

Bases de Datos

Nombre: Joshua Morocho

PARTE 1

CREACIÓN DE ROLES Y ASIGNACIÓN DE PRIVILEGIOS

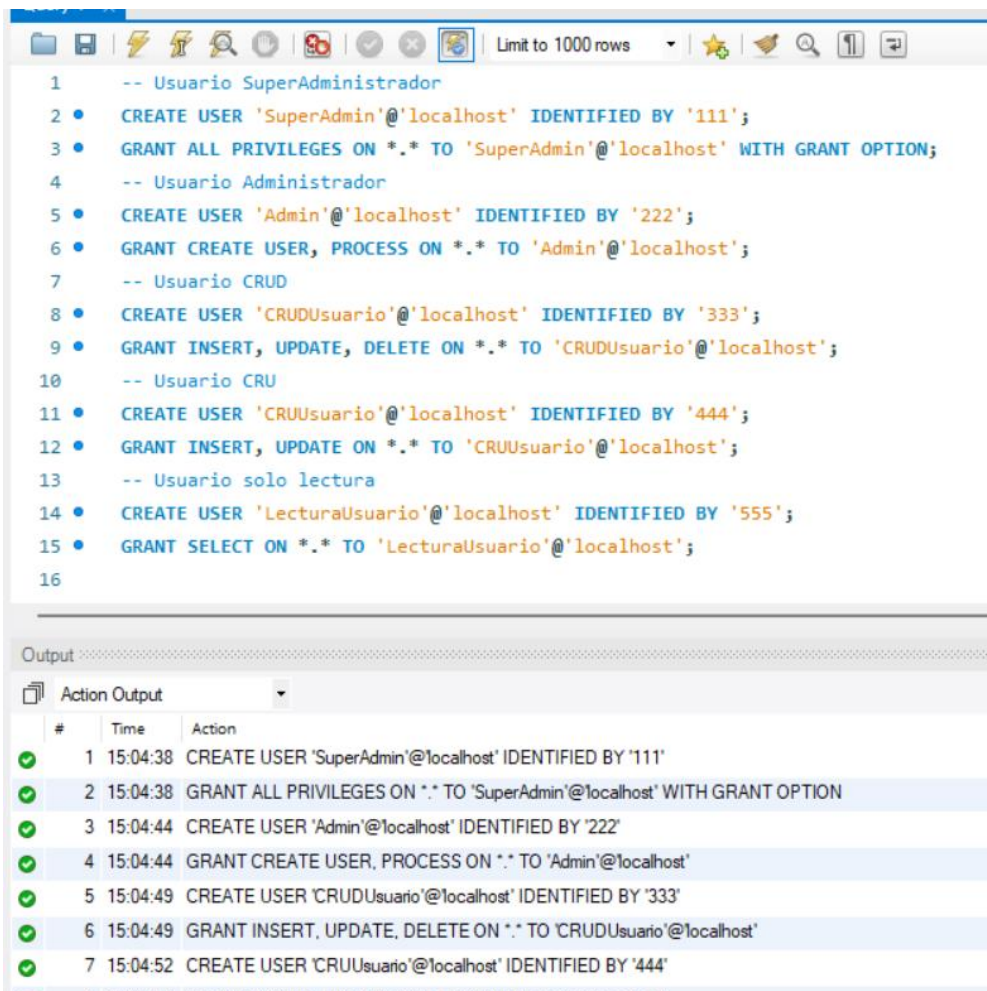
Creación del Script SQL

Escribe un script SQL que cree los cinco usuarios

Asegúrate de agregar comentarios que expliquen qué puede hacer cada usuario.

Usa contraseñas seguras y personalízalas si es necesario en la creación de usuarios.

