

INVESTIGAR QUE SON Procedimientos Almacenados en Bases de Datos

- Entender qué son los procedimientos almacenados y cómo funcionan.
- Aprender a crear procedimientos almacenados sencillos.
- PRACTICA - Realizar operaciones de **INSERT**, **SELECT**, **DELETE** y **UPDATE** usando procedimientos almacenados.
- **Revisión de Buenas Prácticas**

Introducción a los Procedimientos Almacenados **MSQL- PostgreSQL – Sql Server**

1. Concepto y Beneficios de los Procedimientos Almacenados

- **Explicación:** Los procedimientos almacenados son conjuntos de instrucciones SQL que se guardan y ejecutan en el servidor de base de datos. Permiten ejecutar operaciones complejas, con seguridad, rendimiento optimizado y reutilización de código.
- **Beneficios:**
 - Reutilización de código.
 - Mejora en la seguridad (al evitar inyecciones SQL).
 - Optimización en el rendimiento de consultas frecuentes.
 - Consistencia en las operaciones realizadas.

2. ESPECIFICAR LA Sintaxis Básica de un Procedimiento Almacenado

- **Explicación:** El delimitador se cambia temporalmente para permitir el uso de **;** dentro del procedimiento.

Crear la tabla de cliente:

CREATE TABLE cliente (

 ClienteID INT AUTO_INCREMENT PRIMARY KEY, -- Campo para el ID único del cliente

 Nombre VARCHAR(100), -- Campo para el nombre del cliente

 Estatura DECIMAL(5,2), -- Campo para la estatura del cliente con dos decimales

 FechaNacimiento DATE, -- Campo para la fecha de nacimiento del cliente

 Sueldo DECIMAL(10,2) -- Campo para el sueldo del cliente con dos decimales

);

3. Ejercicio 1: Crear un procedimiento simple que seleccione datos de la tabla cliente

Inserción, Actualización y Eliminación de Datos

1. Procedimiento de Inserción (INSERT)

- Crear un procedimiento que permita insertar un nuevo cliente en la tabla cliente

```
/*1. Procedimiento de Inserción (INSERT)*/
DELIMITER //

-- Crear el procedimiento de inserción
CREATE PROCEDURE InsertarCliente(
    IN p_Nombre VARCHAR(100),
    IN p_Estatura DECIMAL(5,2),
    IN p_FechaNacimiento DATE,
    IN p_Sueldo DECIMAL(10,2)
)
BEGIN
    INSERT INTO cliente (Nombre, Estatura, FechaNacimiento, Sueldo)
    VALUES (p_Nombre, p_Estatura, p_FechaNacimiento, p_Sueldo);
END;
//

DELIMITER ;
```

- Ejecutar - LLAMAR el procedimiento

```
/*Llamar al Procedimiento para Insertar un Nuevo Cliente*/
CALL InsertarCliente('Luis Gómez', 1.80, '1992-06-14', 35000.00);
```

Verificar

```
/*Verificar la Inserción*/
SELECT * FROM cliente;
```

Result Grid					
Filter Rows: <input type="text"/>					
	ClienteID	Nombre	Estatura	FechaNacimiento	Sueldo
	3	Juan García	1.82	1978-11-12	28000.75
	4	Ana Torres	1.60	1995-05-30	30000.00
	5	Pedro Martínez	1.90	1988-01-18	27000.25
	6	Luis Gómez	1.80	1992-06-14	35000.00
*	NULL	NULL	NULL	NULL	NULL

cliente 1

2. Procedimiento de Actualización (UPDATE)

Actualizar la edad de un cliente específico:

Como no existe en la tabla el atributo de edad, se va hacer el procedimiento para sueldo

```
/*Procedimiento de Actualización*/
DELIMITER //

-- Crear el procedimiento de actualización
CREATE PROCEDURE ActualizarCliente(
    IN p_ClienteID INT,          -- ID del cliente a actualizar
    IN p_Nombre VARCHAR(100),    -- Nuevo nombre
    IN p_Estatura DECIMAL(5,2),  -- Nueva estatura
    IN p_Sueldo DECIMAL(10,2)    -- Nuevo sueldo
)
BEGIN
    -- Actualizar los datos del cliente específico
    UPDATE cliente
    SET Nombre = p_Nombre,
        Estatura = p_Estatura,
        Sueldo = p_Sueldo
    WHERE ClienteID = p_ClienteID;
END;
//

DELIMITER ;
```

