Universidad Nacional Autónoma de México
Facultad de Ciencias

Fundamentos de Bases de Datos

Práctica 5

Procedimientos Almacenados, Funciones y Triggers

Autor: Arroyo Martínez Erick Daniel

Introducción

En esta práctica, se explorará el uso de procedimientos almacenados, funciones y triggers en MySQL. Estas herramientas son fundamentales para automatizar operaciones en bases de datos y mejorar la eficiencia en la ejecución de tareas repetitivas o que requieren lógica compleja.

Un **procedimiento almacenado** es un conjunto de instrucciones SQL que se pueden almacenar en la base de datos y ejecutar cuando sea necesario. Estos procedimientos permiten encapsular lógica compleja y reutilizarla en diferentes partes de una aplicación. Una **función** es similar a un procedimiento almacenado, pero tiene una característica clave: siempre devuelve un valor. Las funciones se utilizan cuando se necesita un resultado específico a partir de una operación y pueden ser invocadas dentro de otras sentencias SQL, como parte de consultas. Un **trigger** es una operación que se ejecuta automáticamente en respuesta a eventos en una tabla, como la inserción, actualización o eliminación de datos. Los triggers se utilizan para asegurar que se cumplan las reglas de negocio o para mantener la integridad de los datos.

A continuación, se presentan las estructuras básicas para definir procedimientos almacenados, funciones y triggers en MySQL, para los distinos DBMS[1] bastaría traducir a su sintaxis.

Estructura Base para Procedimientos Almacenados

La sintaxis básica para declarar un procedimiento almacenado en MySQL es la siguiente:

Sitaxis básica

```
CREATE PROCEDURE procedure_name (IN|OUT|INOUT param_name datatype, ...)
BEGIN
    -- Instrucciones SQL
END;
```

[1] Del inglés: Data Base Management System

Ejemplo concreto

Ejemplo Concreto

```
DELIMITER //

CREATE PROCEDURE AddEmployee(
    IN emp_name VARCHAR(100),
    IN emp_salary DECIMAL(10, 2),
    OUT new_emp_id INT
)
BEGIN
    INSERT INTO employees (name, salary)
    VALUES (emp_name, emp_salary);

    SET new_emp_id = LAST_INSERT_ID();
END //

DELIMITER ;
```

- `IN`: El parámetro es de solo entrada. Los valores son proporcionados por el usuario o la aplicación.
- `OUT`: El parámetro es de solo salida. Los valores serán devueltos por el procedimiento.
- `INOUT`: El parámetro puede ser tanto de entrada como de salida.

El bloque `BEGIN ... END` contiene las instrucciones SQL que conforman el cuerpo del procedimiento.

Estructura Base para Funciones

La sintaxis básica para declarar una función en MySQL es la siguiente:

Sitaxis básica

```
CREATE FUNCTION function_name (param_name datatype, ...)
RETURNS return_datatype
DETERMINISTIC | NOT DETERMINISTIC
BEGIN
    -- Instrucciones SQL
    RETURN value;
END;
```

Ejemplo concreto

Ejemplo Concreto

```
DELIMITER //

CREATE FUNCTION CalculateBonus(emp_salary DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE bonus DECIMAL(10, 2);

    IF emp_salary > 5000 THEN
        SET bonus = emp_salary * 0.10;  -- 10% bonus
    ELSE
        SET bonus = emp_salary * 0.05;  -- 5% bonus
    END IF;

    RETURN bonus;
END //

DELIMITER ;
```

- `RETURNS`: Define el tipo de dato que la función devolverá.
- `DETERMINISTIC | NOT DETERMINISTIC`: Indica si la función siempre devuelve el mismo resultado para un conjunto dado de entradas (`DETERMINISTIC`) o si puede variar (`NOT DETERMINISTIC`).

El bloque `BEGIN ... END` define el cuerpo de la función, y siempre debe contener una instrucción `RETURN` que especifique el valor a devolver.