- Inicia sesión con un usuario que tenga privilegios de super administrador (por ejemplo, `root`).
- CREAR ROLES
- Super Administrador: Crear y eliminar bases de datos.
- Administrador: Crear usuarios y procesos.
- CRUD: Insertar, actualizar y eliminar datos.
- CRU: Insertar y actualizar, pero sin eliminar.
- Solo Lectura: Realizar consultas a las tablas.



```
1  -- Usuario SuperAdministrador
2  ● CREATE USER 'SuperAdmin'@'localhost' IDENTIFIED BY '111';
3  ● GRANT ALL PRIVILEGES ON *.* TO 'SuperAdmin'@'localhost' WITH GRANT OPTION;
4  -- Usuario Administrador
5  ● CREATE USER 'Admin'@'localhost' IDENTIFIED BY '222';
6  ● GRANT CREATE USER, PROCESS ON *.* TO 'Admin'@'localhost';
7  -- Usuario CRUD
8  ● CREATE USER 'CRUDUsuario'@'localhost' IDENTIFIED BY '333';
9  ● GRANT INSERT, UPDATE, DELETE ON *.* TO 'CRUDUsuario'@'localhost';
10 -- Usuario CRU
11 ● CREATE USER 'CRUUsuario'@'localhost' IDENTIFIED BY '444';
12 ● GRANT INSERT, UPDATE ON *.* TO 'CRUUsuario'@'localhost';
13 -- Usuario solo lectura
14 ● CREATE USER 'LecturaUsuario'@'localhost' IDENTIFIED BY '555';
15 ● GRANT SELECT ON *.* TO 'LecturaUsuario'@'localhost';
16
```

Output

#	Time	Action
✓ 1	15:04:38	CREATE USER 'SuperAdmin'@'localhost' IDENTIFIED BY '111'
✓ 2	15:04:38	GRANT ALL PRIVILEGES ON *.* TO 'SuperAdmin'@'localhost' WITH GRANT OPTION
✓ 3	15:04:44	CREATE USER 'Admin'@'localhost' IDENTIFIED BY '222'
✓ 4	15:04:44	GRANT CREATE USER, PROCESS ON *.* TO 'Admin'@'localhost'
✓ 5	15:04:49	CREATE USER 'CRUDUsuario'@'localhost' IDENTIFIED BY '333'
✓ 6	15:04:49	GRANT INSERT, UPDATE, DELETE ON *.* TO 'CRUDUsuario'@'localhost'
✓ 7	15:04:52	CREATE USER 'CRUUsuario'@'localhost' IDENTIFIED BY '444'

Verificación de Permisos

Usa el comando `SHOW GRANTS FOR 'usuario'@'localhost';` para verificar qué permisos tiene cada usuario.

The image displays four screenshots of a SQL client interface showing the results of the `SHOW GRANTS` command for different users. Each screenshot includes the command entered in the query editor and the resulting permissions listed in a table.

Screenshot 1: SuperAdmin@localhost

```
17 • SHOW GRANTS FOR 'SuperAdmin'@'localhost';
```

Grants for SuperAdmin@localhost
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, SHUTDOWN, PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE, CREATE ROLE, DROP ROLE ON *.* TO 'SuperAdmin'@'localhost' WITH GRANT OPTION
GRANT ALLOW_NONEXISTENT_DEFINER, APPLICATION_PASSWORD_ADMIN ON *.* TO 'SuperAdmin'@'localhost'

Screenshot 2: Admin@localhost

```
17 • SHOW GRANTS FOR 'Admin'@'localhost';
```

Grants for Admin@localhost
GRANT PROCESS, CREATE USER ON *.* TO 'Admin'@'localhost'

Screenshot 3: CRUDUsuario@localhost

```
17 • SHOW GRANTS FOR 'CRUDUsuario'@'localhost';
```

Grants for CRUDUsuario@localhost
GRANT INSERT, UPDATE, DELETE ON *.* TO 'CRUDUsuario'@'localhost'

Screenshot 4: CRUUsuario@localhost

```
17 • SHOW GRANTS FOR 'CRUUsuario'@'localhost';
```

Grants for CRUUsuario@localhost
GRANT INSERT, UPDATE ON *.* TO 'CRUUsuario'@'localhost'

Screenshot 5: LecturaUsuario@localhost

```
17 • SHOW GRANTS FOR 'LecturaUsuario'@'localhost';
```

Grants for LecturaUsuario@localhost
GRANT SELECT ON *.* TO 'LecturaUsuario'@'localhost'

PARTE 2 TRIGGERS

Objetivo:

Comprender la importancia de los *triggers* en bases de datos, cómo se aplican en diferentes escenarios, y reconocer áreas en las que su uso es crucial para la integridad de los datos y el control de los procesos.

