establecidas independientemente de la

fuente de ingreso. Esto promueve la coherencia y uniformidad en la gestión de datos en todo el sistema.

- Además, al asegurar que nuestra base de datos cumpla con las características *ACID*, estamos garantizando la atomicidad, consistencia, aislamiento y durabilidad de las transacciones, lo que es fundamental para la integridad de los datos en entornos críticos.

3. Gestión automática de Stock en `Producto` y de ValorTotal en `Compra`

- Esta implementación garantiza el funcionamiento mediante los triggers ya que realiza actualizaciones apropiadas al instante en el que se crean registros en la base de datos

¿Dónde implementaremos NoSQL y por qué?

1. Entidad `Reseña`

- Hemos decidido crear las reseñas con relación a los productos que ofrece la tienda y a las cuentas que realizan esas reseñas. Adicional, se debe implementar la posibilidad de hacer replicas a esas reseñas, tanto los usuarios de tipo cliente como de administradores. Para ello, se debe implementar en una no relacional ([MongoDB](#))

2. Relación reflexiva en `Producto` para los diseños personalizados

- Dentro de la tabla Producto tenemos almacenada información tanto de Artículos como de Diseños, para poder garantizar el correcto funcionamiento y almacenado de la información, Esta tabla debe ser creada en un formato no relacional para poder realizar documentos embebidos de forma correcta y poder registrar en `Contiene`, qué diseños comprados aplican sobre qué artículos comprados.

¿Cómo probar la funcionalidad del Backend

1. `CREATE` un registro de compra ([POST](#))

```
{  
    "id_compra": 11,  
    "fecha_compra": "2022-05-01T05:00:00.000+00:00",  
    "cuenta_Usuario": {  
        "tipo": "Cliente",  
        "contraseña": "D14z!",  
        "usuario": "Diego_el_master"  
    },  
    "costo_total": 0  
}
```

2. `READ` a Producto ([GET](#))

- Esto para obtener el listado de productos disponibles y poder visualizar el ID del producto que deseamos elegir

3. `CREATE` un registro de contiene ([POST](#))

```
{  
    "id_contiene": 11,  
    "cantidad": 1,  
    "compra_ID_compra": {  
        "id_compra": 11,  
        "fecha_compra": "2022-05-01T05:00:00.000+00:00",  
        "cuenta_Usuario": {  
            "tipo": "Cliente",  
            "contraseña": "D14z!",  
            "usuario": "Diego_el_master"  
        },  
        "costo_total": 0  
    }  
}
```

```

        "costo_total": 0
    },
    "especificaciones": [{"Talla": "L", "Cantidad": 1}],
    "producto_Identificacion": {
        "precio": 50.0,
        "esPaquete": false,
        "tipo_producto": "Articulo",
        "imagen_producto": "imagen1.jpg",
        "identificacion": 1,
        "tipo_Articulo": "Camisa",
        "especificaciones_Articulo": null,
        "cantidad_Articulo": 96,
        "productos_paquete": null,
        "seccion_Articulo": "Hombre"
    }
}

```

4. **CREATE** un registro de envío ([POST](#))

```
{
    "id_envio": 11,
    "compra_ID_compra": {
        "id_compra": 11,
        "costo_total": 50,
        "cuenta_Usuario": {
            "tipo": "Cliente",
            "contraseña": "D14z!",
            "usuario": "Diego_el_master"
        },
        "fecha_compra": "2022-05-01T05:00:00.000+00:00"
    },
    "direccion": "Calle 1 #123",
    "codigo_postal": 12345,
    "lugar_Codigo": {
        "nombre": "Bogotá",
        "codigo": 1
    },
    "fecha_envio": "2024-05-01T05:00:00.000+00:00"
}
```

El UPDATE en envío tiene esta misma forma y los campos que se pueden actualizar son: Direccion, Fecha_envio, Código_postal, Compra_ID_compra y Lugar_Código.

