



ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS



BASE DE DATOS

PROFESOR:

Ing. Yadira Franco R

PERÍODO ACADÉMICO:

2024-B

TAREA

TÍTULO:

INVESTIGACIÓN Y PRACTICA



Estudiante

Cristian Tambaco

2024-B

Examen -PRACTICO de Base de Datos:

Objetivo del examen: El examen tiene como objetivo evaluar tu capacidad para trabajar con bases de datos utilizando MySQL. A lo largo de este examen, deberás crear tablas, procedimientos almacenados, funciones, triggers y gestionar usuarios con privilegios específicos. Es importante que sigas las instrucciones detalladamente y utilices buenas prácticas al diseñar y crear la base de datos.

Link GitHub:

https://github.com/CristianTambaco/Base_Datos_PRACTICA-TAREA--VS-EXAMEN.git

Instrucciones Generales:

1. Crear la Base de Datos y las Tablas:

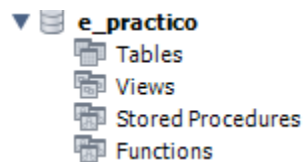
Crea la base de datos y las tablas necesarias utilizando el script SQL proporcionado. Asegúrate de que las tablas se creen correctamente con las relaciones adecuadas y los campos correctos.

-- crear la base de datos "E_practico"

CREATE DATABASE E_practico;

-- seleccionar la base de datos "E_practico"

USE E_practico;



Las tablas a crear son:

- Clientes

-- Creación de la tabla Clientes

CREATE TABLE Clientes (

```

ClienteID INT AUTO_INCREMENT PRIMARY KEY,
Nombre VARCHAR(100) NOT NULL,
Apellidos VARCHAR(100) NOT NULL,
FechaNacimiento DATE NOT NULL,
Telefono VARCHAR(15),
Correo VARCHAR(100)
);

```

Table: **clientes**

Columns:

ClienteID	int AI PK
Nombre	varchar(100)
Apellidos	varchar(100)
FechaNacimiento	date
Telefono	varchar(15)
Correo	varchar(100)

	ClienteID	Nombre	Apellidos	FechaNacimiento	Telefono	Correo
*	NULL	NULL	NULL	NULL	NULL	NULL

▪ Categorías

-- Creación de la tabla Categorías

```

CREATE TABLE Categorías (
  CategoriáID INT AUTO_INCREMENT PRIMARY KEY,
  Descripción VARCHAR(100) NOT NULL
);

```

Table: **categorías**

Columns:

CategoriáID	int AI PK
Descripción	varchar(100)

	CategoriáID	Descripción
*	NULL	NULL

▪ Productos

-- Creación de la tabla Productos

```

CREATE TABLE Productos (
  ProductoID INT AUTO_INCREMENT PRIMARY KEY,
  Nombre VARCHAR(100) NOT NULL,

```

```

    Precio DECIMAL(10, 2) NOT NULL,
    CategoriaID INT,
    FOREIGN KEY (CategoriaID) REFERENCES Categorias(CategoriaID)
);

```

Table: **productos**

Columns:

ProductoID int AI PK
 Nombre varchar(100)
 Precio decimal(10,2)
CategoriaID int

	ProductoID	Nombre	Precio	CategoriaID
*	NULL	NULL	NULL	NULL

▪ Ordenes

-- Creación de la tabla Ordenes

```

CREATE TABLE Ordenes (
    OrdenID INT AUTO_INCREMENT PRIMARY KEY,
    ClienteID INT,
    FechaOrden DATE NOT NULL,
    MontoTotal DECIMAL(10, 2),
    FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);

```

Table: **ordenes**

Columns:

OrdenID int AI PK
ClienteID int
 FechaOrden date
 MontoTotal decimal(10,2)

	OrdenID	ClienteID	FechaOrden	MontoTotal
*	NULL	NULL	NULL	NULL

▪ DetallesOrden

-- Creación de la tabla DetallesOrden

```

CREATE TABLE DetallesOrden (
    DetalleID INT AUTO_INCREMENT PRIMARY KEY,
    OrdenID INT,
    ProductoID INT,
    Cantidad INT,
    PrecioUnitario DECIMAL(10, 2),

```

```

FOREIGN KEY (OrdenID) REFERENCES Ordenes(OrdenID),
FOREIGN KEY (ProductoID) REFERENCES Productos(ProductoID)
);

