

Índice

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL.....	2
Cargar la Base de Datos.....	2
Ejercicio de Consultas y Manipulación de Datos.....	2
Consulta SELECT básica:.....	2
SELECT con filtro:.....	3
Uso de operadores en WHERE:.....	3
Uso de operadores avanzados:.....	4
Consultas con Ordenación:.....	5
Funciones de agregado:.....	6
Cláusula GROUP BY:.....	7
JOIN:.....	8
Transacciones:.....	9
Desarrollo de la Aplicación.....	10
Inserción de Nuevos Clientes desde la Consola:.....	10
Permitir la inserción de clientes en la base de datos (Nombre, Apellido, Email, Dirección y Teléfono), ingresados por medio de la consola.....	10
Visualización de Clientes:.....	10
Desarrollo de Consultas Adicionales.....	10
Gestión de Transacciones:.....	10
Creación/Eliminación de Objetos para Almacenar Resultados:.....	10
Constantes para la conexión a la base de datos.....	11
Main.....	11
Método insertarCliente.....	12
Método mostrarClientes.....	12
Método insertarProductoDesdeConsola y insertarProducto.....	13
Método insertarPedidoDesdeConsola y insertarPedido.....	13
Método consultarClientePorId.....	14
Método consultarDetallesPedidosPorCliente.....	16
Método obtenerId.....	17
Método obtenerConexión.....	17
Método cerrarConexión.....	17

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL

Cargar la Base de Datos

He realizado modificaciones en el sql que ha dado el profesor para que funcione, he añadido esto:

```
DROP DATABASE IF EXISTS database_script;  
CREATE DATABASE database_script;  
USE database_script;
```

Para que no haya redundancia he quitado el campo city de la tabla CUSTOMER

Ejercicio de Consultas y Manipulación de Datos

Hacer consultas en workbench.

- Consulta SELECT básica:

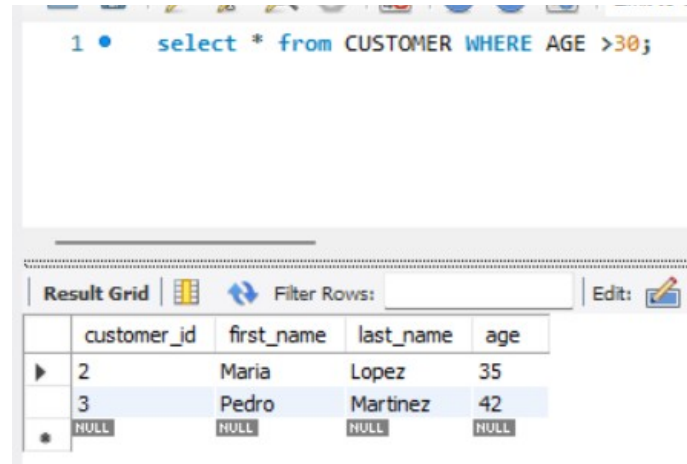
Muestra todos los registros de una tabla específica (por ejemplo, CUSTOMER).

The screenshot shows the MySQL Workbench interface. At the top, a tab labeled 'SQL File 4*' contains the file 'database_script'. Below the tab is a toolbar with various icons. The SQL editor shows the query: `1 • select * from address;`. Below the editor, the 'Result Grid' is displayed, showing the results of the query. The grid has columns: address_id, customer_id, street, city, and zipcode. The results are as follows:

	address_id	customer_id	street	city	zipcode
▶	1	1	Calle Falsa 123	Málaga	29001
	2	2	Avenida Real 45	Sevilla	41001
	3	3	Plaza Mayor 22	Granada	18001
	4	4	Calle Alta 8	Córdoba	14001
	5	5	Calle Baja 9	Jaén	23001
*	NULL	NULL	NULL	NULL	NULL

- **SELECT con filtro:**

- Realiza una consulta que obtenga solo los clientes cuya edad sea mayor de 30 años.
- Ordena los resultados alfabéticamente por nombre.



The screenshot shows a SQL query editor with the following query:

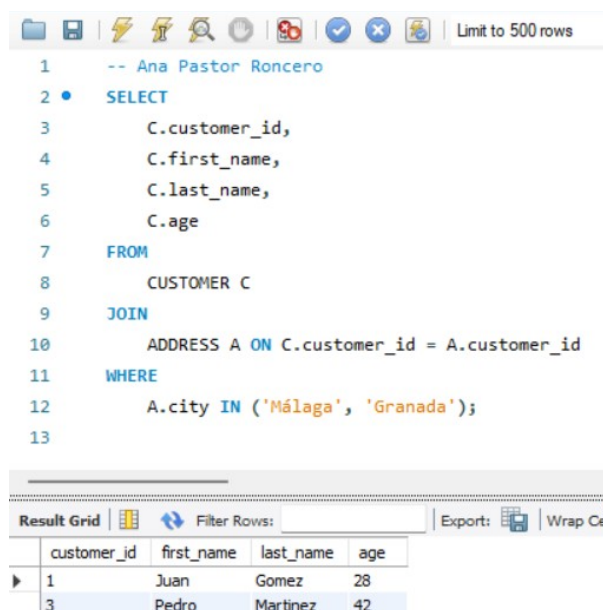
```
1 • select * from CUSTOMER WHERE AGE >30;
```

Below the query, there is a "Result Grid" tab. The results are displayed in a table with the following columns: customer_id, first_name, last_name, and age. The results are ordered alphabetically by first_name.

	customer_id	first_name	last_name	age
▶	2	Maria	Lopez	35
	3	Pedro	Martinez	42
*	NULL	NULL	NULL	NULL

- **Uso de operadores en WHERE:**

- Selecciona los clientes cuya ciudad sea "Málaga" o "Granada".