Estructura Base para Triggers

La sintaxis básica para declarar un trigger en MySQL es la siguiente:

Sitaxis básica

```
CREATE TRIGGER trigger_name_{acrónimo}
{BEFORE|AFTER} {INSERT|UPDATE|DELETE}
ON table_name FOR EACH ROW
BEGIN
    -- Instrucciones SQL
END;
```

- `BEFORE | AFTER`: Define si el trigger debe ejecutarse antes o después de la operación.
- `INSERT | UPDATE | DELETE`: Especifica el tipo de operación que activa el trigger.
- `FOR EACH ROW`: El trigger se ejecuta una vez por cada fila afectada por la operación.

El bloque `BEGIN ... END` contiene las instrucciones SQL que se ejecutan cuando el trigger es activado.

Ejemplo concreto

Ejemplo Concreto

```
DELIMITER //

CREATE TRIGGER BeforeEmployeeDelete
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO deleted_employees (employee_id, name, deleted_at)
    VALUES (OLD.id, OLD.name, NOW());
END //

DELIMITER ;
```

Objetivos

Al finalizar esta práctica, el estudiante será capaz de:

1. Comprender la estructura básica y los componentes de los procedimientos almacenados, funciones y triggers en MySQL.
2. Implementar procedimientos almacenados que reciban parámetros de entrada, salida y mixtos.
3. Crear funciones que realicen cálculos y devuelvan resultados de manera eficiente dentro de consultas SQL.
4. Desarrollar triggers que se activen ante eventos como inserciones, actualizaciones o eliminaciones, para garantizar la integridad y consistencia de los datos.
5. Aplicar las diferencias entre procedimientos almacenados y funciones en casos prácticos y entender sus usos más adecuados.

Especificaciones de Desarrollo

A continuación, se deben implementar tres procedimientos almacenados, tres funciones y tres triggers sobre el esquema dado de la práctica 4. Cada uno de estos elementos debe estar interrelacionado para simular un entorno real de trabajo con bases de datos. Se proporcionará un esquema básico, pero los alumnos deberán completar la lógica y garantizar la integridad en la base de datos.

Procedimientos Almacenados

1. **Procedimiento 1: AddNewEmployee**
   Este procedimiento almacenado permite agregar un nuevo empleado y, opcionalmente, asignarlo como gerente de un departamento. Se verifica que el departamento no tenga ya un gerente.
2. **Procedimiento 2: AssignEmployeeToProject**
   Este procedimiento asigna a un empleado a un proyecto. Además, valida que el empleado no trabaje más de 40 horas en todos los proyectos combinados.

3. **Procedimiento 3: AddCustomerToProject**
   Este procedimiento asigna un cliente a un proyecto, validando que el cliente no esté vinculado a más de tres proyectos activos al mismo tiempo.

Funciones

1. **Función 1: CalculateTotalHoursWorked**
   Calcula el total de horas trabajadas por un empleado en todos los proyectos a los que está asignado.
2. **Función 2: GetProjectBudgetRemaining**
   Calcula el presupuesto restante de un proyecto teniendo en cuenta los pagos realizados a los proveedores asociados.
3. **Función 3: IsDepartmentManager**
   Verifica si un empleado es gerente de su departamento.

Triggers

1. **Trigger 1: BeforeEmployeeDelete**
   Evita la eliminación de un empleado si este es gerente de algún departamento.
2. **Trigger 2: AfterInsertAssignment**
   Se activa después de que se inserte una asignación de proyecto, invocando la función *CalculateTotalHoursWorked* para verificar que el empleado no exceda las 40 horas.
3. **Trigger 3: BeforeUpdateProjectBudget**
   Verifica, antes de actualizar el presupuesto de un proyecto, si los pagos exceden el presupuesto original utilizando la función *GetProjectBudgetRemaining*.

1. Entregables

- Código SQL con la implementación de las tres funciones, tres procedimientos almacenados y tres triggers.

2. Rúbrica de Evaluación

Criterio	**Puntos**
Implementación correcta de las funciones	30 %
Implementación correcta de los procedimientos almacenados	30 %
Implementación correcta de los triggers	30 %
Claridad y presentación del código	10 %

Recursos

- Live SQL
- SQL Language Reference
- SQL Tutorial
- mockaroo
- MySQL Documentation