```

Table: **detallesorden**

Columns:

DetalleID	int AI PK
OrdenID	int
ProductoID	int
Cantidad	int
PrecioUnitario	decimal(10,2)

	DetalleID	OrdenID	ProductoID	Cantidad	PrecioUnitario
*	NULL	NULL	NULL	NULL	NULL

▪ AuditoriaEliminaciones

-- Creación de la tabla AuditoriaEliminaciones

```

CREATE TABLE AuditoriaEliminaciones (
    ProductoID INT,
    Nombre VARCHAR(100),
    Precio DECIMAL(10, 2),
    FechaEliminacion DATETIME
);

```

Table: **auditoriaeliminaciones**

Columns:

ProductoID	int
Nombre	varchar(100)
Precio	decimal(10,2)
FechaEliminacion	datetime

	ProductoID	Nombre	Precio	FechaEliminacion
--	------------	--------	--------	------------------

▪ AuditoriaPrecios

-- Creación de la tabla AuditoriaPrecios

```

CREATE TABLE AuditoriaPrecios (
    ProductoID INT,
    PrecioAnterior DECIMAL(10, 2),
    PrecioNuevo DECIMAL(10, 2),
    FechaCambio DATETIME
);

```

);

Table: auditoriaprecios

Columns:

ProductoID int
PrecioAnterior decimal(10,2)
PrecioNuevo decimal(10,2)
FechaCambio datetime

	ProductoID	PrecioAnterior	PrecioNuevo	FechaCambio

2. Insertar Datos de Ejemplo:

Inserta los datos de ejemplo proporcionados en el script. Asegúrate de que cada tabla tenga al menos unos pocos registros para realizar las consultas posteriores.

-- Insertar registros de ejemplo en Clientes

INSERT INTO Clientes (Nombre, Apellidos, FechaNacimiento, Telefono, Correo) VALUES

('Juan', 'Pérez', '1985-02-15', '555123456', 'juan.perez@mail.com'),

('Ana', 'Gómez', '1990-06-25', '555654321', 'ana.gomez@mail.com'),

('Carlos', 'Lopez', '1982-11-12', '555987654', 'carlos.lopez@mail.com');

-- visualizar tabla

SELECT * FROM Clientes;

	ClienteID	Nombre	Apellidos	FechaNacimiento	Telefono	Correo
▶	1	Juan	Pérez	1985-02-15	555123456	juan.perez@mail.com
	2	Ana	Gómez	1990-06-25	555654321	ana.gomez@mail.com
	3	Carlos	Lopez	1982-11-12	555987654	carlos.lopez@mail.com
*	NULL	NULL	NULL	NULL	NULL	NULL

-- Insertar registros de ejemplo en Categorías

INSERT INTO Categorías (Descripción) VALUES

('Electrónica'),

('Hogar'),

('Ropa'),

('Alimentos');

-- visualizar tabla

SELECT * FROM Categorias;

	CategoriaID	Descripcion
▶	1	Electrónica
	2	Hogar
	3	Ropa
	4	Alimentos
✱	NULL	NULL

-- Insertar registros de ejemplo en Productos

INSERT INTO Productos (Nombre, Precio, CategoriaID) VALUES

('Televisor', 2500.00, 1),

('Sofá', 1500.00, 2),

('Camiseta', 25.00, 3),

('Leche', 2.50, 4);

-- visualizar tabla

SELECT * FROM Productos;

	ProductoID	Nombre	Precio	CategoriaID
▶	1	Televisor	2500.00	1
	2	Sofá	1500.00	2
	3	Camiseta	25.00	3
	4	Leche	2.50	4
✱	NULL	NULL	NULL	NULL

-- Insertar registros de ejemplo en Ordenes

INSERT INTO Ordenes (ClienteID, FechaOrden, MontoTotal) VALUES

(1, '2024-12-01', 2550.00),

(2, '2024-12-05', 1525.00),

(3, '2024-12-10', 27.50);

-- visualizar tabla

SELECT * FROM Ordenes;

	OrdenID	ClienteID	FechaOrden	MontoTotal
▶	1	1	2024-12-01	2550.00
	2	2	2024-12-05	1525.00
	3	3	2024-12-10	27.50
*	NULL	NULL	NULL	NULL