The screenshot shows a SQL query editor with the following query:

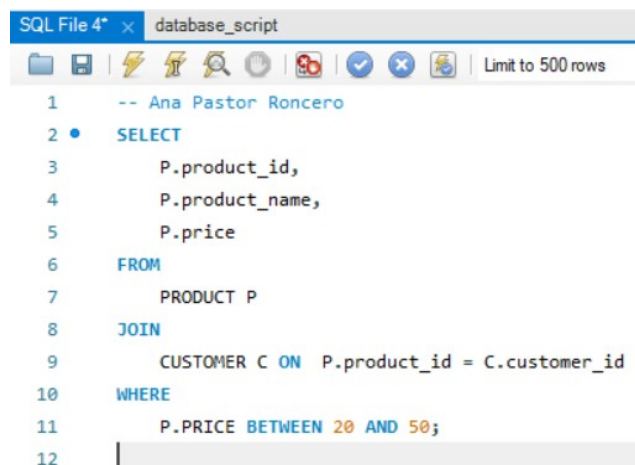
```
1 -- Ana Pastor Roncero
2 • SELECT
3     C.customer_id,
4     C.first_name,
5     C.last_name,
6     C.age
7 FROM
8     CUSTOMER C
9 JOIN
10    ADDRESS A ON C.customer_id = A.customer_id
11 WHERE
12    A.city IN ('Málaga', 'Granada');
13
```

Below the query, there is a "Result Grid" tab. The results are displayed in a table with the following columns: customer_id, first_name, last_name, and age. The results are ordered by customer_id.

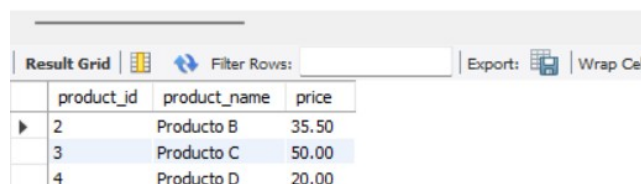
	customer_id	first_name	last_name	age
▶	1	Juan	Gomez	28
	3	Pedro	Martinez	42

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL

- Selecciona los productos cuyo precio esté entre 20 y 50 unidades monetarias.



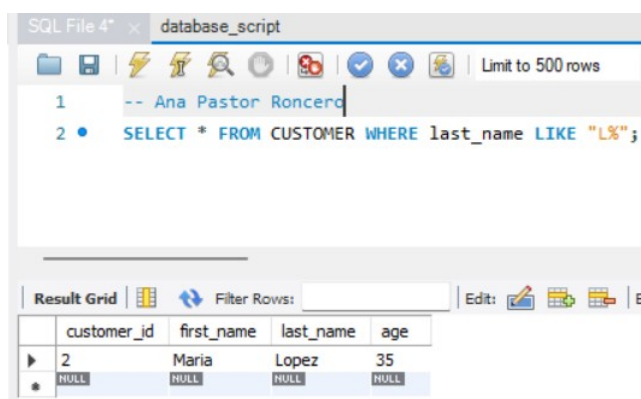
```
SQL File 4* x database_script
-- Ana Pastor Roncero
2 • SELECT
3     P.product_id,
4     P.product_name,
5     P.price
6 FROM
7     PRODUCT P
8 JOIN
9     CUSTOMER C ON P.product_id = C.customer_id
10 WHERE
11     P.PRICE BETWEEN 20 AND 50;
12
```



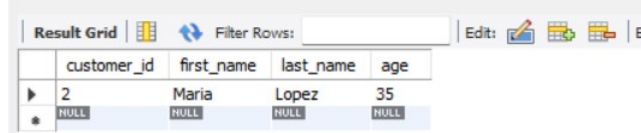
	product_id	product_name	price
▶	2	Producto B	35.50
	3	Producto C	50.00
	4	Producto D	20.00

- **Uso de operadores avanzados:**

- Usa el operador `LIKE` para seleccionar todos los registros de `CUSTOMER` cuyo apellido comience por "L".