5. En caso de querer cancelar la compra. Se deben ejecutar los siguientes [DELETE](#)

```
DELETE envio
URL localhost:8080/compra/{ID del envío a borrar}
```

```
DELETE compra
localhost:8080/compra/{ID de compra a borrar}
```

```
DELETE contiene
URL localhost:8080/contiene/{ID del contiene a borrar}
```

▼ Operaciones adicionales disponibles

UPDATE a compra

Al final de "http://localhost:8080/compra/" se debe agregar el id de la compra que se desea actualizar

```
{
  "id_compra": 11,
  "fecha_compra": "2022-05-01T05:00:00.000+00:00",
  "cuenta_Usuario": {
    "tipo": "Cliente",
    "contraseña": "Per3z#1024",
    "usuario": "juan_perez"
  },
  "costo_total": 120.20
}
```

En compra solo se van a actualizar los campos de Costo_total, Fecha_compra y Cuenta_Usuario

Objetos Almacenados

▼ Trigger

▼ Validar la existencia de un usuario antes de registrar una compra

▼ Código

```
-- Trigger para validar la existencia de un usuario antes de registrar una compra
DELIMITER //
CREATE TRIGGER validar_usuario_existente
BEFORE INSERT ON Compra
FOR EACH ROW
BEGIN
  DECLARE usuario_valido INT;

  -- Verificar si el usuario especificado existe en la tabla Cuenta
  SELECT COUNT(*) INTO usuario_valido
  FROM Cuenta
  WHERE Usuario = NEW.Cuenta_Usuario;

  -- Si el usuario no existe, emitir un mensaje de error
  IF usuario_valido = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'El usuario especificado no existe';
  END IF;
END;
//DELIMITER ;
```

▼ Test

```
-- Validación de existencia de Usuario
INSERT INTO Compra (`Costo_total`, `Fecha_compra`, `Cuenta_Usuario`)
VALUES (150.00, '2024-04-17', 'usuario_no_existente');
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following code:

```
1 •  INSERT INTO Compra (`Costo_total`, `Fecha_compra`, `Cuenta_Usuario`)
2   VALUES (150.00, '2024-04-17', 'usuario_no_existente');
```

The screenshot shows the MySQL Workbench interface with the output pane open. It displays the following error message:

Action Output	Message
# 1 03:39:45	INSERT INTO Compra ('Costo_total', 'Fecha_compra', 'Cuenta_Usuario') VALUES (150.00, '2024-04-17', 'usu... Error Code: 1644. El usuario especificado no existe

The screenshot shows the MySQL Workbench interface. The SQL editor contains the same code as before:

```
1 •  INSERT INTO Compra (`Costo_total`, `Fecha_compra`, `Cuenta_Usuario`)
2   VALUES (150.00, '2024-04-17', 'Lauris');
```

The screenshot shows the MySQL Workbench interface with the output pane open. It displays the following success message:

Action Output	Message
<input checked="" type="checkbox"/> 1 03:41:24	INSERT INTO Compra ('Costo_total', 'Fecha_compra', 'Cuenta_Usuario') VALUES (150.00, '2024-04-17', 'La... 1 row(s) affected

▼ Validar la disponibilidad antes de registrar qué contiene una compra

▼ Código

```
-- Trigger para validar la disponibilidad antes de registrar qué contiene una compra
DELIMITER //
CREATE TRIGGER validar_disponibilidad
BEFORE INSERT ON Contiene
FOR EACH ROW
BEGIN
    DECLARE cantidad_disponible INT;

    -- Obtener la cantidad disponible del producto
    SELECT Cantidad_Articulo INTO cantidad_disponible
    FROM Producto
    WHERE Identificacion = NEW.Producto_Identificacion;

    -- Verificar si la cantidad seleccionada es mayor que la cantidad disponible
    IF NEW.Cantidad > cantidad_disponible THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No hay suficiente stock disponible para insertar esta cantidad';
    END IF;
END;
// DELIMITER ;
```

▼ Test

```
-- Validación de disponibilidad antes de registrar qué contiene una compra
INSERT INTO Contiene (`Producto_Identificacion`, `Compra_ID_compra`, `Cantidad`, `Especificaciones`)
VALUES (1, 2, 400, '{"Talla":"M", "Cantidad":1});
```

The screenshot shows the MySQL Workbench interface with a query editor and a results pane.

Query Editor:

```
1 -- Validación de disponibilidad antes de registrar qué contiene una compra
2 • INSERT INTO Contiene ('Producto_Identificacion', 'Compra_ID_compra', 'Cantidad', 'Especificaciones') VALUES
3   (1, 2, 400, '{"Talla":"M", "Cantidad":1});
```

Results Pane:

#	Time	Action	Message
1	03:42:26	INSERT INTO Contiene ('Producto_Identificacion', 'Compra_ID_compra', 'Cantidad', 'Especificaciones') VAL...	Error Code: 1644. No hay suficiente stock disponible para insertar esta cantidad de producto

The screenshot shows the MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, a code editor window displays the following SQL code:

```

1 -- Validación de disponibilidad antes de registrar qué contiene una compra
2 • INSERT INTO Contiene (`Producto_Identificacion`, `Compra_ID_compra`, `Cantidad`, `Especificaciones`) VALUES
3 (1, 2, 2, '{"Talla":"M", "Cantidad":1});

```

Below the code editor is an output window titled "Output". It contains a table with the following data:

#	Time	Action	Message
1	03:43:43	INSERT INTO Contiene (Producto_Identificacion, Compra_ID_compra, Cantidad, Especificaciones) VAL...	1 row(s) affected

▼ Actualizar la cantidad disponible de un producto en el inventario después de una compra

▼ Código

```

-- Trigger para actualizar la cantidad disponible de un producto en el inventario después de una compra
DELIMITER //
CREATE TRIGGER ActualizarCantidadDisponible
AFTER INSERT ON Contiene
FOR EACH ROW
BEGIN
    UPDATE Producto
    SET Cantidad_Articulo = Cantidad_Articulo - NEW.Cantidad
    WHERE Identificacion = NEW.Producto_Identificacion;
END //
DELIMITER ;

```

▼ Test

```

-- Actualización de la cantidad disponible de un producto en el inventario después de una compra
SELECT * FROM producto;
INSERT INTO Contiene (`Producto_Identificacion`, `Compra_ID_compra`, `Cantidad`, `Especificaciones`) VALUES
(1, 2, 4, '{"Talla":"M", "Cantidad":1});
SELECT * FROM producto;

```

```

1 -- Actualización de la cantidad disponible de un producto en el inventario después de una compra
2 • SELECT * FROM producto;
3 • INSERT INTO Contiene (`Producto_Identificacion`, `Compra_ID_compra`, `Cantidad`, `Especificaciones`) VALUES
4 (1, 2, 4, '{"Talla":"M", "Cantidad":3}');
5 • SELECT * FROM producto;
6

```

Result Grid

Identificacion	Precio	Imagen_producto	Tipo_Producto	Tipo_Articulo	Seccion_Articulo	Cantidad_Articulo	Especificaciones_Articulo	Productos_paquete
1	50.00	BL00	Artículo	Camisa	Hombre	74	[{"Talla": "XS", "Cantidad": 20}, {"Talla": "S", "C..."}, {"Talla": "M", "Cantidad": 3}, {"Talla": "L", "Cantidad": 10}, {"Talla": "XL", "Cantidad": 5}], [{"Talla": "XS", "Cantidad": 20}, {"Talla": "S", "Cantidad": 10}, {"Talla": "M", "Cantidad": 3}, {"Talla": "L", "Cantidad": 10}, {"Talla": "XL", "Cantidad": 5}], 0	NULL
2	35.00	BL00	Artículo	Pantalon	Mujer	77	0	NULL
3	20.00	BL00	Artículo	Gorra	Nino	119	0	NULL
4	45.00	BL00	Diseño	NULL	NULL	88	0	NULL
5	60.00	BL00	Diseño	NULL	NULL	67	0	NULL
6	25.00	BL00	Artículo	Zapatillas	Nino	109	[{"Talla": "20", "Cantidad": 5}, {"Talla": "21", "C..."}, {"Talla": "22", "Cantidad": 10}], [{"Talla": "20", "Cantidad": 5}, {"Talla": "21", "Cantidad": 10}, {"Talla": "22", "Cantidad": 10}], 0	NULL

producto 10 x

Output

Action Output

#	Time	Action	Message
1	03:47:28	SELECT * FROM producto LIMIT 0, 1000	13 row(s) returned