-- Insertar registros de ejemplo en DetallesOrden

INSERT INTO DetallesOrden (OrdenID, ProductoID, Cantidad, PrecioUnitario) VALUES

(1, 1, 1, 2500.00),

(1, 2, 1, 50.00),

(2, 3, 2, 25.00),

(3, 4, 5, 2.50);

-- visualizar tabla

SELECT * FROM DetallesOrden;

	DetalleID	OrdenID	ProductoID	Cantidad	PrecioUnitario
▶	1	1	1	1	2500.00
	2	1	2	1	50.00
	3	2	3	2	25.00
	4	3	4	5	2.50
*	NULL	NULL	NULL	NULL	NULL

3. Realizar las Consultas y Procedimientos:

Desarrolla las consultas, funciones, procedimientos almacenados, triggers y usuarios según los siguientes requerimientos.

Requerimientos de la practica Examen:

1. Función para Calcular el Total de una Orden:

- Crea una **función llamada CalcularTotalOrden** que reciba como parámetro un **OrdenID** y devuelva el monto total de la orden.

- El monto total debe ser la suma del precio de los productos multiplicado por la cantidad de cada uno, utilizando las tablas **DetallesOrden** y **Productos**.

DELIMITER //

CREATE FUNCTION CalcularTotalOrden(OrdenID INT) -- función que recibe como parámetro un OrdenID

RETURNS DECIMAL(10, 2)

DETERMINISTIC

BEGIN

DECLARE total DECIMAL(10, 2);

SELECT SUM(p.Precio * d.Cantidad) INTO total -- suma del precio de los productos multiplicado por la cantidad

FROM DetallesOrden d

JOIN Productos p ON d.ProductoID = p.ProductoID

WHERE d.OrdenID = OrdenID;

RETURN total; -- devuelva el monto total de la orden

END//

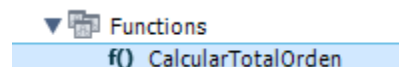
DELIMITER ;

Function: **CalcularTotalOrden**

Parameters:

OrdenID: INT

Returns: decimal(10,2)



-- Llamar funcion

SELECT CalcularTotalOrden(2);

	CalcularTotalOrden(2)
▶	50.00

2. Procedimiento para Insertar un Cliente:

- Crea un **procedimiento almacenado** llamado **InsertarCliente** que reciba los parámetros **Nombre**, **Apellidos**, **FechaNacimiento**, **Telefono** y **Correo**, y los inserte en la tabla **Clientes**.
- Después de la inserción, muestra un mensaje que indique que el cliente ha sido registrado exitosamente.

DELIMITER //

CREATE PROCEDURE InsertarCliente(-- procedimiento almacenado que recibe parámetros Nombre, Apellidos, FechaNacimiento, Telefono y Correo

IN p_Nombre VARCHAR(50),

IN p_Apellidos VARCHAR(50),

IN p_FechaNacimiento DATE,

IN p_Telefono VARCHAR(20),

IN p_Correo VARCHAR(50)

)

BEGIN

INSERT INTO Clientes (Nombre, Apellidos, FechaNacimiento, Telefono, Correo) -- inserta en la tabla Clientes

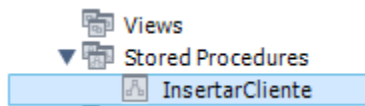
VALUES (p_Nombre, p_Apellidos, p_FechaNacimiento, p_Telefono, p_Correo);

SELECT 'Cliente registrado exitosamente' AS Mensaje; -- muestra un mensaje indicando que el cliente ha sido registrado exitosamente

END//

DELIMITER ;

Procedure: **InsertarCliente**



-- Llamar procedimiento almacenado

CALL InsertarCliente("Marco", "Lopez", "1990-04-04", 255555, "marcolopez@gmail.com");

Mensaje
▶ Cliente registrado exitosamente

	ClienteID	Nombre	Apellidos	FechaNacimiento	Telefono	Correo
▶	1	Juan	Pérez	1985-02-15	555123456	juan.perez@mail.com
	2	Ana	Gómez	1990-06-25	555654321	ana.gomez@mail.com
	3	Carlos	Lopez	1982-11-12	555987654	carlos.lopez@mail.com
	4	Marco	Lopez	1990-04-04	255555	marcolopez@gmail.com
*	NULL	NULL	NULL	NULL	NULL	NULL

3. Trigger para Auditar Eliminaciones:

- Crea un **trigger** que se active después de una eliminación en la tabla **Productos**.
- El trigger debe registrar la eliminación en la tabla **AuditoriaEliminaciones**, incluyendo el **ProductoID**, **Nombre**, **Precio** y la fecha de eliminación.

