PLSQL

Programación a nivel de BD

Procedimientos Almacenados

Un procedimiento almacenado es un conjunto de instrucciones, que se almacenan bajo un **nombre**

Ventajas:

- Concentran lógica de negocio.
- Evitan acceso directo a las tablas.
- Reducen el tráfico de red.

Desventajas:

- Se aplican lógicas de negocios, pero no muy complejas.
- Difíciles de depurar.
- Ejecución de un procedimiento almacenado: Se invoca por su nombre
- Se pueden referenciar tablas, vistas, funciones y procedimientos almacenados.
- Se pueden utilizar instrucciones DML no DDL.

Procedimientos Almacenados

```
--  Crear Base de datos de prueba denominada peliculas
CREATE DATABASE peliculas;

-- Crea tabla
USE peliculas;

DROP TABLE IF EXISTS programas;

CREATE TABLE programas (
  id INTEGER NOT NULL AUTO_INCREMENT,
  titulo VARCHAR(100) NOT NULL,
  estudio VARCHAR(64) NOT NULL,
  anio integer NOT NULL DEFAULT 0,
  PRIMARY KEY(id)
) ENGINE = InnoDB;
```

Procedimientos Almacenados

```
-- Insertamos datos de prueba

insert into programas(titulo, estudio, anio) values('ET','Warner',1983);
insert into programas(titulo, estudio, anio) values('Indiana Jones','Hollywood',1987);
insert into programas(titulo, estudio) values('El Barco','Amblin');
```

Procedimientos Almacenados

```
-- Primer procedimiento Almacenado

delimiter //

CREATE PROCEDURE pa_programas_lista()
BEGIN
 SELECT *
 FROM programas;
END
//

delimiter ;

-- Ejecución
call pa_programas_lista();
```

Procedimientos Almacenados

Ejercicios
==========
Crear un procedimiento almacenado que cuente la cantidad de programas.

Crear un procedimiento almacenado que muestre el título más antiguo

Procedimientos Almacenados

Ejemplo:

-- Delimitadores

-- Variables

-- Begin....end

```
DELIMITER // -- Cambia Delimitadores

CREATE PROCEDURE pa_programas_cantidad2()
BEGIN
 -- Declara variable local al procedimiento
 DECLARE vprogramas INT;

 SELECT COUNT(*)
 FROM programas
 INTO vprogramas;   -- Almacena resultado consulta en variable

 SELECT vprogramas; -- Muestra variable

END

//
DELIMITER ; -- Recompone Delimitadores
```

Procedimientos Almacenados

Ejercicios
==========
Agregar una columna llamada precio a la tabla programas

Crear un procedimiento almacenado que complete los valores de precio con 100

Procedimientos Almacenados

```
-- Parámetros
delimiter //
CREATE PROCEDURE pa_programas_buscar(_id integer)
BEGIN

 SELECT *
 FROM programas
 WHERE id = _id;

END
//
delimiter ;
```

Procedimientos Almacenados

Ejercicios
========

Crear un procedimiento almacenado que permita ingresar como parámetro una o más letras y busque todos los programas que empiecen con esas letras

Crear un procedimiento almacenado que actualice los precios en un % pasado como parámetro

Procedimientos Almacenados

```
-- Parámetros de E/S

-- Agregamos columna tipo
alter table programas add column tipo varchar(20);

update programas set tipo ='serie' where id=1;
update programas set tipo ='documental' where id=2;
update programas set tipo ='serie' where id=3;
```

```
-- Calcular cantidad de programas.
DELIMITER //

CREATE PROCEDURE pa_programas_cantidad_tipo
(IN _tipo VARCHAR(20),  OUT _programas INT)

BEGIN

 SELECT COUNT(*)
 INTO _programas
 FROM programas
 WHERE tipo =_tipo;

END

//
DELIMITER ;
```

Procedimientos Almacenados

Ejercicios
==========

Crear un procedimiento almacenado que cuente la cantidad de programas por tipo y con un precio mayor que un valor dado.

Funciones Almacenadas

Las funciones son bloques de código que permiten agrupar y organizar sentencias SQL que se ejecutan al invocar la función.

Tienen una cabecera, una sección de declaración de variables y el bloque "begin...end" que encierra las acciones.

Una función, además contiene la cláusula "return".