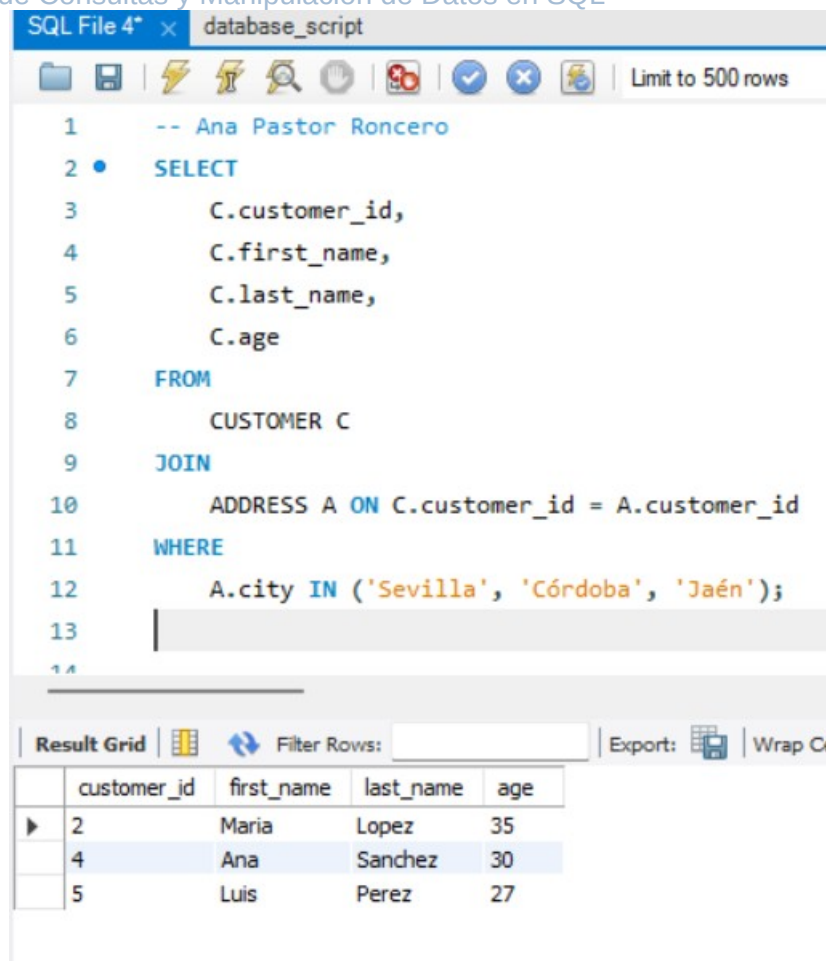
```
SQL File 4* x database_script
-- Ana Pastor Roncero
2 • SELECT * FROM CUSTOMER WHERE last_name LIKE "L%";
```



	customer_id	first_name	last_name	age
▶	2	Maria	Lopez	35
*	NULL	NULL	NULL	NULL

- Usa el operador `IN` para mostrar los clientes que vivan en "Sevilla", "Córdoba" o "Jaén".

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL



SQL File 4* x database_script

```

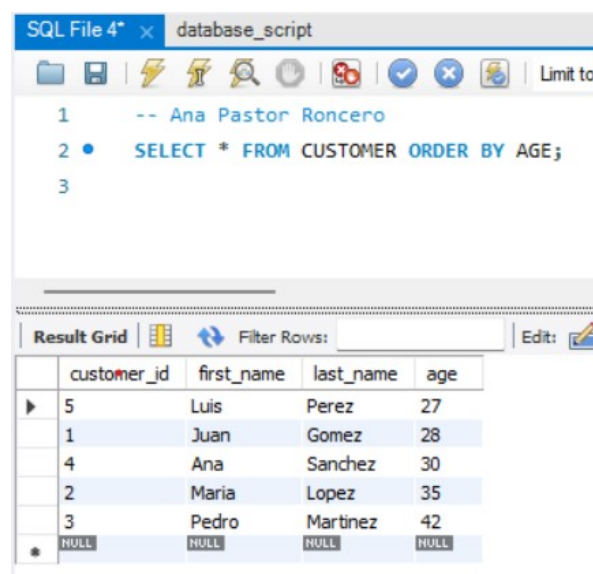
1  -- Ana Pastor Roncero
2  • SELECT
3      C.customer_id,
4      C.first_name,
5      C.last_name,
6      C.age
7  FROM
8      CUSTOMER C
9  JOIN
10     ADDRESS A ON C.customer_id = A.customer_id
11 WHERE
12     A.city IN ('Sevilla', 'Córdoba', 'Jaén');
13

```

Result Grid | Filter Rows: | Export: | Wrap C

	customer_id	first_name	last_name	age
▶	2	Maria	Lopez	35
	4	Ana	Sanchez	30
	5	Luis	Perez	27

- Consultas con Ordenación:
 - Realiza una consulta que muestre todos los clientes ordenados por edad.



SQL File 4* x database_script

```

1  -- Ana Pastor Roncero
2  • SELECT * FROM CUSTOMER ORDER BY AGE;
3

```

Result Grid | Filter Rows: | Edit:

	customer_id	first_name	last_name	age
▶	5	Luis	Perez	27
	1	Juan	Gomez	28
	4	Ana	Sanchez	30
	2	Maria	Lopez	35
	3	Pedro	Martinez	42
*	NULL	NULL	NULL	NULL

- Realiza una consulta que muestre los productos en orden ascendente por nombre.

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL

Limit to 500 rows

```

1  -- Ana Pastor Roncero
2  • SELECT * FROM PRODUCT ORDER BY product_name ASC;
    
```

product_id	product_name	price	category
1	Producto A	15.99	Electrónica
2	Producto B	35.50	Hogar
3	Producto C	50.00	Electrónica
4	Producto D	20.00	Jardinería
5	Producto E	70.99	Electrónica
* NULL	NULL	NULL	NULL

- Funciones de agregado:
 - Calcula el número total de clientes en la tabla.

Limit to 500 rows

```

1  -- Ana Pastor Roncero
2  • SELECT COUNT(*) FROM CUSTOMER AS numero_clientes;
    
```

COUNT(*)
5

- Calcula la media de edad de los clientes.