```

1 -- Actualización de la cantidad disponible de un producto en el inventario después de una compra
2 • SELECT * FROM producto;
3 • INSERT INTO Contiene (`Producto_Identificacion`, `Compra_ID_compra`, `Cantidad`, `Especificaciones`) VALUES
4 (1, 2, 4, '{"Talla":"M", "Cantidad":5'});
5 • SELECT * FROM producto;
6

```

Result Grid

Identificacion	Precio	Imagen_producto	Tipo_Producto	Tipo_Articulo	Seccion_Articulo	Cantidad_Articulo	Especificaciones_Articulo	Productos_paquete
1	50.00	BL00	Artículo	Camisa	Hombre	74	[{"Talla": "XS", "Cantidad": 20}, {"Talla": "S", "C..."}, {"Talla": "M", "Cantidad": 5}, {"Talla": "L", "Cantidad": 10}, {"Talla": "XL", "Cantidad": 5}], [{"Talla": "XS", "Cantidad": 20}, {"Talla": "S", "Cantidad": 10}, {"Talla": "M", "Cantidad": 5}, {"Talla": "L", "Cantidad": 10}, {"Talla": "XL", "Cantidad": 5}], 0	NULL
2	35.00	BL00	Artículo	Pantalon	Mujer	77	0	NULL
3	20.00	BL00	Artículo	Gorra	Nino	119	0	NULL
4	45.00	BL00	Diseño	NULL	NULL	88	0	NULL
5	60.00	BL00	Diseño	NULL	NULL	67	0	NULL
6	25.00	BL00	Artículo	Zapatillas	Nino	109	[{"Talla": "20", "Cantidad": 5}, {"Talla": "21", "C..."}, {"Talla": "22", "Cantidad": 10}], [{"Talla": "20", "Cantidad": 5}, {"Talla": "21", "Cantidad": 10}, {"Talla": "22", "Cantidad": 10}], 0	NULL

producto 11 x

Output

Action Output

#	Time	Action	Message
1	03:47:28	SELECT * FROM producto LIMIT 0, 1000	13 row(s) returned
2	03:48:07	SELECT * FROM producto LIMIT 0, 1000	13 row(s) returned

▼ Actualizar el valor de la compra después de insertar un nuevo registro en la tabla Contiene

▼ Código

```

-- Trigger para actualizar el valor de la compra después de insertar un nuevo registro en la tabla Contiene
DELIMITER //
CREATE TRIGGER actualizar_costo_total_despues_insertar
AFTER INSERT ON Contiene
FOR EACH ROW
BEGIN
    DECLARE total DECIMAL(10,2);

    -- Calcular el costo total de la compra
    SELECT SUM(p.Precio * c.Cantidad) INTO total
    FROM Producto p
    JOIN Contiene c ON p.Identificacion = c.Producto_Identificacion
    WHERE c.Compra_ID_compra = NEW.Compra_ID_compra;

```

```
-- Actualizar el costo total de la compra
UPDATE Compra
SET Costo_total = total
WHERE ID_compra = NEW.Compra_ID_compra;
END;
//DELIMITER ;
```

▼ Test

```
-- Actualizació del costo total de compra
SELECT * FROM Compra; -- Para ver la compra
INSERT INTO Contiene (Producto_Identificacion, Cantidad, Compra_ID_compra) VALUES
(1, 10, 1);
SELECT * FROM Compra; -- Para ver los cambios
```

▼ Funciones

▼ Función para obtener la cantidad de productos disponible en el inventario

▼ Código

