PROCEDIMIENTOS ALMACENADOS.
EVENTOS

Diseño y Programación de una BBDD

ÍNDICE

ÍNDICES Y OPTIMIZACIÓN DE CONSULTAS

1. PROCEDIMIENTOS ALMACENADOS (Stored Procedures)

1.1 Explica qué es un procedimiento almacenado y en qué se diferencia de una función UDF y de un disparador.

Los procedimientos almacenados son instrucciones SQL predefinidas y compiladas que se almacenan en la base de datos para se ejecuten cuando sea necesario. Son capaces de aceptar parámetros de entrada y devolver resultados a través de parámetros de salida. Además, tienen la capacidad de modificar datos directamente en la base de datos.

Son extremadamente útiles para realizar operaciones complejas que involucran múltiples consultas, actualizaciones e incluso manipulación de datos, ya que es posible llamarlos varias veces desde distintas partes de la aplicación.

Uno de sus puntos positivos son que mejoran el rendimiento de la aplicación ya que se ejecutan en el servidor, por ello, son capaces de minimizar el uso de la red.

Diferencia con una Función UDF:

La principal diferencia que tiene un procedimiento almacenado con una función UDF es que esta última devuelve un valor cuando se ejecuta.

Aparte, los procedimientos almacenados se aplican con “EXECUTE” o “CALL y las UDF se utilizan en la sentencia SQL.

Por otro lado, un procedimiento almacenado utilizando instrucciones DML, puede llegar a modificar el estado de la BBDD, mientras que las funciones UDF solo operan en lectura.

Diferencia con Disparador “Trigger”:

La diferencia que tiene con un disparador es que el disparador se ejecuta automáticamente cuando se cumplen una serie de condiciones. También hay que tener en cuenta que un disparador se vincula directamente con una table mientras que los procedimientos almacenados se llaman desde la BBDD.

A la hora de ejecutarse también se diferencian en que los disparadores lo hacen cuando se ejecuta una operación específica, y los procedimientos almacenados en la sesión del usuario.

1.2 Explica cuál es la sintaxis básica para crear un procedimiento almacenado.

```
DELIMITER //

CREATE PROCEDURE nombre_procedimiento(IN parametro1 tipo_de_dato, IN parametro2 tipo_de_dato)

BEGIN

    -- Cuerpo del procedimiento almacenado

    -- Aquí se incluyen las instrucciones SQL

    -- Ejemplo: Consulta SELECT

    SELECT columna1, columna2



    FROM nombre_tabla

    WHERE condicion = parametro1;



    -- Ejemplo: Inserción de datos

    INSERT INTO otra_tabla (columna1, columna2)

    VALUES (parametro1, parametro2);

    -- Otras instrucciones SQL...

END //

DELIMITER ;
```

Delimiter // : se indica un nuevo delimitador para que el punto y coma (‘;’) no finalice la creación del procedimiento almacenado.

Create Procedure nombre_procedimiento: se define el nombre del procedimiento almacenado.

IN parámetro tipo_de_dato: se indica opcionalmente si el procedimiento almacenado se añadirá una entrada.

BEGIN…END: se define el contenido del procedimiento almacenado, siendo el lugar donde se indican las instrucciones SQL. Puede haber instrucciones de SELECT, UPDATE, INSERT, DELETE entre otras.

1.3 ¿Qué tipos de parámetros puede tener un Procedimiento Almacenado? ¿Cuál es la diferencia entre ellos?

Pueden ser de tres tipos, parámetros de entrada, de salida o de entrada/salida.

Los parámetros de entrada permiten introducir valores desde el exterior del procedimiento almacenado. Estos parámetros se pueden modificar dentro del procedimiento almacenado pero los cambios realizados en ellos no se reflejarán fuera del procedimiento. Se definen con la palabra clave IN.

En segundo lugar, nos encontramos los parámetros de salida OUT. Son utilizados para devolver valores fuera del procedimiento almacenado. Estos valores se asignan dentro del procedimiento almacenado y pueden usar utilizados una vez el procedimiento se ha ejecutado.

Por último, indicar los parámetros de entrada y salida. Como su nombre indica son aquellos parámetros que actúan tanto como parámetros de entrada como de salida. Se utilizan para pasar valores dentro y fuera del procedimiento almacenado.

1.4 ¿Cómo invocamos a un Procedimiento Almacenado para ejecutarlo?

