Procedimientos almacenados

Definición: un procedimiento almacenado es una colección con nombre de instrucciones que se almacena en el servidor. Son un método para encapsular tareas repetitivas. Admiten variables declaradas pro el usuario.

Ventaja:

- **Compartir** la lógica de la app con las restantes aplicaciones. Pueden encapsular la funcionalidad del negocio.
- **Apartar** a los usuarion de la exposición de los detalles de las tablas de la base de datos.
- **Proporcionar mecanismo de seguridad**
- **Mejorar el rendimiento**
- **Reducir el tráfico de red.**

Ejemplo:

```
Delimiter $$
    create procedure ingresarEquipo( pnombre varchar(50))
begin
    insert into equipo (nombre_equipo) value (pnombre);
end $$
```

Trigger

Un **Trigger** es un objeto de la base de datos que está asociado con una tabla y que se activa cuando ocurre un evento sobre la tabla.

Los eventos que pueden ocurrir sobre la tabla son:

- **INSERT**: el Trigger se activa cuando se inserta una nueva fila sobre la tabla asociada.
- **Update**:
- **Delete:**

Los eventos se pueden activar cuando se lo especifique, con las sentencias BEFORE O AFTER

Para recordar:

Se tiene acceso a través de new y Old a los valores de la fila afectada por el trigger.

Old solo existe en Triggers DELETE y UPDATE

New solo existe en triggers INSERT y UPDATE.

(1) Generar un trigger que evite la carga de nuevos jugadores

#before insert

Delimiter $$

create trigger TIB_evitar_cargar_jugadores before insert on jugadores for each row

begin

SIGNAL SQLSTATE '11111' SET MESSAGE_TEXT = 'No se puede Insertar Jugador', mysql_errno = '1000';

end

$$

Cursores

Definición: un cursor es un objeto de la base de datos usados por app para manipular los datos fila a fila en lugar de hacerlos en bloque de filas como los comandos normales. Puede ser usado internamente en la base de datos para realizar alguna operación registro a registro.

Ciclo de vida:

Para poder trabajar con cursores debemos realizar los siguientes pasos:

- Declarar el cursor con la instrucción DECLARE
- Abrir el cursor con la instrucción Open
- Declarar una variable para manejar
- Recuperar las filas desde el cursor de la instrucción FETCH
- Procesar las filas obtenidas.
- Cerrar el cursor con la instrucción CLOSE.

Modelo de cursores

1) **Declaración:**

   DECLARE <nombre cursor> CURSOR

   FOR

   <sentencia SQL> ;

   Ejemplo:

   DECLARE CUR_PERSONAS CURSOR FOR SELECT ID_PERSONA, NOMBRE, APELLIDO, DNI FROM PERSONAS; WHERE NACIONALIDAD = 'ARGENTINA'

2) **Declarar un not_found handler**
   Para poder verificar que el cursor ha traído datos, debemos declarar una variable que sea modificada con TRUE|FALSE según el resultado de la última operación.

   Para esto debemos declarar una variable booleana y luego asignarle a esa variable booleana la tarea de cambiar su valor en base a la última operación.
   DECLARE vFinished INTEGER DEFAULT 0;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET vFinished = 1;

Cuando una operación FETCH no encuentre datos, la variable vFinished cambiara a 1, por lo cual es sumamente importante inicializar está en 0.

3) Apertura

Para poder comenzar a utilizarla debemos abrir el cursor con la siguiente sintaxis:
OPEN <nombre cursor>;

En nuestro caso de ejemplo sería:

```
OPEN CUR_PERSONAS;
```

4) **Recuperación de datos:**

En este paso constará de recorrer los resultados del cursor, la instrucción FETCH permitirá efectuar dicha operación. Las filas podrán copiarse a variables ultilizando la sentencia INTO en combinación con la sentencia FETCH. Tiene q haber tantas variables como tantos registros se llaman en el SELECT.

```
FETCH CUR_PERSONAS INT vId, vNnombre, vApellido, vDni;
```

Recorrer un cursor

```
Get_personas: LOOP
        FETCH CUR_PERSONAS INT vId, vNnombre, vApellido, vDni;
        IF vFinished = 1 THEN
                LEAVE get_personas;
        End if;
        Select concat(vId, vNombre);
END LOOP get_personas;
```

5) Cierre de cursor

En el cierre del cursor se liberarán los registros tomados por el mismo, una vez que el cursor es cerrado ya no podrá recorrerse el conjunto de resultados hasta que el mismo sea reabierto.