```
-- Función para obtener la cantidad de productos disponible en el inventario
DELIMITER //
CREATE FUNCTION ObtenerCantidadDisponible(
    p_identificacion INT
)
RETURNS INT
READS SQL DATA
BEGIN
    DECLARE cantidad_disponible INT;
    SELECT Cantidad INTO cantidad_disponible
    FROM Producto
    WHERE Identificacion = p_identificacion;
    RETURN cantidad_disponible;
END //
DELIMITER ;
```

▼ Test

```
-- ¿Cómo llamar la función?
SELECT ObtenerCantidadDisponible(1) AS CantidadDisponible;
```

▼ Función para calcular el costo total de una compra, teniendo en cuenta la cantidad y el precio de los productos.

▼ Código

```
-- Función para calcular el costo total de una compra, teniendo en cuenta la cantidad y
DELIMITER //
CREATE FUNCTION CalcularCostoTotal(
    p_id_compra INT
)
RETURNS DECIMAL(10,2)
READS SQL DATA
```

```

BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT SUM(p.Precio * c.Cantidad) INTO total
    FROM Producto p
    JOIN Contiene c ON p.Identificacion = c.Producto_Identificacion
    WHERE c.Compra_ID_compra = p_id_compra;
    RETURN total;
END //
DELIMITER ;

```

▼ Test

```

-- ¿Cómo llamar la función?
SELECT CalcularCostoTotal(3) AS CostoTotalCompra;

```

▼ Función para calcular el total de ingresos obtenidos en un período de tiempo específico

▼ Código

```

-- Función para calcular el total de ingresos obtenidos en un período de tiempo específico
DELIMITER //
CREATE FUNCTION CalcularTotalIngresos(
    p_fecha_inicio DATE,
    p_fecha_fin DATE
)
RETURNS DECIMAL(10,2)
READS SQL DATA
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT SUM(Costo_total) INTO total
    FROM Compra
    WHERE Fecha_compra BETWEEN p_fecha_inicio AND p_fecha_fin;
    RETURN total;
END //
DELIMITER ;

```

▼ Test

```

-- ¿Cómo llamar la función?
SELECT CalcularTotalIngresos('2024-03-01', '2024-03-10') AS TotalIngresos;

```

▼ Vistas

▼ Vista de los productos disponibles junto con su cantidad en stock

▼ Código

```

-- Vista de los productos disponibles junto con su cantidad en stock
CREATE VIEW ProductosDisponibles AS
SELECT Identificacion, Precio, Tipo_Producto, Tipo_Articulo, Seccion_Articulo, Cantidad
FROM Producto
WHERE Cantidad_Articulo > 0;

```

▼ Test

```
-- ¿Cómo llamar la vista?
SELECT * FROM ProductosDisponibles;
-- Otra forma de llamar la vista
SELECT * FROM ProductosDisponibles WHERE Tipo = 'Articulo' AND Seccion = 'Hombre';
```

The screenshot shows the MySQL Workbench interface. At the top, there are four numbered lines of SQL code:

- ¿Cómo llamar la vista?
- SELECT * FROM ProductosDisponibles;
- Otra forma de llamar la vista
- SELECT * FROM ProductosDisponibles WHERE Tipo = 'Articulo' AND Seccion = 'Hombre';

Below the code, the "Result Grid" pane displays a table with 10 rows of data. The columns are: Identificacion, Precio, Tipo_Producto, Tipo_Articulo, Seccion_Articulo, and Cantidad_Articulo. The data is as follows:

Identificacion	Precio	Tipo_Producto	Tipo_Articulo	Seccion_Articulo	Cantidad_Articulo
1	50.00	Articulo	Camisa	Hombre	44
2	35.00	Articulo	Pantalon	Mujer	77
3	20.00	Articulo	Gorra	Nino	119
4	45.00	Diseño	NULL	NULL	88
5	60.00	Diseño	NULL	NULL	67
6	25.00	Articulo	Zapatillas	Nino	109
7	40.00	Articulo	Polo	Hombre	93

At the bottom of the interface, the "Output" pane shows two log entries:

- Action Output: # 1 03:58:05 SELECT * FROM ProductosDisponibles LIMIT 0, 1000
- Message: 10 row(s) returned

This screenshot is identical to the one above, showing the execution of the same four SQL statements and displaying the same 10 rows of product data in the results grid. The output pane also shows the same two log entries.

▼ Vista de los productos con el conteo de ventas

▼ Código