Con parámetros de entrada:

```
CALL mi_procedimiento(10, 20);
```

Con parámetros de salida:

```
CALL procedimiento_con_salida(@resultado);

SELECT @resultado AS resultado_obtenido;
```

1.5 ¿Con cuál sentencia se elimina una Procedimiento Almacenado?

La sentencia para eliminar un procedimiento almacenado es la siguiente:

```
DROP PROCEDURE IF EXISTS nombre_procedimiento;
```

2. ESTRUCTURAS DE CONTROL

2.1 Explicar e ilustrar con un ejemplo (para cada instrucción): Cómo se utilizan las instrucciones condicionales IF y CASE dentro de un Procedimiento Almacenado

Las instrucciones condicionales IF y CASE permiten ejecutar lógica condicional basada en diferentes condiciones.

Ejemplo de IF:

```
DELIMITER //

CREATE PROCEDURE mostrar_experiencia(IN nombre_mecanico VARCHAR(50))

BEGIN

    DECLARE experiencia_mecanico INT;

    SELECT experiencia INTO experiencia_mecanico
```

```
    FROM mecanicos WHERE nombre = nombre_mecanico;

    IF experiencia_mecanico >= 5 THEN

        SELECT CONCAT(nombre_mecanico, ' tiene mucha experiencia.') AS mensaje;

    ELSE

        SELECT CONCAT(nombre_mecanico, ' tiene experiencia moderada o baja.') AS
mensaje;

    END IF;

END //

DELIMITER ;
```

Ejemplo de CASE:

```
DELIMITER //

CREATE PROCEDURE asignar_bono(IN nombre_mecanico VARCHAR(50), OUT bono
DECIMAL(10, 2))

BEGIN

    SELECT

        CASE especialidad

            WHEN 'Motor' THEN bono := 500.00

            WHEN 'Suspensión' THEN bono := 400.00

            WHEN 'Frenos' THEN bono := 300.00

            ELSE bono := 200.00
```

```
        END
    INTO bono
    FROM mecanicos WHERE nombre = nombre_mecanico;
END //
DELIMITER ;
```

2.2 Explicar e ilustrar con un ejemplo (para cada instrucción): Cómo se utilizan los bucles LOOP, REPEAT y WHILE dentro de un Procedimiento Almacenado

Ejemplo con LOOP se utiliza para ejecutar repetidamente mostrando el nombre de los mecánicos de que tengan una experiencia igual o superior a la experiencia mínima.

```
DELIMITER //
CREATE PROCEDURE mostrar_mecanicos_con_experiencia(IN experiencia_minima INT)
BEGIN
    DECLARE contador INT DEFAULT 1;
    DECLARE max_id INT;
    SELECT MAX(id) INTO max_id FROM mecanicos;
    LOOP
        IF contador > max_id THEN
            LEAVE;
        END IF;
        DECLARE nombre_mecanico VARCHAR(50);
```

```
        SELECT nombre INTO nombre_mecanico
        FROM mecanicos WHERE id = contador AND experiencia >= experiencia_minima;
        IF nombre_mecanico IS NOT NULL THEN
            SELECT nombre_mecanico AS mecanico_con_experiencia;
        END IF;
        SET contador = contador + 1;
    END LOOP;
END //
DELIMITER ;
```

Ejemplo REPEAT para mostrar los nombres de aquellos mecánicos que tengan una experiencia menor o igual a la experiencia máxima.

```
DELIMITER //
CREATE PROCEDURE mostrar_mecanicos_menos_experiencia(IN experiencia_maxima INT)
BEGIN
    DECLARE contador INT DEFAULT 1;
    DECLARE max_id INT;
    SELECT MAX(id) INTO max_id FROM mecanicos;
    REPEAT
        DECLARE nombre_mecanico VARCHAR(50);
```

```
        SELECT nombre INTO nombre_mecanico
        FROM mecanicos WHERE id = contador AND experiencia <= experiencia_maxima;
        IF nombre_mecanico IS NOT NULL THEN
            SELECT nombre_mecanico AS mecanico_con_menos_experiencia;
        END IF;
        SET contador = contador + 1;
    UNTIL contador > max_id END REPEAT;
END //
DELIMITER ;
```