Close <nombre cursor>;

```
Delimiter $$
        create procedure cur_jugadores_equipo ()
 begin
        declare vJugador varchar(50);
        declare vEquipo varchar(50);
        declare vFound  int default 0;
        declare  info_jugadores cursor
        for select j.nombre,  e.nombre_equipo
        from jugadores j
        inner join equipo e
        on j.fk_id_equipo = e.nombre_equipo;
        declare continue handler for not found set vFound = 1;
        open info_jugadores;
                ciclo : loop
                        fetch info_jugadores into vJugador, vEquipo;
                        if(vFound = 1) then
                                leave ciclo;
                        end if;
                        select vJugador, vEquipo;
                end loop ciclo;
end; $$
```

Transacciones

Definición: una transacción es una unidad única de trabajo. Si uan transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos. Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de lso datos.

Una transacción debe exhibir cuatro propiedades conocidas como (ACID)

- **Atomicidad**: Una transacción debe ser una unidad atómica de trabajo, tanto si se realizan todas sus modificaciones en los datos, como si no se realiza ninguna de ellas.
- **Coherencia**: Cuando finaliza, una transacción debe dejar todos los datos en un estado coherente

- **Aislamiento**: Las modificaciones realizadas por transacciones simultáneas se deben aislar de las modificaciones llevadas a cabo por otras transacciones simultáneas.
- **Durabilidad**: Una vez concluida una transacción, sus efectos son permanentes en el sistema.

Tipos de transacciones:

Explicitas: Son aquellas en que se define explícitamente el inicio y el final de la transacción. En un motor SQL utilizan las instrucciones :

START TRANSACTION: marca el punto de inicio

COMMIT: Se utiliza para finalizar una transacción correctamente si no hubo errores.

ROLLBACK: Se utiliza para eliminar una transacción en la que se encontraron errores.

```
START TRANSACTION
    update cuenta
     set monto = monto – 100
     where id_cuenta = 101;
     update cuenta
    set monto = monto + 100
    where id_cuenta = 102;
    COMMIT;
```

Implicitas: inicia automáticamente una nueva transacción después de confirmar o revertir la transacción actual. No tiene que realizar ninguna acción para delinear el inicio de una transacción, sólo tiene que confirmar o revertir cada transacción. El modo de transacciones implícitas genera una cadena continua de transacciones.

```
    update cuenta
     set monto = monto – 100
     where id_cuenta = 101;
     update cuenta
    set monto = monto + 100
    where id_cuenta = 102;
    COMMIT;
```

Estados de una transacción

Activa: la transacción permanece en este estado durante su ejecución

Parcialmente Comprometida: la transacción pasa a este estado cuando acaba de realiza la última instrucción,

Fallida: la transacción pasa a este estado tras descubrir que no puede continuar la ejecución normal.

Abortada: la transacción pasa a este estado después de haber restablecido la base de datos de su estado anterior.

Comprometida: la transacción pasa a este estado tras completarse con éxito.

Niveles de aislamiento:

Read uncommited: te muestra la última versión de los datos, incluso (pero no necesariamente) si proviene de una transacción aún no cometida.

Read Commited: te muestra la última versión cometida de los datos. Permite que entre dos lecturas de un mismo registro en una transacción A, otra transacción B pueda modificar dicho registro, obteniéndose diferentes resultados de la misma lectura

Repeatable Read: Evita que entre dos lecturas de un mismo registro en una transacción A, otra transacción B pueda modificar dicho registro, con el efecto de que en la segunda lectura de la transacción A se obtuviera un dato diferente.

Serealizable:

REPEATABLE READ y SERIALIZABLE: te muestran inicialmente la última versión cometida de los datos, y además te garantizan que, para todo lo que dure la transacción actual, siempre trabajarán con el valor que te devolvieron por primera vez para cada fila en esa transacción, incluso si luego hay una nueva versión cometida de esas filas. Para ver nuevas versiones cometidas tienes que terminar la transacción actual y abrir otra nueva.

Vistas

Definicion: podemos definir una vista como una consulta almacenada en la base de datos que se utiliza como una tabla virtual. Se trata de una perspectiva de la base de datos que permite a uno o varios usuarios ver solamente las filas y columnas necesarias para su trabajo.