Llamar al procedimiento

```
/*Llamar al Procedimiento para Actualizar un Cliente*/
CALL ActualizarCliente(2, 'María González', 1.70, 34000.00);
```

Verificar

```

/*Verificar la Actualización*/
SELECT * FROM cliente;

```

Antes

Result Grid						Filter Rows:	Edit:	Export/Import:
	CienteID	Nombre	Estatura	FechaNacimiento	Sueldo			
▶	1	Carlos López	1.75	1985-03-15	25000.50			
	2	María Pérez	1.68	1990-07-22	32000.00			
	3	Juan García	1.82	1978-11-12	28000.75			
	4	Ana Torres	1.60	1995-05-30	30000.00			
	5	Pedro Martínez	1.90	1988-01-18	27000.25			

cliente 1 x App

Después

Result Grid						Filter Rows:	Edit:	Export/Import:
	CienteID	Nombre	Estatura	FechaNacimiento	Sueldo			
▶	1	Carlos López	1.75	1985-03-15	25000.50			
	2	María González	1.70	1990-07-22	34000.00			
	3	Juan García	1.82	1978-11-12	28000.75			
	4	Ana Torres	1.60	1995-05-30	30000.00			
	5	Pedro Martínez	1.90	1988-01-18	27000.25			

cliente 2 x App

3. Procedimiento de Eliminación (DELETE)

Eliminar un cliente de la base de datos usando su ClienteID:

```

DELIMITER //

CREATE PROCEDURE EliminarCliente(
    IN p_ClienteID INT -- ID del cliente a eliminar
)
BEGIN
    DELETE FROM cliente
    WHERE ClienteID = p_ClienteID;
END;
//

DELIMITER ;

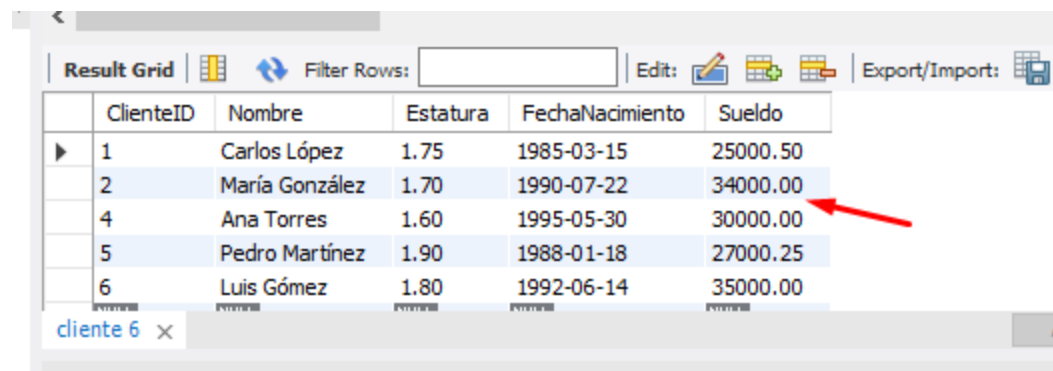
```

Llamar al procedimiento

```
/*Llamar al procedimiento*/  
CALL EliminarCliente(3);
```

Verificar

```
/*Verificar*/  
SELECT * FROM cliente;
```



	ClienteID	Nombre	Estatura	FechaNacimiento	Sueldo
▶	1	Carlos López	1.75	1985-03-15	25000.50
	2	María González	1.70	1990-07-22	34000.00
	4	Ana Torres	1.60	1995-05-30	30000.00
	5	Pedro Martínez	1.90	1988-01-18	27000.25
	6	Luis Gómez	1.80	1992-06-14	35000.00

Introducción a Condiciones en Procedimientos Almacenados

Uso de Condicionales (IF)

El uso de condicionales dentro de los procedimientos es fundamental para tomar decisiones basadas en los datos.