2.3 Cuando trabajamos con procedimientos almacenados en MySQL tenemos que tener en cuenta que durante la ejecución de nuestra aplicación se pueden producir errores. Para llevar el control de estos errores podemos definir "handlers" (controladores de error) en nuestros procedimientos almacenados. Responder:

- **¿Cuál es la sintaxis de los controladores de error?**

  La sintaxis básica para tener controlados los errores es la siguiente:

  ```
  DECLARE mi_handler CONTINUE HANDLER FOR SQLEXCEPTION
  ```

- **¿Qué acciones se pueden tomar al encontrar un error y con qué sentencias se realizan estas acciones?**

  Puede tener tres acciones, la primera continuar ('CONTINUE') la ejecución tras el error:

  ```
  DECLARE nombre_handler CONTINUE HANDLER FOR tipo_error ACTION
  conjunto_sentencias;
  ```

  La segunda sería salir ('EXIT') de procedimiento almacenado:

DECLARE nombre_handler EXIT HANDLER FOR tipo_error ACTION conjunto_de_sentencias;

La tercera sería deshacer (‘UNDO’) la operación:

DECLARE nombre_handler UNDO HANDLER FOR tipo_error ACTION conjunto_de_sentencias;

- **¿Cuáles son las posibles condiciones de error?**

  Las posibles condiciones de error son varias en MYSQL, las indicamos a continuación:

  - **SQLEXCEPTION:** recoge cualquier error SQL genérico que ocurra durante la ejecución.
  - **SQLWARNING:** recoge advertencias SQL que pueden surgir durante la ejecución.
  - **NOTFOUND:** se activa cuando una consulta no existe.
  - **DUP_VAL_ON_INDEX:** se activa cuando se realiza un Insert o actualización de registro con un valor que viola la restricción de índice único o clave primaria.
  - **TOO_MANY_ROWS:** se produce cuando una consulta devuelve más de una fila en un contexto donde se espera un solo valor.
  - **STRICT_TRANS_TABLE:** captura los errores relacionados con las configuraciones del modo estricto, como intentar insertar un valor no valido en una columna.
  - **DATA_TRUNCATION:** ocurre cuando se produce una perdida de datos debido a un truncamiento durante una operación de asignación o conversión.
  - **ER_LOCK_DEADLOCK:** ocurre cuando dos o más transacciones no pueden avanzar porque cada una espera un bloqueo que la otra tiene.

3. CURSORES

3.1 ¿Qué es un cursor y qué nos permiten realizar los cursores?

Un cursor permite recorrer y manipular filas de un conjunto de resultados, normalmente son a través de las consultas, permitiendo realizar operaciones específicas como seleccionar datos basados en ciertos criterios o condiciones.

Tienen la capacidad de proporcionar un nivel de control más granular sobre los datos y son útiles cuando se necesita trabajar cada fila de manera individual. Hay que tener en cuenta que puede afectar al rendimiento, especialmente cuando se utilizan para recorrer una gran cantidad de datos, es por ello que se requiere de más recursos en comparación con otras operaciones.

3.2 ¿Cuál es secuencia de instrucciones para trabajar con los cursores?

Declarar el cursor para definirlo con las necesidades oportunas:

DECLARE nombre_cursor CURSOR FOR SELECT columna1, columna2 FROM tabla WHERE condicion;

Apertura del cursor para ejecutar la consulta definida en el cursor:

OPEN nombre_cursor;

Recorrer las filas para procesar cada fila:

FETCH nombre_cursor INTO variable1, variable2;

Procesamiento de datos con un bucle y finalmente cerrar el cursor con CLOSE:

CLOSE nombre_cursor;

3.3 Cuando se está recorriendo un cursor y no quedan filas por recorrer se lanza el error NOT FOUND, que se corresponde con el valor SQLSTATE '02000'. ¿Cómo se puede manejar este error?

Declarando una variable que cuando no haya más filas se cambie el valor a TRUE para que salga del bucle

DECLARE exit_loop BOOLEAN DEFAULT FALSE;

4. TRANSACCIONES EN PROCEDIMIENTOS ALMACENADOS

4.1 ¿Cómo se pueden incluir transacciones dentro de Procedimientos Almacenados y qué ventajas tiene incluirlas?