Instrucciones:

1. Investigación sobre los *Triggers*:

Conceptos clave sobre los Triggers

2. Tipos de triggers:

- **BEFORE:** Se ejecuta **antes** de que se realice una acción en la base de datos (INSERT, UPDATE, DELETE). Son útiles para validar datos antes de que se efectúe el cambio en la tabla.
- **AFTER:** Se ejecuta **después** de que la acción se haya realizado (INSERT, UPDATE, DELETE). Es útil cuando quieres registrar cambios o ejecutar acciones adicionales después de que se haya completado una operación.
- **INSTEAD OF:** Se utiliza principalmente en vistas, reemplazando la acción que se habría realizado por otra. Por ejemplo, en una vista, puedes usar un trigger **INSTEAD OF** para manejar **INSERT**, **UPDATE** o **DELETE** en lugar de modificar directamente las vistas.

3. Eventos que pueden activar un trigger:

- **INSERT:** Cuando se agrega un nuevo registro a la tabla.
- **UPDATE:** Cuando se modifica un registro existente en la tabla.
- **DELETE:** Cuando se elimina un registro de la tabla.

4. Contexto de los triggers:

- **NEW:** En triggers de tipo **INSERT** o **UPDATE**, puedes utilizar la palabra clave **NEW** para hacer referencia a los valores que van a insertarse o actualizarse en una fila. Es decir, los nuevos valores de una columna.
- **OLD:** En triggers de tipo **DELETE** o **UPDATE**, puedes utilizar la palabra clave **OLD** para hacer referencia a los valores anteriores de una fila antes de la modificación o eliminación.

AMPLIAR INFORMACIÓN Y ENTENDER

- Definición y funcionamiento: Investigar qué son los *triggers* en bases de datos, cómo funcionan, y los diferentes tipos de *triggers* (por ejemplo,

BEFORE, AFTER, INSTEAD OF.

¿Que son los triggers?

Son procedimientos almacenados que se ejecutan automáticamente cuando ciertos eventos en una tabla o vista son ejecutados y de tipo INSERT, UPDATE o DELETE.

- Importancia de su uso: Explicar por qué es importante usar *triggers* en una base de datos y cuáles son los beneficios que aportan, tales como la automatización de tareas, la integridad referencial, el control de cambios, entre otros.

Los triggers se utilizan para mantener la integridad de los datos, manejar cambios, y aplicar condiciones en operaciones automáticamente esto reduce la necesidad de intervención manual y minimiza errores humanos.

Los triggers pueden registrar automáticamente cualquier cambio en los datos, lo que es útil para auditorías y seguimiento de modificaciones.

- Ventajas y desventajas: Reflexionar sobre las ventajas (por ejemplo, evitar la corrupción de datos, asegurar reglas de negocio) y desventajas (por ejemplo, sobrecarga en el rendimiento) de utilizar *triggers*.

Ventajas:

- Se pueden sincronizar datos entre diferentes tablas o bases de datos, manteniendo la consistencia en sistemas distribuidos.
- Al realizar operaciones directamente en la base de datos, pueden ser más eficientes que hacerlo a nivel de aplicación, mejorando el rendimiento general del sistema.

Desventajas:

- Depurar y mantener triggers puede ser complicado, ya que se ejecutan automáticamente y pueden no ser evidentes en el flujo normal de la aplicación.
- Si no se diseñan cuidadosamente, los triggers pueden afectar negativamente el rendimiento de la base de datos, especialmente si realizan operaciones costosas o se ejecutan con frecuencia.

5. Aplicaciones de *Triggers*:

- Áreas de aplicación: Identificar en qué áreas de una base de datos se aplican *triggers*. Ejemplos comunes incluyen la validación de datos, la auditoría, el seguimiento de cambios y la implementación de reglas de negocio automáticas.

Se pueden aplicar a los siguientes campos en base de datos:

- Auditoría y seguimiento de cambios
- Mantenimiento de la integridad referencial
- Aplicación de reglas de negocio
- Seguridad y control de acceso
- Casos de uso específicos: Investigar ejemplos reales de empresas o sistemas que utilicen *triggers* para gestionar procesos como auditorías de registros, actualizaciones automáticas de información, control de integridad referencial, entre otros.

Enunciado de la Práctica:

Objetivo:

Crear un **trigger** que registre todas las operaciones (insert, update, delete) realizadas en una tabla de empleados en una tabla de auditoría. El objetivo es llevar un historial detallado de las acciones realizadas, incluyendo el tipo de operación, los datos afectados y la fecha y hora en que ocurrió cada cambio.