Limit to 500 rows

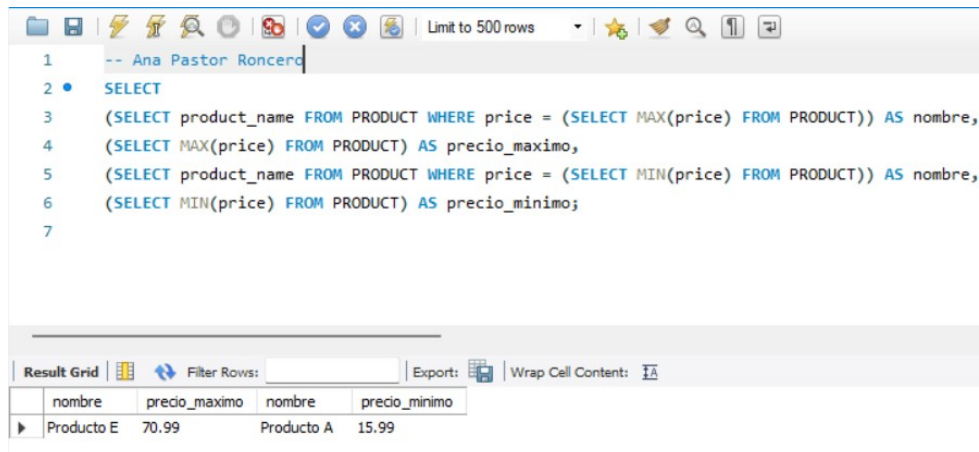
```

1  -- Ana Pastor Roncero
2  • SELECT avg(age) FROM CUSTOMER AS media_edad;
    
```

avg(age)
32.4000

- Obtén el precio máximo y mínimo de los productos.

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL



```

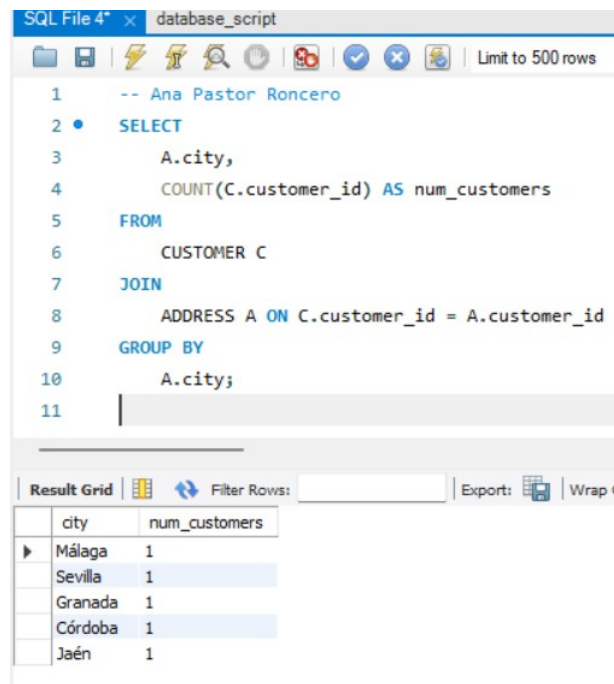
1  -- Ana Pastor Roncero
2  • SELECT
3  (SELECT product_name FROM PRODUCT WHERE price = (SELECT MAX(price) FROM PRODUCT)) AS nombre,
4  (SELECT MAX(price) FROM PRODUCT) AS precio_maximo,
5  (SELECT product_name FROM PRODUCT WHERE price = (SELECT MIN(price) FROM PRODUCT)) AS nombre,
6  (SELECT MIN(price) FROM PRODUCT) AS precio_minimo;
7

```

Result Grid

nombre	precio_maximo	nombre	precio_minimo
Producto E	70.99	Producto A	15.99

- Cláusula GROUP BY:
 - Muestra el número de clientes por cada ciudad.



```

1  -- Ana Pastor Roncero
2  • SELECT
3      A.city,
4      COUNT(C.customer_id) AS num_customers
5  FROM
6      CUSTOMER C
7  JOIN
8      ADDRESS A ON C.customer_id = A.customer_id
9  GROUP BY
10     A.city;
11

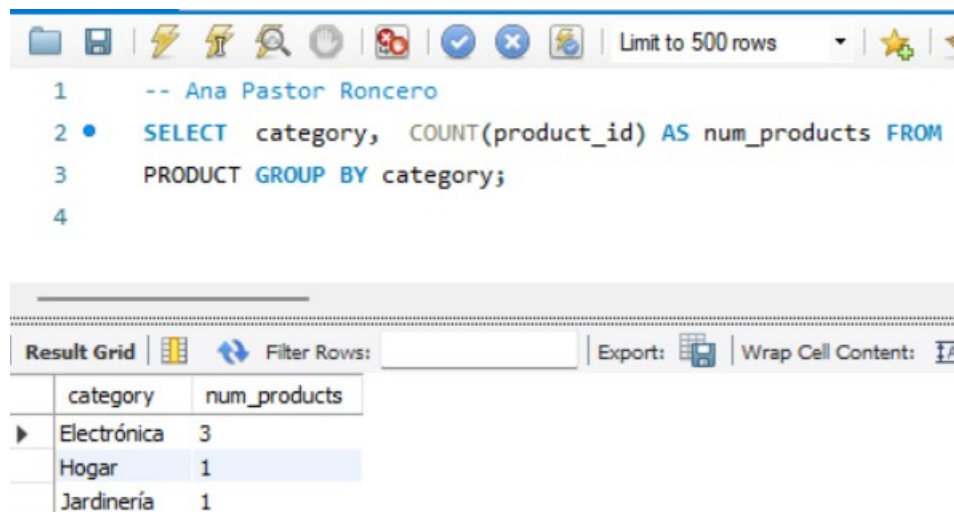
```

Result Grid

city	num_customers
Málaga	1
Sevilla	1
Granada	1
Córdoba	1
Jaén	1

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL

- Agrupa los productos por categoría y muestra la cantidad de productos en cada categoría.