Ventajas:

Centrar el interés en los datos de los usuarios: Las vistas crean un entorno controlado que permite el acceso a datos específicos mientras se oculta el resto.

Seguridad y confidencialidad: Las vistas ocultan al usuario la complejidad del diseño de la base de datos. De este modo, los programadores pueden cambiar el diseño sin afectar a la interacción entre el usuario y la base de datos. Además, el usuario verá una versión más descriptiva de los datos, con nombres más fáciles de comprender que los términos crípticos que a menudo se utilizan en las bases de datos.

Mejorar el rendimiento: Las vistas le permiten almacenar los resultados de consultas complejas. Otras consultas pueden utilizar estos resultados resumidos.

Sintaxis

```
create view view_name ([Lista de columnas])
as Select [campos]
from tabla
```

Índices

Definición: es una estructura de datos definida sobre una columna de tabla y que permite localizar de forma rápida las filas de la tabla de la base de datos a su contenido en la columna indexada además de permitir recuperar las filas de la tabla ordenadas por esa misma columna.

Mysql emplea los índices para las siguientes acciones:

- Encontrar las filas que cumplen con la condición WHERE
- Para recuperar las filas de otras tablas cuando se emplean operaciones de tipo Join . importante q el índice sea del mismo tipo
- Disminuir el tiempo de ejecución de las consultas con ordenación o agrupamiento.

Tipos de índices:

- **unique index:** se refiere a un índice en el que todas las columnas deben tener un valor único.
- **Primary index:** igual al anterior, pero en este caso solo puede existir un índice primary en cada una de las tablas.
- **Key:**
- **Full text:** Estos índices se emplean para realizar búsquedas sobre texto (CHAR, VARCHAR y TEXT).

Estructura de índices:

- **B-TREE**: este tipo de índice se usa para comparaciones del tipo =, >, =, <=, BETWEEN y LIKE (siempre y cuando se utilice sobre constantes que no empiecen por %).
- HASH: este tipo de índice sólo se usa para comparaciones del tipo = o <=>

En el siguiente ejemplo se crea un índice sobre la tabla PERSONAS ordenando el mismo por los campos apellido en forma descendente y nombre en forma descendente.

```
CREATE UNIQUE INDEX IDX_PERSONAS
ON personas (apellido desc, nombre asc)
USING BTREE;
```

Tipos de acceso

- **TABLE FULL SCAN:** Se busca toda la tabla los registros que cumplan con la condición
- **INDEX UNIQUE SCAN:** E l acceso busca una única clave a través de un índice único. Eso significa que siempre la búsqueda devolverá un resultado
- **INDEX MERGE:** Para obtener la tabla se debe acceder a dos índices y luego hacer merge de los resultados.
- **INDEX RANGE SCAN:** Como su nombre indica, en este caso no buscamos por un único valor, sino por un rango de ellos, por lo que nos devolverá más de un registro. Esto ocurre cuando usamos los operadores de rango (, <>, <=, >=, BETWEEN).
- **INDEX FULL SCAN:** Esta es la peor de las decisiones con índices y para que ocurra deberemos de no filtrar por el índice