DELIMITER //

CREATE TRIGGER AuditarEliminacionProducto

AFTER DELETE ON Productos -- trigger que se activa después de una eliminación en la tabla Productos

FOR EACH ROW

BEGIN -- El trigger registra en la tabla AuditoriaEliminaciones el ProductoID, Nombre, Precio y FechaEliminacion.

INSERT INTO AuditoriaEliminaciones (ProductoID, Nombre, Precio, FechaEliminacion)

VALUES (OLD.ProductoID, OLD.Nombre, OLD.Precio, NOW());

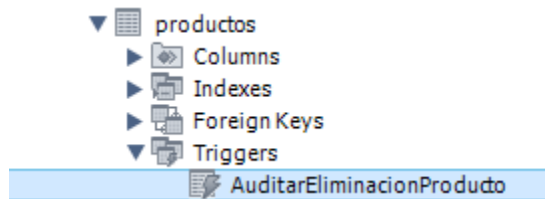
END//

DELIMITER ;

Trigger: AuditarEliminacionProducto

Definition:

Event DELETE
Timing AFTER



-- Insertar registro en la tabla Productos

```
INSERT INTO Productos (ProductoID, Nombre, Precio, CategoriaID)
VALUES (5,"Queso",4.00,4);
```

-- Eliminar registro de la tabla Productos

```
DELETE FROM Productos WHERE ProductoID = 5;
```

-- Visualizar la tabla AuditoriaEliminaciones

```
SELECT * FROM AuditoriaEliminaciones;
```

	ProductoID	Nombre	Precio	FechaEliminacion
▶	5	Queso	4.00	2025-01-04 16:51:16

4. Creación de un Usuario con Privilegios Específicos:

- Crea un **usuario llamado usuario_cliente** con una contraseña segura.
- Este usuario debe tener privilegios de solo lectura (**SELECT**) sobre las tablas **Clientes**, **Productos** y **Ordenes**, sin permisos para insertar, actualizar ni eliminar registros.

-- usuario llamado usuario_cliente con una contraseña

CREATE USER 'usuario_cliente'@'localhost' IDENTIFIED BY 'read123';

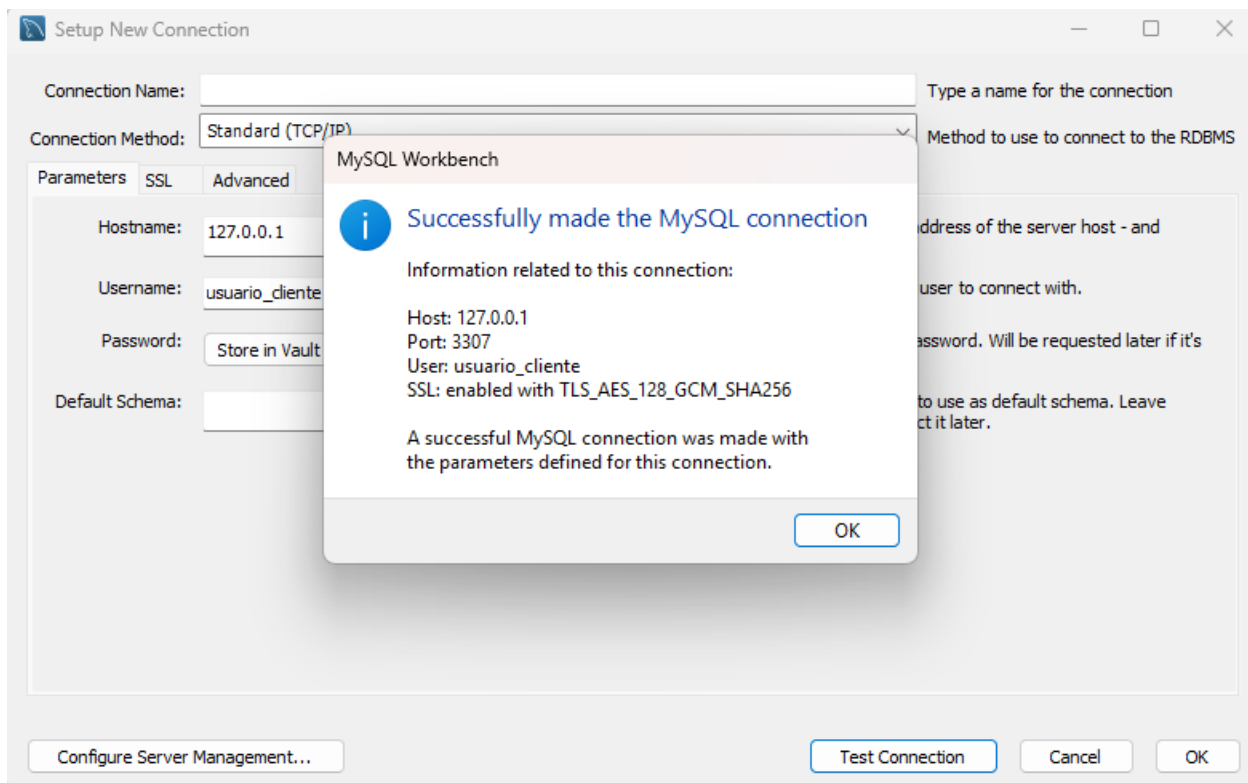
-- privilegios de solo lectura (SELECT) sobre las tablas Clientes, Productos y Ordenes de la base de datos E_practico