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 500 rows' dropdown. The SQL editor contains the following query:

```

1  -- Ana Pastor Roncero
2  • SELECT category, COUNT(product_id) AS num_products FROM
3     PRODUCT GROUP BY category;
4

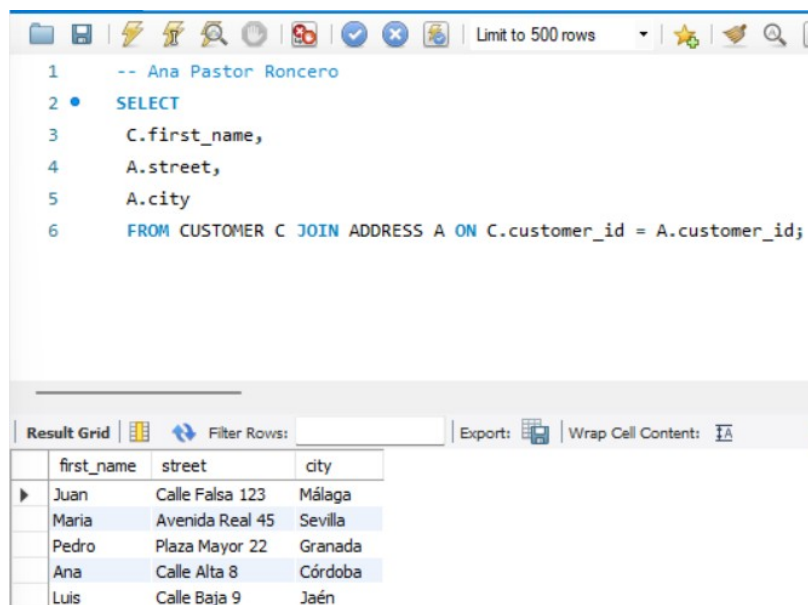
```

Below the editor is the 'Result Grid' tab, which displays the query results in a table format:

	category	num_products
▶	Electrónica	3
	Hogar	1
	Jardinería	1

- JOIN:

- Realiza una consulta que muestre el nombre del cliente y su dirección, combinando las tablas CUSTOMER y ADDRESS.



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 500 rows' dropdown. The SQL editor contains the following query:

```

1  -- Ana Pastor Roncero
2  • SELECT
3     C.first_name,
4     A.street,
5     A.city
6  FROM CUSTOMER C JOIN ADDRESS A ON C.customer_id = A.customer_id;

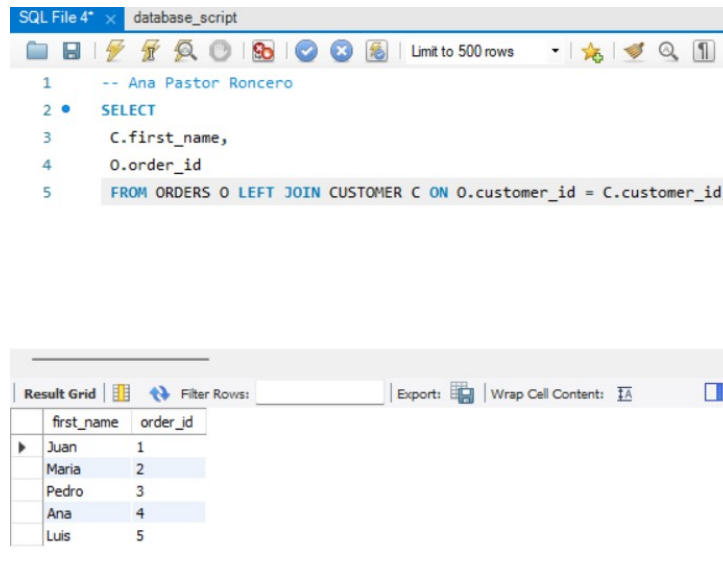
```

Below the editor is the 'Result Grid' tab, which displays the query results in a table format:

	first_name	street	city
▶	Juan	Calle Falsa 123	Málaga
	Maria	Avenida Real 45	Sevilla
	Pedro	Plaza Mayor 22	Granada
	Ana	Calle Alta 8	Córdoba
	Luis	Calle Baja 9	Jaén

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL

- Usa un **LEFT JOIN** para listar todos los pedidos junto con el nombre del cliente, aunque algunos pedidos no tengan cliente asignado.



- **Transacciones:**

- Realiza una transacción que incluya la inserción de un nuevo cliente y un nuevo pedido asociado a este cliente.
- Realiza un **ROLLBACK** en caso de error durante la transacción.
- Usa **COMMIT** para guardar los cambios si la transacción es exitosa.

```

DELIMITER //
CREATE PROCEDURE anadirCustomerYOrder()
BEGIN
    -- Iniciar la transacción
    START TRANSACTION;

    SET @customer_id = 6;

    -- Verificar si el cliente ya existe
    SELECT COUNT(*) INTO @exists FROM CUSTOMER WHERE customer_id = @customer_id;

    -- Intentar insertar el cliente solo si no existe
    INSERT INTO CUSTOMER (customer_id, first_name, last_name, age)
    VALUES (@customer_id, 'Sofia', 'Martinez', 29)
    ON DUPLICATE KEY UPDATE customer_id = customer_id; -- No hace nada si existe

    -- Comprobar si la inserción fue exitosa
    IF @exists = 0 THEN
        -- Insertar el pedido solo si el cliente fue insertado
        INSERT INTO ORDERS (order_id, customer_id, product_id, order_date, quantity)
        VALUES (6, @customer_id, 2, '2023-10-30', 2);

        -- Hacer commit
        COMMIT;
        SELECT 'Transacción completada exitosamente' AS Message;
    ELSE
        -- Hacer rollback si el cliente ya existe
        ROLLBACK;
        SELECT 'El cliente ya existe. Se han revertido los cambios.' AS Message;
    END IF;
END //
DELIMITER ;

```

Desarrollo de la Aplicación

- **Insertión de Nuevos Clientes desde la Consola:**

Permitir la inserción de clientes en la base de datos (Nombre, Apellido, Email, Dirección y Teléfono), ingresados por medio de la consola.