Descripción del Ejercicio:

Imagina que tienes una empresa que desea llevar un control detallado sobre los cambios realizados en los registros de sus empleados. Para ello, se necesita crear una tabla de auditoría que registre cualquier acción (inserción, actualización o eliminación) que se realice en la tabla de **Empleados**. Cada vez que se realice una operación sobre la tabla de empleados, el sistema debe registrar la siguiente información en la tabla de auditoría:

- Tipo de operación realizada (INSERT, UPDATE, DELETE)
- ID del empleado afectado
- Nombre y departamento del empleado
- Salario del empleado
- Fecha y hora en que se realizó la operación

Pasos para la práctica:

1. **Crear la tabla de empleados** con los siguientes campos:

- `EmpID` (ID del empleado)
- `Nombre` (Nombre del empleado)
- `Departamento` (Departamento en el que trabaja el empleado)

- **Salario** (Salario del empleado)

```

20 • create table empleados (
21     EmpID INT PRIMARY KEY,
22     Nombre VARCHAR(50),
23     Departamento VARCHAR(50),
24     Salario decimal (10,2)
25 );
26

```

Output		
Action Output		
#	Time	Action
✓ 1	13:06:53	use db1
✓ 2	13:13:30	create table empleados (EmpID INT PRIMARY KEY, Nombre VARCHAR(50), Departamento VARCHAR(...

2. Crear la tabla de auditoría con los siguientes campos:

- **AudID** (ID del registro de auditoría)
- **Accion** (Tipo de operación: INSERT, UPDATE, DELETE)
- **EmpID** (ID del empleado afectado)
- **Nombre** (Nombre del empleado)
- **Departamento** (Departamento del empleado)
- **Salario** (Salario del empleado)
- **Fecha** (Fecha y hora de la operación)

```

27 • create table auditoria(
28     AudID INT PRIMARY KEY AUTO_INCREMENT,
29     Accion VARCHAR(10),
30     EmpID INT,
31     Nombre VARCHAR(100),
32     Departamento VARCHAR(100),
33     Salario DECIMAL(10,2),
34     Fecha DATE,
35     FOREIGN KEY (EmpID) REFERENCES empleados(EmpID)
36 );

```

Output		
Action Output		
#	Time	Action
✓ 1	13:06:53	use db1
✓ 2	13:13:30	create table empleados (EmpID INT PRIMARY KEY, Nombre VARCHAR(50), Departamento VARCHAR(...
✓ 3	13:13:35	create table auditoria(AudID INT PRIMARY KEY AUTO_INCREMENT, Accion VARCHAR(10), EmpID INT, ..

3. Crear el trigger:

- El trigger debe activarse **después** de realizar cualquier operación (INSERT, UPDATE o DELETE) sobre la tabla de empleados. El trigger debe insertar un nuevo registro en la tabla de auditoría cada vez que se realice una de estas operaciones.

Requerimientos:

- El trigger debe registrar correctamente el tipo de operación realizada (INSERT, UPDATE, DELETE).
- El trigger debe almacenar el nombre del empleado, su departamento y salario.
- El trigger debe capturar la fecha y hora de la operación.
- [Crear el trigger para auditar eliminaciones Y Ver los cambios realizados](#)

```
18  -- Trigger
19  DELIMITER //
20  ● -- Despues de insertar
21  CREATE TRIGGER auditoria_empleados
22  AFTER INSERT ON empleados
23  FOR EACH ROW
24  ○ BEGIN
25      INSERT INTO auditoria (Accion, EmpID, Nombre, Departamento, Salario, Fecha)
26      VALUES ('INSERT', NEW.EmpID, NEW.Nombre, NEW.Departamento, NEW.Salario, NOW());
27  END; //
28  -- Despue de actualizar datos
29  ● CREATE TRIGGER auditoria_empleados_update
30  AFTER UPDATE ON empleados
31  FOR EACH ROW
32  ○ BEGIN
33      INSERT INTO auditoria (Accion, EmpID, Nombre, Departamento, Salario, Fecha)
34      VALUES ('UPDATE', NEW.EmpID, NEW.Nombre, NEW.Departamento, NEW.Salario, NOW());
35  END; //
36  -- Despues de eliminar datos
37  ● CREATE TRIGGER auditoria_empleados_delete
38  AFTER DELETE ON empleados
39  FOR EACH ROW
40  ○ BEGIN
41      INSERT INTO auditoria (Accion, EmpID, Nombre, Departamento, Salario, Fecha)
42      VALUES ('DELETE', OLD.EmpID, OLD.Nombre, OLD.Departamento, OLD.Salario, NOW());
43  END; //
```


Enlace GitHub:

<https://github.com/IsaacMorocho/Roles-Triggers>