GRANT SELECT ON E_practico.Clientes TO 'usuario_cliente'@'localhost';

GRANT SELECT ON E_practico.Productos TO 'usuario_cliente'@'localhost';

GRANT SELECT ON E_practico.Ordenes TO 'usuario_cliente'@'localhost';

FLUSH PRIVILEGES;



5. Procedimiento para Actualizar el Precio de un Producto:

- Crea un **procedimiento almacenado** llamado **ActualizarPrecioProducto** que reciba como parámetros un **ProductoID** y un **PrecioNuevo**.
- Este procedimiento debe actualizar el precio del producto en la tabla **Productos** y registrar este cambio en la tabla **AuditoriaPrecios**, almacenando el **PrecioAnterior**, **PrecioNuevo** y la fecha de cambio.

DELIMITER //

```

CREATE PROCEDURE ActualizarPrecioProducto( -- procedimiento almacenado que recibe como
parámetros ProductoID y PrecioNuevo

    IN p_ProductoID INT,

    IN p_PrecioNuevo DECIMAL(10, 2)

)

BEGIN

    DECLARE p_PrecioAnterior DECIMAL(10, 2);

    -- Obtener el precio anterior del producto

    SELECT Precio INTO p_PrecioAnterior

    FROM Productos

    WHERE ProductoID = p_ProductoID;

    -- Actualizar el precio del producto en la tabla Productos

    UPDATE Productos

    SET Precio = p_PrecioNuevo

    WHERE ProductoID = p_ProductoID;

    -- Registrar el cambio en la tabla AuditoriaPrecios, almacenando el PrecioAnterior, PrecioNuevo y
    FechaCambio

    INSERT INTO AuditoriaPrecios (ProductoID, PrecioAnterior, PrecioNuevo, FechaCambio)

    VALUES (p_ProductoID, p_PrecioAnterior, p_PrecioNuevo, NOW());

END//

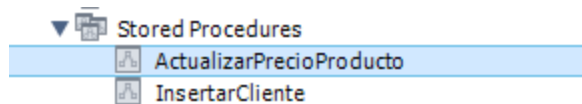
DELIMITER ;

```

Procedure: **ActualizarPrecioProducto**

Parameters:

p_ProductoID: [IN] INT
p_PrecioNuevo: [IN] DECIMAL(10,2)



-- Llamar procedimiento almacenado

CALL ActualizarPrecioProducto(3,50.00);

	ProductoID	Nombre	Precio	CategoriaID
▶	1	Televisor	2500.00	1
	2	Sofá	1500.00	2
	3	Camiseta	50.00	3
	4	Leche	2.50	4
*	NULL	NULL	NULL	NULL

6. Función para Obtener el Total Gastado por un Cliente:

- Crea una **función llamada TotalGastadoPorCliente** que reciba como parámetro un **ClienteID** y devuelva el monto total gastado por ese cliente en todas sus órdenes.
- La función debe calcular el total utilizando las tablas **Ordenes** y **DetallesOrden**.