- **Visualización de Clientes:**

Permitir la visualización de todos los clientes almacenados en la base de datos, mostrando su información en una lista en la consola.

- **Desarrollo de Consultas Adicionales**

Crear consultas para buscar clientes por nombre o email. Consultar detalles de pedidos para un cliente específico y calcular el precio total de cada pedido.

- **Gestión de Transacciones:**

Deshabilitar el modo autocommit y utilizar confirmaciones (COMMIT) y deshacer (ROLLBACK) para mantener la integridad de los datos durante la inserción y eliminación de registros.

- **Creación/Eliminación de Objetos para Almacenar Resultados:**

Usa executeUpdate() , que devuelve un número entero con el conteo de filas afectadas, Usar cerrarConexion(dbConnection) para liberar los recursos al finalizar.

Aquí se ven todos los métodos y el main:

```

1  import java.sql.*;
2  import java.util.Scanner;
3
4  public class ProductoManager {
5
6      private static final String DRIVER = "com.mysql.cj.jdbc.Driver";
7      private static final String URL_CONEXION = "jdbc:mysql://localhost:3306/database_script";
8      private static final String USUARIO = "root";
9      private static final String PASSWORD = "Med@e";
10
11      Run | Debug
12      public static void main(String[] args) { ...
13
14          private static void insertarClienteDesdeConsola() { ...
15
16          private static void insertarCliente(int customerId, String firstName, String lastName, int age) ...
17
18          private static void mostrarClientes() { ...
19
20          private static void insertarProductoDesdeConsola() { ...
21
22          private static void insertarProducto(int productId, String productName, double price, String category) ...
23
24          private static void insertarPedidoDesdeConsola() { ...
25
26          private static void insertarPedido(int orderId, int customerId, int productId, int quantity) ...
27
28          private static void consultarClientePorId() { ...
29
30          private static void consultarDetallesPedidosPorCliente() { ...
31
32          private static int obtenerNuevoId(String tabla, String columna) throws SQLException, ClassNotFoundException { ...
33
34          private static Connection obtenerConexion() throws SQLException, ClassNotFoundException { ...
35
36          private static void cerrarConexion(Connection connection) { ...
37      }
38  }

```

Constantes para la conexión a la base de datos

En la clase ProductoManager.java creamos las variables privadas y estáticas para acceder desde los métodos. Para luego introducirlas en el método conexión a la base de datos.

```
public class ProductoManager {  
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";  
    private static final String URL_CONEXION = "jdbc:mysql://localhost:3306/database_script";  
    private static final String USUARIO = "root";  
    private static final String PASSWORD = "Med@c";  
}
```

Main

En el método main, creamos un menú de opciones que se repite continuamente hasta que el usuario elige la opción 7 para salir. En cada caso del switch, llamamos al método correspondiente que realiza las consultas, los cuales están definidos fuera del método main.

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
  
    while (true) {  
        System.out.println("Seleccione una opción:");  
        System.out.println("1. Insertar nuevo cliente");  
        System.out.println("2. Mostrar todos los clientes");  
        System.out.println("3. Insertar nuevo producto");  
        System.out.println("4. Insertar nuevo pedido");  
        System.out.println("5. Consultar cliente por ID");  
        System.out.println("6. Consultar detalles de pedidos por cliente");  
        System.out.println("7. Salir");  
        int opcion = scanner.nextInt();  
        scanner.nextLine();  
  
        switch (opcion) {  
            case 1:  
                insertarClienteDesdeConsola();  
                break;  
            case 2:  
                mostrarClientes();  
                break;  
            case 3:  
                insertarProductoDesdeConsola();  
                break;  
            case 4:  
                insertarPedidoDesdeConsola();  
                break;  
            case 5:  
                consultarClientePorId();  
                break;  
            case 6:  
                consultarDetallesPedidosPorCliente();  
                break;  
            case 7:  
                System.out.println("Saliendo...");  
                scanner.close();  
                System.exit(0);  
                break;  
            default:  
                System.out.println("Opción no válida. Por favor, intente de nuevo.");  
        }  
    }  
}
```

Método insertarCliente

En este método establece una conexión a la base de datos y desactiva el autocommit para manejar transacciones. Intenta insertar un nuevo cliente preguntándole al usuario nombre... y si lo consigue insertar lo insertamos llamando al método insertarCliente y se confirma la transacción.