Usuarios

```
/**
 * Estos son los diferentes tipos de privilegios que
 * pueden otorgarse y revocarse a los usuarios
 **/
Privilege Type                  Description
ALL                             Grants all privileges, except WITH GRANT OPTION
ALTER                           Grants use of ALTER TABLE
CREATE                          Grants use of CREATE TABLE
CREATE TEMPORARY TABLES         Grants use of CREATE TEMPORARY TABLE
DELETE                          Grants use of DELETE
DROP                            Grants use of DROP TABLE
EXECUTE                         Grants use of stored procedures
FILE                            Grants use of SELECT INTO OUTFILE and LOAD DATA INFILE
GRANT OPTION                    Used to revoke WITH GRANT OPTION
INDEX                           Grants use of CREATE INDEX and DROP INDEX
INSERT                          Grants use of INSERT
LOCK TABLES                     Grants use of LOCK TABLES on tables on which the user already has the SELECT privilege
PROCESS                         Grants use of SHOW FULL PROCESSLIST
RELOAD                          Grants use of FLUSH
REPLICATION CLIENT              Grants ability to ask where the slaves/masters are
REPLICATION SLAVE               Grants ability of the replication slaves to read information from master
SELECT                          Grants use of SELECT
SHOW DATABASES                  Grants use of SHOW DATABASES
SHUTDOWN                        Grants use of MYSQLADMIN SHUTDOWN
SUPER                           Grants the user one connection, one time,
                                even if the server is at maximum connections limit,
                                and grants use of CHANGE MASTER, KILL THREAD, MYSQLADMIN DEBUG,
                                PURGE MASTER LOGS, and SET GLOBAL
UPDATE                          Grants use of UPDATE
USAGE                           Grants access to log in to the MySQL Server but bestows no privileges
WITH GRANT OPTION               Grants ability for users to grant any privilege they possess to another user
```

```
-- Crean una base de datos nueva
create database clase_usuarios;

-- Crean un usuario nuevo
-- (en el link de abajo tienen información (usando root como ejemplo)
-- de por qué es necesario crear el mismo usuario para diferentes
-- "versiones" de localhost).
-- https://dba.stackexchange.com/questions/59412/multiple-root-users-in-mysql
create user 'adminclaseusuarios'@'localhost' identified by 'adminclaseusuarios';

-- Le dan todos los permisos posibles, se transforma
-- en su ADMIN para esta base de datos.
GRANT ALL PRIVILEGES ON clase_usuarios.* TO 'ssssss'@'localhost' WITH GRANT OPTION;

-- Cierren la conexión actual e ingresen a MySQL con el nuevo usuario creado
mysql -u adminclaseusuarios -p

-- Verifiquen que solo tienen acceso a dos bases de datos.
SHOW databases;

-- Verifiquen los permisos que tiene su usuario actual
SHOW GRANTS FOR CURRENT_USER();
```

Sql vs NoSql

Ventajas SQL:

- **Madurez**: existe una gran variedad y cantidad de información para poder realizar cualquier tipo de desarrollo o extracción de información,
- **Atomicidad**: En las operaciones e información, esto quiere decir que cualquier operación realizada en la base de datos, garantiza que si a la mitad de cualquier operación de base de datos, surgió algún tipo de problema, la información no se completa, o se realiza al 100% o no se realiza nada.
- **Estándares bien definidos**: Por ejemplo, la creación de tablas, el insertar, eliminar y actualizar información, consultas, se escriben bajo la misma sintaxis, basados en el estándar de SQL

Desventajas SQL:

- **Cambios en la estructura:** En muchas ocasiones, los negocios necesitan realizar cambios, tanto en sus operaciones como en los sistemas de informática, detener el sistema por un tiempo moderado
- **Elección del más adecuado:**
- **Complejidad en la instalación:** Algunos RDBMS dependen del sistema operativo donde se vayan a instalar, no garantizan el buen funcionamiento si no cumplen con los requerimientos mínimos de instalación.

Qué son las base de datos NoSQL?

Son un grupo de aplicaciones que rompen con el esquema tradicional de las bases de datos RDBMS (relational database management system), que nos garantizan mayor rendimiento y escalabilidad horizontal. Básicamente difieren en que no usan un esquema fijo para el almacenamiento de los datos.

Ventajas NoSQL:

Escalabilidad: a fin de cubrir los requisitos de demanda de la aplicación, en las bases de datos NoSQL no resulta una tarea complicada agregar más nodos o instancias.

Flexibilidad: está relacionada con el hecho de que las bases de datos NoSQL no están atadas a un esquema predefinido, con lo cual ofrecen mucha flexibilidad.

Alta disponibilidad: gracias a la escalabilidad, se garantiza la continuidad del servicio en caso de que se origine un error de software o hardware.

Rendimiento: por su naturaleza distribuida, hacen que el tiempo de respuesta de las consultas sea muy superior con respecto a las RBDMS tradicionales

Desventajas NoSQL:

Atomicidad: No todas las bases de datos contienen la característica de la atomicidad en la información, esto quiere decir, que la información en ocasiones no es consistente.

Documentación del Software: Dado que NoSQL, es relativamente nuevo, las operaciones pueden ser limitadas y se requiera de conocimientos avanzados con el uso de la herramienta

Estándares en el lenguaje: No se tiene un estándar definido entre los diferentes motores que ofrecen este servicio.

Herramientas GUI(Graphical User Interface): Las herramientas que ofrecen para la administración de estas herramientas, suelen tener acceso por consola.