DELIMITER //

CREATE FUNCTION TotalGastadoPorCliente(ClienteID INT) -- función que recibe como parámetro ClienteID

RETURNS DECIMAL(10, 2)

DETERMINISTIC

BEGIN

DECLARE total DECIMAL(10, 2);

SELECT SUM(d.Cantidad * d.PrecioUnitario) INTO total -- suma de la cantidad por el precio unitario

FROM DetallesOrden d -- calcular el total utilizando las tablas Ordenes y DetallesOrden

JOIN Ordenes o ON d.OrdenID = o.OrdenID

WHERE o.ClienteID = ClienteID;

RETURN total; -- devuelva el monto total gastado por ese cliente en todas sus órdenes

END//

DELIMITER ;

Function: TotalGastadoPorCliente

Parameters:

ClienteID: INT

Returns: decimal(10,2)

▼ Functions
f() CalcularTotalOrden
f() TotalGastadoPorCliente

-- Llamar funcion

SELECT TotalGastadoPorCliente(1);

TotalGastadoPorCliente(1)
2550.00

	OrdenID	ClienteID	FechaOrden	MontoTotal
▶	1	1	2024-12-01	2550.00
	2	2	2024-12-05	1525.00
	3	3	2024-12-10	27.50
*	NULL	NULL	NULL	NULL

7. Trigger para Actualizar el Stock de Productos:

- Crea un **trigger AFTER INSERT** en la tabla **DetallesOrden** para actualizar el stock de productos en la tabla **Productos** cada vez que se inserte un nuevo detalle de orden.
- La cantidad vendida debe restarse del stock actual del producto. Si el stock es menor que 1, el trigger debe generar un error.

DELIMITER //

CREATE TRIGGER ActualizarStock AFTER INSERT ON DetallesOrden -- trigger AFTER INSERT en la tabla DetallesOrden

FOR EACH ROW -- para actualizar el stock cada vez que se inserte un nuevo detalle de orden

BEGIN

DECLARE stock_actual INT;

-- Obtener el stock actual del producto

SELECT Stock INTO stock_actual

FROM Productos

WHERE ProductoID = NEW.ProductoID;

-- Actualizar el stock del producto

UPDATE Productos

SET Stock = stock_actual - NEW.Cantidad -- La cantidad vendida se resta del stock actual del producto

WHERE ProductoID = NEW.ProductoID;

-- Verificar si el stock es suficiente

IF (stock_actual - NEW.Cantidad) < 1 THEN -- Si el stock es menor que 1, el trigger genera un error y muestra un mensaje

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stock insuficiente';

END IF;

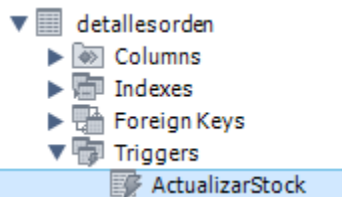
END//

DELIMITER ;

Trigger: **ActualizarStock**

Definition:

Event INSERT
Timing AFTER



-- Visualizar la tabla Auditoriaprecios

SELECT * FROM Auditoriaprecios;

	ProductoID	PrecioAnterior	PrecioNuevo	FechaCambio
▶	3	25.00	50.00	2025-01-04 16:18:12

8. Procedimiento para Generar un Informe de Ventas por Categoría:

- Crea un **procedimiento almacenado** llamado **InformeVentasPorCategoria** que reciba como parámetro un **CategoriaID** y devuelva el total de ventas para esa categoría.
- El informe debe mostrar el nombre de la categoría y el total de ventas, incluyendo el nombre de los productos vendidos y la cantidad de cada uno.

DELIMITER //

CREATE PROCEDURE InformeVentasPorCategoria(-- procedimiento almacenado que recibe como parámetro CategoriaID

IN p_CategoriaID INT

)

BEGIN

-- el informe muestra el nombre de la categoría y el total de ventas incluyendo el nombre de los productos y la cantidad vendida

SELECT c.Descripcion AS Categoria,

p.Nombre AS Producto,

SUM(d.Cantidad) AS CantidadVendida,

SUM(d.Cantidad * d.PrecioUnitario) AS TotalVentas

FROM DetallesOrden d

JOIN Productos p ON d.ProductoID = p.ProductoID

JOIN Categorias c ON p.CategoriaID = c.CategoriaID

WHERE p.CategoriaID = p_CategoriaID

GROUP BY c.Descripcion, p.Nombre;