```
private static void insertarClienteDesdeConsola() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Ingresar información del cliente:");
    int customerId = 0;

    try {
        customerId = obtenerNuevoId("CUSTOMER", "customer_id"); // Obtener nuevo ID
        System.out.print("Nombre: ");
        String firstName = scanner.nextLine();
        System.out.print("Apellido: ");
        String lastName = scanner.nextLine();
        System.out.print("Edad: ");
        int age = scanner.nextInt();

        insertarCliente(customerId, firstName, lastName, age);
    } catch (SQLException | ClassNotFoundException e) {
        System.out.println("Error al insertar el cliente: " + e.getMessage());
    }
}
```

Método mostrarClientes

Este método es una consulta que muestra toda la tabla de Clientes

```
private static void mostrarClientes() {
    Connection dbConnection = null;

    try {
        dbConnection = obtenerConexion();
        Statement statement = dbConnection.createStatement();
        String selectTableSQL = "SELECT * FROM CUSTOMER";
        ResultSet rs = statement.executeQuery(selectTableSQL);

        while (rs.next()) {
            int id = rs.getInt("customer_id");
            String nombre = rs.getString("first_name");
            String apellido = rs.getString("last_name");
            int edad = rs.getInt("age");
            System.out.println("ID Cliente: " + id);
            System.out.println("Nombre: " + nombre);
            System.out.println("Apellido: " + apellido);
            System.out.println("Edad: " + edad);
            System.out.println("-----");
        }

    } catch (SQLException e) {
        System.out.println("Error al mostrar los datos de los clientes: " + e.getMessage());
    } catch (ClassNotFoundException e) {
        System.out.println("Error al cargar el controlador de la base de datos: " + e.getMessage());
    } finally {
        cerrarConexion(dbConnection); // Cerrar la conexión al finalizar
    }
}
```

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL

Método insertarProductoDesdeConsola y insertarProducto

Este método es para la opción de insertar productos pide al usuario el nombre del producto, el precio y la categoría. Pero no comprueba nada y lo inserta llamando al método insertarProducto

```
private static void insertarProductoDesdeConsola() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Ingresar información del producto:");
    int productId = 0;

    try {
        productId = obtenerNuevoId("PRODUCT", "product_id"); // Obtener nuevo ID

        System.out.print("Nombre del producto: ");
        String productName = scanner.nextLine();
        System.out.print("Precio: ");
        double price = scanner.nextDouble();
        scanner.nextLine(); // Limpiar el buffer
        System.out.print("Categoría: ");
        String category = scanner.nextLine();

        insertarProducto(productId, productName, price, category);
    } catch (SQLException | ClassNotFoundException e) {
        System.out.println("Error al insertar el producto: " + e.getMessage());
    }
}
```

Este no se usa en el menú y la diferencia es que recibe por parámetro es útil si alguna vez cambiamos el programa.

```
private static void insertarProducto(int productId, String productName, double price, String category)
    throws SQLException, ClassNotFoundException {
    Connection dbConnection = null;

    try {
        dbConnection = obtenerConexion();
        dbConnection.setAutoCommit(false); // Deshabilitar el modo autocommit

        // Insertar el producto
        String insertProductoSQL = "INSERT INTO PRODUCT (product_id, product_name, price, category) VALUES (?, ?, ?, ?)";
        PreparedStatement productoStatement = dbConnection.prepareStatement(insertProductoSQL);
        productoStatement.setInt(1, productId);
        productoStatement.setString(2, productName);
        productoStatement.setDouble(3, price);
        productoStatement.setString(4, category);

        int filasAfectadasProducto = productoStatement.executeUpdate();

        // Confirmar la transacción si la inserción es exitosa
        if (filasAfectadasProducto > 0) {
            dbConnection.commit(); // Confirma la transacción
            System.out.println("Producto insertado con éxito.");
        } else {
            System.out.println("No se pudo insertar el producto.");
            dbConnection.rollback(); // Deshacer la transacción en caso de fallo
        }
    } catch (SQLException e) {
        System.out.println("Error: " + e.getMessage());
        if (dbConnection != null) {
            dbConnection.rollback(); // Deshacer la transacción en caso de error
        }
    } finally {
        cerrarConexion(dbConnection);
    }
}
```

Método insertarPedidoDesdeConsola y insertarPedido

Métodos que funcionan igual que los de insertar producto desde consola e insertar producto.

Tarea 6.1: Ejecución de Consultas y Manipulación de Datos en SQL

Solo que pide el id del cliente que hace el pedido y el id del producto que quiere comprar y la cantidad que quiere comprar.

```
private static void insertarPedido(int orderId, int customerId, int productId, int quantity)
    throws SQLException, ClassNotFoundException {
    Connection dbConnection = null;

    try {
        dbConnection = obtenerConexion();
        dbConnection.setAutoCommit(false); // Deshabilitar el modo autocommit

        // Insertar el pedido
        String insertPedidoSQL = "INSERT INTO ORDERS (order_id, customer_id, product_id, quantity) VALUES (?, ?, ?, ?)";
        PreparedStatement pedidoStatement = dbConnection.prepareStatement(insertPedidoSQL);
        pedidoStatement.setInt(1, orderId);
        pedidoStatement.setInt(2, customerId);
        pedidoStatement.setInt(3, productId);
        pedidoStatement.setInt(4, quantity);

        int filasAfectadasPedido = pedidoStatement.executeUpdate();

        // Confirmar la transacción si la inserción es exitosa
        if (filasAfectadasPedido > 0) {
            dbConnection.commit(); // Confirma la transacción
            System.out.println("Pedido insertado con éxito.");
        } else {
            System.out.println("No se pudo insertar el pedido.");
            dbConnection.rollback(); // Deshacer la transacción en caso de fallo
        }
    } catch (SQLException e) {
        System.out.println("Error: " + e.getMessage());
        if (dbConnection != null) {
            dbConnection.rollback(); // Deshacer la transacción en caso de error
        }
    } finally {
        cerrarConexion(dbConnection);
    }
}
```

```
private static void insertarPedidoDesdeConsola() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Ingresar información del pedido:");

    try {
        int orderId = obtenerNuevoId("ORDERS", "order_id"); // Obtener nuevo ID automáticamente
        System.out.print("ID del cliente: ");
        int customerId = scanner.nextInt();
        System.out.print("ID del producto: ");
        int productId = scanner.nextInt();
        System.out.print("Cantidad: ");
        int quantity = scanner.nextInt();

        // Llamar al método para insertar el pedido
        insertarPedido(orderId, customerId, productId, quantity);
    } catch (SQLException | ClassNotFoundException e) {
        System.out.println("Error al insertar el pedido: " + e.getMessage());
    }
}
```

Método consultarClientePorId

Muestra la información de un cliente por id, pidiendole al usuario el id del cliente que quiere ver.

En todos los métodos llamamos al método de conexión de base de datos. Y utilizamos el PreparedStatement para hacer la consulta e introducir la variable donde se encuentra la interrogación.