```
-- Vista de los productos con el conteo de ventas
CREATE VIEW VentasProducto AS
SELECT p.Identificacion AS ID_Producto,
       p.Tipo_Producto,
       p.Tipo_Articulo,
       COUNT(c.Producto_Identificacion) AS Cantidad_Ventas
```

```

FROM Producto p
LEFT JOIN Contiene c ON p.Identificacion = c.Producto_Identificacion
GROUP BY p.Identificacion;

```

▼ Test

```

-- ¿Cómo llamar la vista?
SELECT * FROM VentasProducto;

```

The screenshot shows a database interface with a query editor and a results grid.

Query Editor:

```

1 -- ¿Cómo llamar la vista?
2 SELECT * FROM VentasProducto;

```

Results Grid:

ID_Producto	Tipo_Producto	Tipo_Articulo	Cantidad_Ventas
1	Articulo	Camisa	11
2	Articulo	Pantalon	1
3	Articulo	Gorra	1
4	Diseño	HULL	1
5	Diseño	HULL	1
6	Articulo	Zapatillas	1
7	Articulo	Polo	1

Output Panel:

- Action Output: # 1 Time 04:00:21 Action SELECT * FROM VentasProducto LIMIT 0, 1000
- Message: 13 row(s) returned

▼ Vista de Cantidad de ejemplares de un producto

▼ Código

```

-- Vista de Cantidad de ejemplares de un producto
CREATE VIEW Vista_Cantidad_Ejemplares AS
SELECT Identificacion AS Producto_ID, Cantidad_Articulo, Tipo_Articulo AS Ejemplares
FROM Producto;

```

▼ Test

```

-- ¿Cómo llamar la vista?
SELECT * FROM Vista_Cantidad_Ejemplares;

```

The screenshot shows a MySQL Workbench interface. In the top query editor, there are two lines of SQL code:

```

1 -- ¿Cómo llamar la vista?
2 SELECT * FROM Vista_Cantidad_Ejemplares;

```

The bottom pane displays the results of the query in a "Result Grid". The grid has three columns: "Producto_ID", "Cantidad_Articulo", and "Ejemplares". The data is as follows:

Producto_ID	Cantidad_Articulo	Ejemplares
1	44	Camisa
2	77	Pantalon
3	119	Gorra
4	88	NULL
5	67	NULL
6	109	Zapatillas
7	93	Polo

Below the grid, the message "13 row(s) returned" is visible. On the right side of the interface, there are tabs for "Result Grid" and "Form Editor".

▼ Vista de los 3 productos más vendidos

▼ Código

```

-- Vista de los 3 productos más vendidos
CREATE VIEW Top3ProductosMasVendidos AS
SELECT p.Identificacion AS ID_Producto,
       p.Tipo_Articulo AS Nombre_Producto,
       COUNT(*) AS Cantidad_Ventas
FROM Producto p
LEFT JOIN Contiene c ON p.Identificacion = c.Producto_Identificacion
GROUP BY p.Identificacion
ORDER BY Cantidad_Ventas DESC
LIMIT 3;

```

▼ Test

```

-- ¿Cómo llamar la vista?
SELECT * FROM Top3ProductosMasVendidos

```

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a query editor window containing the following SQL code:

```

1 -- ¿Cómo llamar la vista?
2 • SELECT * FROM Top3ProductosMasVendidos
3

```

Below the query editor is a results grid titled "Result Grid". It has three columns: "ID_Producto", "Nombre_Producto", and "Cantidad_Ventas". The data is as follows:

ID_Producto	Nombre_Producto	Cantidad_Ventas
1	Camisa	1
2	Pantalon	1
3	Gorra	1

At the bottom of the interface, there's an "Output" section with a log entry:

```

MasVendidos 1 x
Output
Action Output
# Time Action
1 03:11:17 SELECT * FROM Top3ProductosMasVendidos LIMIT 0, 1000
Message
3 row(s) returned

```

Consultas

▼ ¿Cuál es el lugar que ha recibido más envíos?

▼ Código