END//


DELIMITER ;


Procedure: **InformeVentasPorCategoria**


Parameters:

p_CategoriaID: [IN] INT

▼  Stored Procedures

 ActualizarPrecioProducto

 InformeVentasPorCategoria

 InsertarCliente

-- Llamar procedimiento almacenado

CALL InformeVentasPorCategoria(3);

	Categoria	Producto	CantidadVendida	TotalVentas
▶	Ropa	Camiseta	2	50.00

9. Creación de un Usuario Administrador con Privilegios Completos:

- Crea un **usuario administrador** llamado **admin_ventas** con privilegios completos sobre todas las tablas de la base de datos.
- Este usuario debe tener permisos para ejecutar procedimientos almacenados y triggers.

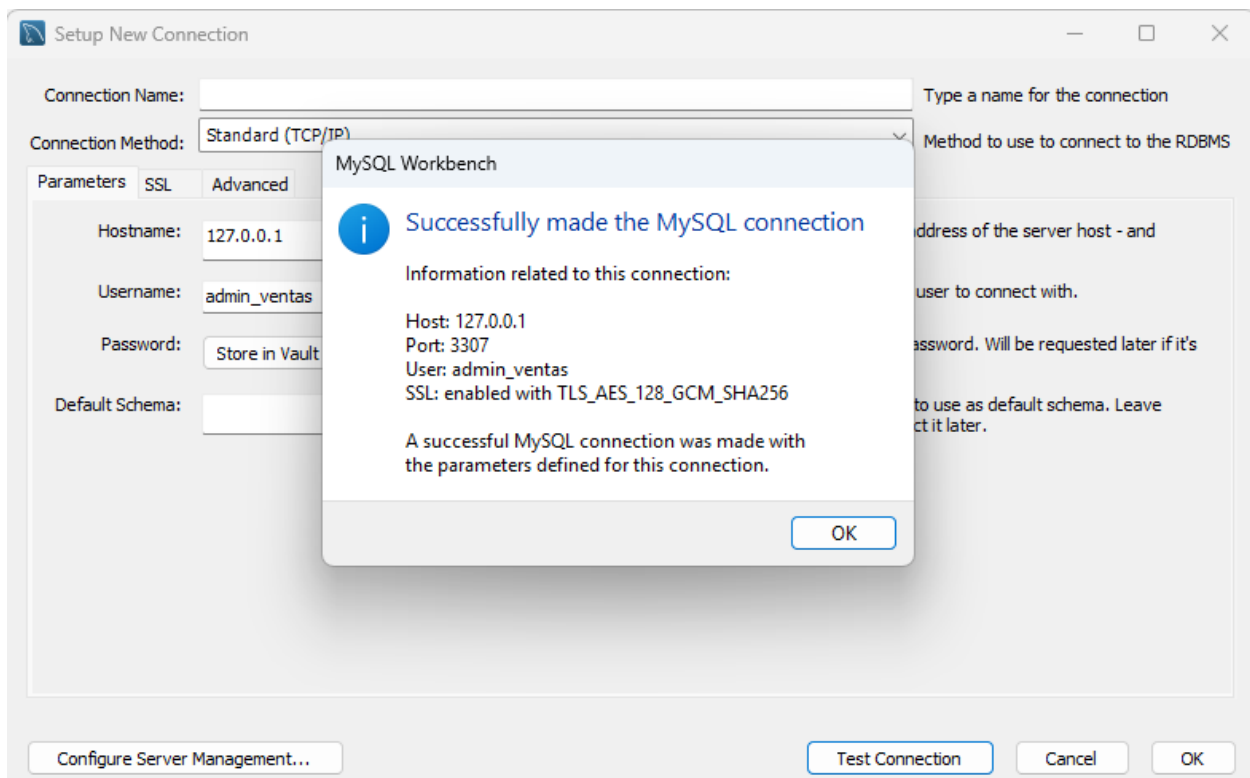
-- usuario llamado admin_ventas con una contraseña

CREATE USER 'admin_ventas'@'localhost' IDENTIFIED BY 'admin123';

-- privilegios completos sobre todas las tablas de la base de datos E_practico

GRANT ALL PRIVILEGES ON E_practico.* TO 'admin_ventas'@'localhost';