```
private static void consultarClientePorId() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Ingrese el ID del cliente: ");
    int customerId = scanner.nextInt();

    Connection dbConnection = null;

    try {
        dbConnection = obtenerConexion();
        String selectSQL = "SELECT * FROM CUSTOMER WHERE customer_id = ?";
        PreparedStatement preparedStatement = dbConnection.prepareStatement(selectSQL);
        preparedStatement.setInt(1, customerId);
        ResultSet rs = preparedStatement.executeQuery();

        if (rs.next()) {
            System.out.println("ID Cliente: " + rs.getInt("customer_id"));
            System.out.println("Nombre: " + rs.getString("first_name"));
            System.out.println("Apellido: " + rs.getString("last_name"));
            System.out.println("Edad: " + rs.getInt("age"));
        } else {
            System.out.println("Cliente no encontrado.");
        }

    } catch (SQLException e) {
        System.out.println("Error al consultar el cliente: " + e.getMessage());
    } catch (ClassNotFoundException e) {
        System.out.println("Error al cargar el controlador de la base de datos: " + e.getMessage());
    } finally {
        cerrarConexion(dbConnection);
    }
}
```


Método consultarDetallesPedidosPorCliente

Muestra todos los pedidos que ha realizado un cliente. Si los tiene si no no. Mostrando la cantidad nombre, precio y el id del producto.

```
private static void consultarDetallesPedidosPorCliente() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Ingrese el ID del cliente: ");
    int customerId = scanner.nextInt();

    Connection dbConnection = null;

    try {
        dbConnection = obtenerConexion();
        String selectSQL = "SELECT o.order_id, o.quantity, p.product_name, p.price " +
            "FROM ORDERS o JOIN PRODUCT p ON o.product_id = p.product_id " +
            "WHERE o.customer_id = ?";
        PreparedStatement preparedStatement = dbConnection.prepareStatement(selectSQL);
        preparedStatement.setInt(1, customerId);
        ResultSet rs = preparedStatement.executeQuery();

        double total = 0.0;
        boolean tienePedidos = false;

        while (rs.next()) {
            tienePedidos = true;
            int orderId = rs.getInt("order_id");
            int quantity = rs.getInt("quantity");
            String productName = rs.getString("product_name");
            double price = rs.getDouble("price");

            double totalPorPedido = quantity * price;
            total += totalPorPedido;

            System.out.println("ID Pedido: " + orderId);
            System.out.println("Producto: " + productName);
            System.out.println("Cantidad: " + quantity);
            System.out.println("Precio Total: " + totalPorPedido);
            System.out.println("-----");
        }

        if (tienePedidos) {
            System.out.println("Precio total de todos los pedidos: " + total);
        } else {
            System.out.println("No se encontraron pedidos para este cliente.");
        }

    } catch (SQLException e) {
        System.out.println("Error al consultar los pedidos: " + e.getMessage());
    } catch (ClassNotFoundException e) {
        System.out.println("Error al cargar el controlador de la base de datos: " + e.getMessage());
    } finally {
        cerrarConexion(dbConnection);
    }
}
```

Método obtenerId

Lo he creado para no tener que pedir en consola que el usuario introduzca el id cuando quiera insertar algo, así no tiene que mirar porque id va la tabla y que se ponga solo en el siguiente id disponible.

Lo llamo en todos los métodos de insertar.

Lo he realizado haciendo una consulta para obtener el máximo Id que haya en la tabla.

```
private static int obtenerNuevoId(String tabla, String columna) throws SQLException, ClassNotFoundException {
    Connection dbConnection = null;
    int nuevoId = 1; // Valor por defecto

    try {
        dbConnection = obtenerConexion();
        String selectSQL = "SELECT MAX(" + columna + ") AS max_id FROM " + tabla;
        PreparedStatement preparedStatement = dbConnection.prepareStatement(selectSQL);
        ResultSet rs = preparedStatement.executeQuery();

        if (rs.next()) {
            nuevoId = rs.getInt("max_id") + 1; // Incrementar el ID máximo
        }

    } catch (SQLException e) {
        System.out.println("Error al obtener el nuevo ID: " + e.getMessage());
    } finally {
        cerrarConexion(dbConnection);
    }

    return nuevoId; // Retornar el nuevo ID
}
```

Para no tener que estar creando en todos los métodos una conexión a la base de datos y ahorrar código he creado estos dos métodos y los he llamado dentro de los otros:

Método obtenerConexion

```
private static Connection obtenerConexion() throws SQLException, ClassNotFoundException {
    Class.forName(DRIVER);
    return DriverManager.getConnection(URL_CONEXION, USUARIO, PASSWORD);
}
```

Método cerrarConexion

```
private static void cerrarConexion(Connection connection) {
    if (connection != null) {
        try {
            connection.close();
        } catch (SQLException e) {
            System.out.println("Error al cerrar la conexión: " + e.getMessage());
        }
    }
}
```