Verifica si la edad de un cliente es mayor o igual a 22:

```
DELIMITER //
```

```
) CREATE PROCEDURE VerificarEdadCliente(  
    IN p_ClienteID INT  
- )  
) BEGIN  
    DECLARE edad INT;  
    -- Calcular la edad del cliente  
    SELECT TIMESTAMPDIF(YEAR, FechaNacimiento, CURDATE()) INTO edad  
    FROM cliente  
    WHERE ClienteID = p_ClienteID;  
  
    -- Verificar si la edad es mayor o igual a 22  
) IF edad >= 22 THEN
```

```

        SELECT CONCAT('El cliente con ID ', p_ClienteID, ' tiene ', edad, ' años y es mayor o igual a 22.');
```

```

ELSE
        SELECT CONCAT('El cliente con ID ', p_ClienteID, ' tiene ', edad, ' años y es menor a 22.');
```

```

END IF;

END;

//

DELIMITER ;
|
CALL VerificarEdadCliente(2);

```

Creación de la Tabla de Órdenes CON RELACIÓN CON EL CLIENTE - FORANEA

Para almacenar las órdenes de los clientes, se debe crear la tabla **ordenes**:

- Procedimientos de Órdenes -Insertar Orden
Crear la tabla ordenes

```

CREATE TABLE ordenes (
    OrdenID INT AUTO_INCREMENT PRIMARY KEY,
    ClienteID INT,
    FechaOrden DATE,
    MontoTotal DECIMAL(10,2),
    FOREIGN KEY (ClienteID) REFERENCES cliente(ClienteID)
);

```

- Procedimientos de Órdenes -Insertar Orden

```

DELIMITER //

CREATE PROCEDURE InsertarOrden(
    IN p_ClienteID INT,
    IN p_FechaOrden DATE,
    IN p_MontoTotal DECIMAL(10,2)
)
BEGIN
    INSERT INTO ordenes (ClienteID, FechaOrden, MontoTotal)
    VALUES (p_ClienteID, p_FechaOrden, p_MontoTotal);
END;

//

DELIMITER ;
|
CALL InsertarOrden(2, '2024-12-17', 500.00);

```

Result Grid

Filter Rows:

Edit:

	OrdenID	ClienteID	FechaOrden	MontoTotal
▶	1	2	2024-12-17	500.00
★	NULL	NULL	NULL	NULL


- Procedimientos Actualizar Orden

```
DELIMITER //
```

```
CREATE PROCEDURE ActualizarOrden(  
    IN p_OrdenID INT,  
    IN p_FechaOrden DATE,  
    IN p_MontoTotal DECIMAL(10,2)  
)  
  
BEGIN  
    UPDATE ordenes  
    SET FechaOrden = p_FechaOrden,  
        MontoTotal = p_MontoTotal  
    WHERE OrdenID = p_OrdenID;  
END;  
//
```

```
DELIMITER ;
```

```
CALL ActualizarOrden(1, '2024-12-18', 600.00);
```

Result Grid			Filter Rows:		Ec
	OrdenID	ClienteID	FechaOrden	MontoTotal	
▶	1	2	2024-12-18	600.00	
✱	NULL	NULL	NULL	NULL	

- Procedimientos Eliminar Orden

```
DELIMITER //
```

```
CREATE PROCEDURE EliminarOrden(  
    IN p_OrdenID INT  
)  
  
BEGIN  
    DELETE FROM ordenes  
    WHERE OrdenID = p_OrdenID;  
END;  
//
```

```
DELIMITER ;
```

```
CALL EliminarOrden(1);
```

Result Grid

Filter Rows:

Edit

	OrdenID	ClienteID	FechaOrden	MontoTotal
*	NULL	NULL	NULL	NULL

Entrega Final

Instrucciones de Entrega:

1. Objetivos:

Crear procedimientos almacenados para **insertar, actualizar, eliminar y consultar** registros en las tablas cliente y ordenes.

2. Archivo de Script:

Los estudiantes deben escribir y guardar el código SQL con todos los procedimientos mencionados.

3. Documento PDF:

Incluir las capturas de pantalla y explicaciones detalladas de los pasos realizados durante la tarea.

4. Subida a GitHub:

Subir el script .sql y el documento PDF a un repositorio en GitHub para su REVISIÓN

Link del repositorio:

https://github.com/Sebastian-Betancourt/Deber_bases_datos.git