```

SELECT l.Nombre FROM lugar AS l JOIN envio AS e USING (Lugar_codigo)
GROUP BY l.Nombre HAVING COUNT(Lugar_codigo) = (SELECT MAX(cuenta) FROM
(SELECT COUNT(Lugar_codigo) AS cuenta FROM envio GROUP BY Lugar_Codigo) AS conteo);

```

▼ Test

```

5
6 -- ¿Cuál es el lugar que ha recibido más envíos?
7 • SELECT l.Nombre FROM lugar AS l JOIN envio AS e USING (Lugar_codigo)
8 GROUP BY l.Nombre HAVING COUNT(Lugar_codigo) = (SELECT MAX(cuenta) FROM
9 (SELECT COUNT(Lugar_codigo) AS cuenta FROM envio GROUP BY Lugar_Codigo) AS conteo);
10

```

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query editor contains the same SQL code as the previous section. The results grid has one column labeled "Nombre" and displays the value "Cúcuta".

▼ ¿Cuál es la fecha en la que más se han hecho compras?

▼ Código

```

SELECT fecha_compra, COUNT(fecha_compra) FROM compra GROUP BY fecha_compra HAVING
COUNT(fecha_compra) = (SELECT MAX(fechas) FROM (SELECT fecha_compra, COUNT(fecha_compra) A

```

```
FROM compra GROUP BY fecha_compra) AS maximo);
```

▼ Test

```
10
11      -- ¿Cuál es la fecha en la que más se han hecho compras?
12 •   SELECT fecha_compra, COUNT(fecha_compra) FROM compra GROUP BY fecha_compra HAVING
13     COUNT(fecha_compra) = (SELECT MAX(fechas) FROM (SELECT fecha_compra, COUNT(fecha_compra) AS fechas
14     FROM compra GROUP BY fecha_compra) AS maximo);
15
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
fecha_compra	COUNT(fecha_compra)			
2024-03-10	2			

▼ ¿Cuál es la sección en la que se venden más productos?

▼ Código

```
SELECT Seccion_Articulo, COUNT(Seccion_Articulo) FROM producto AS p GROUP BY Seccion_Articulo
COUNT(Seccion_Articulo) = (SELECT MAX(suma_secciones) FROM (SELECT Seccion_Articulo, COUNT(Seccion_Articulo) AS suma_secciones
FROM producto AS p GROUP BY Seccion_Articulo) AS maximo);
```

▼ Test

```
15
16      -- ¿Cuál es la sección en la que se venden más productos?
17 •   SELECT Seccion_Articulo, COUNT(Seccion_Articulo) FROM producto AS p GROUP BY Seccion_Articulo HAVING
18     COUNT(Seccion_Articulo) = (SELECT MAX(suma_secciones) FROM (SELECT Seccion_Articulo, COUNT(Seccion_Articulo) AS suma_secciones
19     FROM producto AS p GROUP BY Seccion_Articulo) AS maximo);
20
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
Seccion_Articulo	COUNT(Seccion_Articulo)			
Nino	3			

▼ ¿Cuantos productos de tipo articulo tiene la tienda?

▼ Código

```
SELECT COUNT(Tipo_Producto) AS numArticulos FROM producto WHERE Tipo_Producto = 'Articulo'
```

▼ Test

```
26      -- ¿Cuantos productos de tipo articulo tiene la tienda?
27 •   SELECT COUNT(Tipo_Producto) AS numArticulos FROM producto WHERE Tipo_Producto = 'Articulo';
28
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
numArticulos				
9				

▼ ¿Cuál es el producto con el precio más alto?

▼ Código

```
SELECT Tipo_Producto, Tipo_Articulo, Precio
FROM producto
WHERE Precio = (SELECT MAX(Precio) FROM producto);
```

▼ Test

```
30      -- ¿Cuál es el producto con el precio más alto?
31 •  SELECT Tipo_Producto, Tipo_Articulo, Precio
32    FROM producto
33    WHERE Precio = (SELECT MAX(Precio) FROM producto);
```

Result Grid			
	Tipo_Producto	Tipo_Articulo	Precio
▶	Diseño	HULL	65.00

▼ ¿Cuál es el monto total de ventas por mes de cada año?