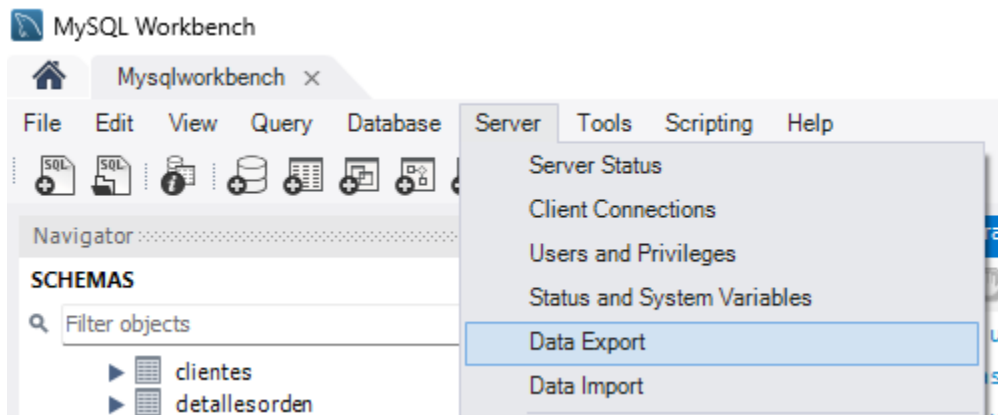
FLUSH PRIVILEGES;



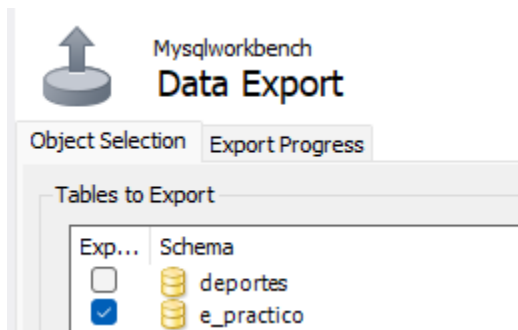
10. Backup Completo de la Base de Datos:

- Realiza un **backup completo de la base de datos** después de haber completado todos los ejercicios. Guarda el archivo generado y súbelo junto con las consultas que hayas ejecutado.

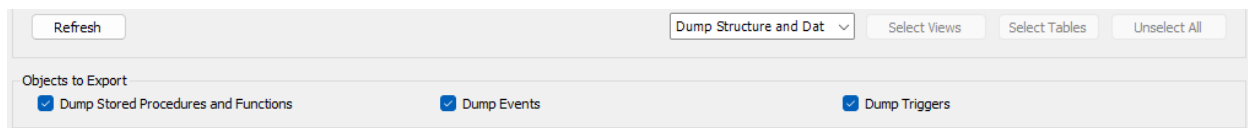
- Se hace clic en **Server > Data Export** en el menú superior



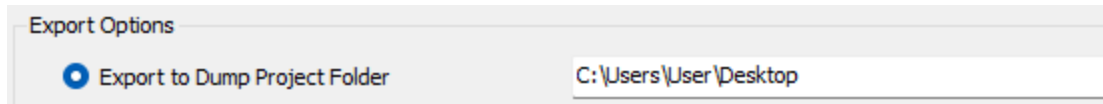
- Se selecciona la base de datos



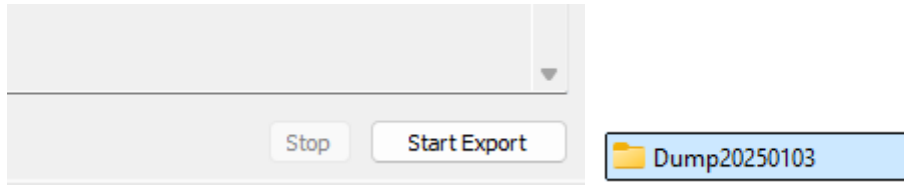
- Se elige las opciones de exportación



- Se selecciona la ruta donde se desea guardar el archivo de respaldo.



- Se hace clic en Start Export y se exporta en una carpeta.



- NO SE OLVIDE CREAR EL INFORME EN PDF

Link GitHub:

https://github.com/CristianTambaco/Base_Datos_PRACTICA-TAREA--VS-EXAMEN.git

DESCARGUE EL SCRIPT EN EL AULA VIRTUAL