Una función acepta parámetros, se invoca con su nombre y retorna un
valor.

Puede ser invocada en una consulta de la siguiente forma:

select nompro, origen, precio,f_incremento(precio,10)
from productos;

```
DELIMITER //

CREATE FUNCTION fa_programas_cantidad()
RETURNS INT -- Especificamos parámetro de salida
BEGIN

DECLARE vcantidad INT;

SELECT COUNT(*)
INTO vcantidad
FROM programas;

RETURN vcantidad;  -- Retornar el valor obtenido

END

//
DELIMITER ;
```

Funciones Almacenadas

Ejercicios
==========

Crear una función almacenada que cuente la cantidad de programas por tipo.

Crear una función almacenada que devuelva el precio de un programa determinado

Funciones Almacenadas

Ejercicios
==========

Crear una función almacenada que cuente la cantidad de programas por tipo.

Crear una función almacenada que devuelva el precio de un programa determinado

Triggers (Disparadores)

Los disparadores o triggers son bloques de código que desencadenan una actividad.

Se disparan ante las operaciones Insert, Update y Delete sobre una tabla.

Los triggers se asocian a una tabla.

Los triggers se utilizan para establecer reglas de gestión en la base.

Complementan otras herramientas.

Su funcionamiento se apoya en dos tablas auxiliares, NEW y OLD.

Ejemplo:

Creamos tabla de detalle de programas

```
CREATE TABLE prog_detalle (
  id_det INTEGER NOT NULL AUTO_INCREMENT,
  id VARCHAR(100) NOT NULL,
  titulo varchar(255),
  entradas  integer,
  recaudacion double(5,2),
  PRIMARY KEY(id_det)
) ENGINE = InnoDB;
```

Triggers (Disparadores)

```
DELIMITER //

CREATE TRIGGER tr_programa_insertar -- Nombre
AFTER INSERT ON programas                -- Tabla
FOR EACH ROW
BEGIN
        insert into prog_detalle(id, titulo,entradas, recaudacion) values (NEW.id, NEW.titulo,'Sin datos', 0);
END
//

DELIMITER ;

insert into programas(titulo, estudio, anio) values('Ghost','Amblin',1985);
```

Triggers (Disparadores)

```
DELIMITER //

CREATE TRIGGER tr_programa_actualizar       -- Nombre
AFTER UPDATE ON programas                   -- Tabla
FOR EACH ROW
BEGIN
        update prog_detalle set titulo=NEW.titulo where id = NEW.id;
END

//
DELIMITER ;

update programas set titulo = 'Titanic' where id = 7;
```

Triggers (Disparadores)

```
DELIMITER //

CREATE TRIGGER tr_programa_borrar    -- Nombre
AFTER DELETE ON programas            -- Tabla
FOR EACH ROW
BEGIN
        delete from prog_detalle where id = OLD.id;
END

//
DELIMITER ;

delete from programas set titulo = 'Titanic' where id = 7;
```

Control de Flujo

```
delimiter //
create procedure p1(in p1 int)        /* Parámetro de entrada */
        begin
                declare miVar int;        /* se declara variable local */
                set miVar = p1 +1 ;       /* se establece la variable */
                if miVar = 12 then
                        select 'verdadero' ;
                else
                        select 'falso' ;
                end if;
        end;
//
delimiter ;
```

Control de Flujo

```
delimiter //
create procedure p2 (in p1 int)
        begin
                declare var int ;
                set var = p1;
                case var
                        when 2 then select 'opción 2';
                        when 3 then  select 'opción 3';
                        else  select 'opción N';
                end case;
        end;
//

delimiter ;
```

Control de Flujo

Ejercicio

Crear un procedimiento almacenado que permita determinar si dos cadena de caracteres son iguales.

Control de Flujo

```
delimiter //
create procedure p3()
        begin
                declare v int;
                set v = 0;
                while v < 5 do
                        select v; -- insert into lista values (v);
                        set v = v +1 ;
                end while;
        end;
//
delimiter ;
```

Control de Flujo

```
delimiter //
create procedure p4()
	begin
		declare v int;
		set v = 20;
		repeat
			select v; --insert into lista values(v);
			set v = v + 1;
			until v >= 1
		end repeat;
	end;
//
delimiter ;
```

Control de Flujo

```
delimiter //
create procedure p5()
        begin
                declare v int;
                set v = 0;
                loop_label : loop
                        insert into lista values (v);
                        set v = v + 1;
                        if v >= 5 then
                                leave loop_label;
                        end if;
                end loop;
        end;
//
delimiter ;
```

Transacciones

Una transacción es un grupo secuencial de sentencias SQL.
Por ejemplo operaciones de actualización.