▼ Código

```
SELECT MONTH(Fecha_compra) AS Mes, YEAR(Fecha_compra) AS Año, SUM(Costo_total) AS Monto_Total
FROM compra
GROUP BY Mes, Año;
```

▼ Test

```
35      -- ¿Cuál es el monto total de ventas por mes de cada año?
36 •  SELECT MONTH(Fecha_compra) AS Mes, YEAR(Fecha_compra) AS Año, SUM(Costo_total) AS Monto_Total
37    FROM compra
38    GROUP BY Mes, Año;
```

Result Grid			
	Mes	Año	Monto_Total
▶	3	2024	925.00
	2	2024	100.00

▼ ¿Cuál es el promedio de edad de los clientes por género?

▼ Código

```
SELECT Genero, ROUND(AVG(Edad)) AS PromedioEdad -- ROUND usamos esta función ya que funciona igual en Python
FROM persona
GROUP BY Genero;
```

▼ Test

```
40      -- ¿Cuál es el promedio de edad de los clientes por género?
41 •  SELECT Genero, ROUND(AVG(Edad)) AS PromedioEdad -- ROUND usamos esta función ya que funciona igual en Python.
42    FROM persona
43    GROUP BY Genero;
```

Result Grid		
	Genero	PromedioEdad
▶	Hombr	35
	Mujer	31

▼ ¿Cuál es el cliente que ha realizado la compra más cara en términos de costo total?

▼ Código

```
SELECT C.Cuenta_Usuario, C.Costo_total AS Compra_mas_cara
FROM Compra AS C
WHERE C.Costo_total = (
    SELECT MAX(Costo_total)
    FROM Compra
);
```

▼ Test

```
55      -- ¿Cuál es el cliente que ha realizado la compra más cara en términos de costo total?
56 •     SELECT C.Cuenta_Usuario, C.Costo_total AS Compra_mas_cara
57     FROM Compra AS C
58     WHERE C.Costo_total = (
59         SELECT MAX(Costo_total)
60         FROM Compra
61     );
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	Cuenta_Usuario	Compra_mas_cara		
▶	juan_perez	200.00		

▼ ¿Cuál es la sección con el precio promedio más alto por artículo vendido?

▼ Código

```
-- Este da NULL ya que el articulo más comprado es un Diseño y estos no tienen Sección
SELECT p.Seccion_Articulo, AVG(p.Precio) AS PrecioPromedio
FROM producto AS p
JOIN contiene AS c ON p.Identificacion = c.Producto_Identificacion
GROUP BY p.Seccion_Articulo
HAVING AVG(p.Precio) = (SELECT MAX(PrecioPromedio)
    FROM (SELECT AVG(p.Precio) AS PrecioPromedio
        FROM producto AS p
        JOIN contiene AS c ON p.Identificacion = c.Producto_Identi
        GROUP BY p.Seccion_Articulo) AS PreciosPorSeccion);
```

▼ Test

```

63      -- CONSULTA ANIDADA -->
64      -- ¿Cuál es la sección con el precio promedio más alto por artículo vendido?
65  •   SELECT p.Seccion_Articulo, AVG(p.Precio) AS PrecioPromedio
66      FROM producto AS p
67      JOIN contiene AS c ON p.Identificacion = c.Producto_Identificacion
68      GROUP BY p.Seccion_Articulo
69      HAVING AVG(p.Precio) = (SELECT MAX(PrecioPromedio)
70          FROM (SELECT AVG(p.Precio) AS PrecioPromedio
71              FROM producto AS p
72              JOIN contiene AS c ON p.Identificacion = c.Producto_Identificacion
73              GROUP BY p.Seccion_Articulo) AS PreciosPorSeccion);

```

Result Grid | Filter Rows: _____ | Export: _____ | Wrap Cell Content: _____

Sección_Artículo	PrecioPromedio
HULL	56.250000