Principalmente se utilizan para agrupar un conjunto de operaciones en SQL en una unidad lógica y asegurar la integridad de los datos. De esta forma se permite tener una consistencia en los datos ya que, si una operación falla, todas las operaciones realizadas hasta el momento pueden revertirse. Proporcionan niveles de aislamientos que controlan como las operaciones de diferentes transacciones pueden interactuar. Ayudan a gestionar las concurrencias cuando múltiples usuarios acceden simultáneamente a la base de datos, permitiendo un control de bloqueos y de esta forma evitar conflictos entre distintas operaciones.

4.2 Ilustrar un ejemplo, con un trozo de código, donde se incluya una transacción en un Procedimiento Almacenado.

```
DELIMITER //

CREATE PROCEDURE registrar_reparacion(

    IN vehiculo_id INT,

    IN descripcion VARCHAR(255),

    IN repuesto_utilizado VARCHAR(100),

    IN cantidad_utilizada INT

)

BEGIN

    DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING

    BEGIN

        -- Manejo de errores: se realiza un ROLLBACK si ocurre algún error
```

```
        ROLLBACK;
        SELECT 'Error en la transacción';
    END;
    START TRANSACTION;
    -- Registrar la reparación en la tabla 'reparaciones'
    INSERT INTO reparaciones (vehiculo_id, descripcion)
    VALUES (vehiculo_id, descripcion);
    -- Actualizar el inventario de repuestos
    UPDATE repuestos
    SET cantidad = cantidad - cantidad_utilizada
    WHERE nombre = repuesto_utilizado;
    -- Confirmar la transacción si todo está bien
    COMMIT;
    SELECT 'Reparación registrada exitosamente';
END //
DELIMITER ;
```

5. EVENTOS

5.1 Definir que es un evento (EVENT) de MySQL.

Es una tarea programada que se ejecuta automáticamente según un horario indicado. De esta forma nos permite automatizar tareas repetitivas dentro de la base de datos como por ejemplo copias de seguridad, actualización de datos, generación de informes, mantenimientos como la optimalización de tablas, administración de sesiones y envío de notificaciones entre otros.

5.2 ¿Cómo se habilita el servidor de Bases de Datos para que puedan ejecutar eventos?

La primera acción que hay que realizar es la de verificar que el planificador de eventos este habilitado:

```
SHOW VARIABLES LIKE 'event_scheduler';
```

En el caso de que se encuentre deshabilitado se habilitara de la siguiente forma:

```
SET GLOBAL event_scheduler = ON;
```

Sin embargo, esta configuración se perderá tras un reinicio. Por lo que si necesitamos que persista se deberá de modifica el fichero en el sistema Linux o carpeta de instalación en Windows.

```
[mysqld]

event_scheduler = ON
```

5.3 Mostrar sintaxis para crear un evento explicando cada una de las líneas que la conforman.

Esta es la sentencia para crear un evento:

```
DELIMITER //

CREATE EVENT nombre_evento
```

```
  ON SCHEDULE
    EVERY 1 DAY
    STARTS CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    COMMENT 'Descripción del evento'
  DO
    BEGIN
      -- Acciones a realizar dentro del evento
      INSERT INTO tabla (columna1, columna2) VALUES (valor1, valor2);
      UPDATE otra_tabla SET columna = nuevo_valor WHERE condicion;
    END //
DELIMITER ;
```

CREATE Event: Para iniciar la creación del evento.

ON SCHEDULE: Para indicar cuando se ejecuta.

START CURRENT TIMESTAMP: Para indicar cuando comienza la ejecución por primera vez.

COMMENT: Permite indicar una anotación o descripción del evento.

DO: Indica el inicio del bloque de ejecución.

BEGIN…END: Encierra las acciones que se realizaran dentro del evento, se ejecutaran dos acciones INSERT inserción y UPDATE actualización.

5.4 ¿Cómo podemos listar los eventos guardados en la Base de Datos?

Con el comando “SHOW EVENTS”; se muestran los eventos guardados en la base de datos. Si necesitamos más detalles de un evento concreto con el comando **“SHOW CREATE EVENT nombre_envento”;**

5.5 ¿Con qué sentencia se elimina un evento y cómo se puede deshabilitar un evento?

Para eliminar un evento utilizaremos el siguiente comando:

DROP EVENT nombre_evento;

Si lo que necesitamos es deshabilitar un evento lo realizaremos con el siguiente comando:

ALTER EVENT nombre_evento DISABLE;

Para habilitar de nuevo el evento lo realizaremos con el siguiente comando:

ALTER EVENT nombre_evento ENABLE;