Una transacción sólo se confirma si cada operación individual dentro del grupo de sentencias tiene éxito.

Si al menos una de las operaciones falla, la transacción NO se completa.

ACID: Propiedades de las transacciones

- **Atomicidad:** asegura que todas las operaciones dentro de la unidad de trabajo se completen con éxito; de lo contrario, la transacción se cancela en el punto de falla y las operaciones anteriores se devuelven a su estado anterior.
- **Coherencia:** garantiza que la base de datos cambie correctamente los estados en una transacción confirmada con éxito.
- **Aislamiento:** permite que las transacciones operen independientemente y transparentes entre sí.
- **Durabilidad:** asegura que el resultado o efecto de una transacción confirmada persista en caso de una falla del sistema.

Transacciones

Inicio de transacción:

- START TRANSACTION // Inicio
- BEGIN WORK

Fin transacción:

- COMMIT // Confirma
- ROLLBACK . // Vuelve atrás

Las sentencias SQL entre las instrucciones de inicio y finalización forman parte de la transacción.

Ejemplo implementación de transacciones

```
START TRANSACTION;

    UPDATE accTable SET
    ledgerAmt=ledgerAmt-@transAmt WHERE
    customerId=1;

    UPDATE accTable SET
    ledgerAmt=ledgerAmt+@transAmt WHERE
    customerId=2;

COMMIT;
```

Transacciones

Procedimientos generales involucrados en la transacción.

- Comenzar la transacción con el comando SQL BEGIN WORK o START TRANSACTION .
- Ejecutar todas las sentencias SQL.
- Comprobar si todas las sentencias se ejecutan correctamente.
- Si la ejecución es válida, confirma mediante COMMIT; de lo contrario, revertir con el comando ROLLBACK.

Transacciones

La variable AUTOCOMMIT se establece como *verdadera* por defecto.

– Cambio a FALSE

```
SET AUTOCOMMIT=false;
SET AUTOCOMMIT=0;
```

--->Cambio a TRUE

```
SET AUTOCOMMIT=true;
SET AUTOCOMMIT=1;
```

Estado de AUTOCOMMIT

```
SELECT @@autocommit;
```

Transacciones con Control de Error

```
DELIMITER //
create procedure pro1(_clave integer, _dato integer)
BEGIN
 -- Declare acción por error
 DECLARE EXIT HANDLER FOR SQLEXCEPTION
 BEGIN
        rollback;
        SELECT 'Error en transaction' AS message;
 END;
        START TRANSACTION;
        update a set z = _dato where c = 1;
        insert into a(c,z) values (_clave,_dato);
        commit;
 end //
DELIMITER ;
```

Cursores

```
DROP PROCEDURE IF EXISTS cursores;

DELIMITER //

CREATE PROCEDURE cursores ()
BEGIN
-- Declaramos variables que almacenan los datos del select especificado
 DECLARE v_id integer;
 DECLARE v_titulo varchar(100);
 DECLARE v_estudio varchar(64);
 DECLARE v_anio integer;
 DECLARE v_precio double(5,2);
 DECLARE v_tipo varchar(20);

-- Variable para controlar el fin del ciclo de recorrido del conjunto select
 DECLARE fin INTEGER DEFAULT 0;

-- Declaramos cursor a utilizar
 DECLARE cursor1 CURSOR FOR
  SELECT id, titulo, estudio, anio, precio, tipo
  FROM programas;
```

Cursores

```
-- Declaramos Condición de salida
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin=1;

 OPEN cursor1;
 ciclo: LOOP
        FETCH cursor1 INTO v_id, v_titulo, v_estudio, v_anio, v_precio, v_tipo;
        IF fin = 1 THEN
          LEAVE ciclo;
        END IF;

        SELECT v_id, v_titulo, v_estudio, v_anio, v_precio, v_tipo;

 END LOOP ciclo;

 CLOSE cursor1;
END //
DELIMITER ;
```