MANEJO DE PROGRAMACION A NIVEL DE BASE DE DATOS

JONATHAN LORENZO ALARCON GUTIERREZ

¿Que es lenguaje procedural en MySQL?

Los procedimientos almacenados MySQL son conjuntos de comandos SQL almacenados en el servidor, para que nuestros clientes no tengan que ejecutar los comandos individuales. Esto nos permite trabajar con grandes cantidades de datos delimitando la complejidad a nivel de servidor y ofreciendo a los usuarios directamente lo que están buscando:

- Pueden recibir y devolver parámetros.
- Pueden manejar tablas, ejecutando operaciones e iteraciones de lectura/escritura.
- Pueden devolver una tabla como resultado.
- Se almacenan en la base de datos en la cual se crean.
- No dependen de ninguna tabla en particular.
- Pueden aceptar recursividad.

¿Que es una function en MySQL?

Las funciones son piezas de código que reciben datos de entrada, realizan operaciones con ellos y luego devuelven un resultado. Podemos utilizarlas en nuestras consultas para que el motor de base de datos resuelva eficazmente los cálculos de algunos datos, que de otra manera, resolviéndolos en nuestra capa de negocio nos podría traer problemas de performance en la aplicación. Existen dos tipos de funciones, las integradas y las almacenadas:

Funciones Integradas son las funciones que ya vienen implementadas en el motor de MySQL y nos permiten manipular los datos al momento de hacer una consulta.

- Funciones de cadenas
- Funciones numericas
- Funciones de fechas

jFunciones Almacenadas las funciones almacenadas son aquellas que podemos definir y reutilizar en sentencias SQL como cualquier otra función definida por el motor de MySQL, donde:

CREATE FUNCTION nombre_funcion(parametro1 TIPO_DATO, parametroN TIPO_DATO): define el nombre de la función y los parámetros que recibirá con sus tipos de datos correspondientes.

RETURNS: indica el tipo de dato que tendrá el valor de salida de la función

BEGIN: determina el inicio de las sentencias que podemos realizar en la función

END: determina el fin de las sentencias

¿Cual es la diferencia entre funciones y procedimientos almacenados?

Un procedimiento almacenado es un conjunto de comandos SQL que pueden ser almacenados en el servidor. Al realizarse, los clientes no necesitan realizar los comandos individuales en su lugar pueden reutilizar los ya almacenados.
Por otro lado tenemos las funciones que al igual que los procedimientos almacenados, están formadas por un conjunto de comandos SQL y están almacenados en la base de datos.
Ambas parecen ser similares, aunque una función parece ser más limitada. ¿Cuándo y porqué usar un procedimiento y cuando una función? Las diferencias más generales entre procedimientos y funciones es que cada una es llamada de modo diferente y para propósitos distintos:
1. Un procedimiento no nos regresa un valor. En su lugar, es llamado con una declaración CALL para realizar una operación como modificar una tabla o procesar la recuperación de registros.
2. Una función es llamada dentro de una expresión y nos regresa un valor único directamente al que lo llama para ser utilizado en la expresión.
3. No se puede invocar una función con una instrucción CALL, ni puedes invocar un procedimiento en una expresión.

¿Como se ejecuta una funciones y un procedimientos almacenados?

PROCEDIMIENTO ALMACENADO

Para crear un procedimiento, MySQL nos ofrece la directiva CREATE PROCEDURE. Al crearlo éste es ligado o relacionado con la base de datos que se está usando, tal como cuando creamos una tabla, por ejemplo.

Para llamar a un procedimiento lo hacemos mediante la instrucción CALL. Desde un procedimiento podemos invocar a su vez a otros procedimientos o funciones.

FUNCION

Para crear una función, MySQL nos ofrece la directiva CREATE FUNCTION. Su sintaxis es:
CREATE FUNCTION nombre (parámetro)
RETURNS tipo
[características] definición

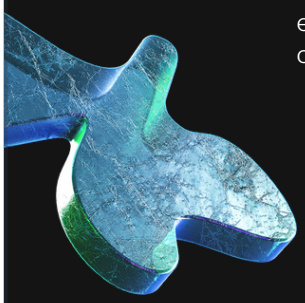
Puede haber más de un parámetro (se separan con comas) o puede no haber ninguno (en este caso deben seguir presentes los paréntesis, aunque no haya nada dentro). Los parámetros tienen la siguiente estructura:nombre tipo. Para llamar a una función lo hacemos simplemente invocando su nombre, como se hace en muchos lenguajes de programación.

DEFINICION DE UN TRIGGER

Un trigger o disparador es una regla que se asocia a una tabla. Mediante esta regla, se ejecutan una serie de instrucciones cuando se producen ciertos eventos sobre una tabla. Los eventos son: INSERT, UPDATE o DELETE. Para poder realizar triggers es necesario que tengas permisos para ejecutar esas consultas.

¿En un trigger que papel juega las variables OLD y NEW?
El alias OLD hace referencia a un registro ya exiEl alias NEW hace referencia a un registro nuevo después de un insert o update.
stente antes de un update o delete.

EN UN TRIGGER QUE PAPEL JUEGA LOS CONCEPTOS O CLAUSULAS BEFORE AND AFTER



Lo interesante aquí es que podemos programar los triggers de tal manera que se ejecuten antes o después, de dichas sentencias; Dando como resultado seis combinaciones de eventos.

- BEFORE INSERT Acciones a realizar antes de insertar uno más o registros en una tabla.
- AFTER INSERT Acciones a realizar después de insertar uno más o registros en una tabla.
- BEFORE UPDATE Acciones a realizar antes de actualizar uno más o registros en una tabla.
- AFTER UPDATE Acciones a realizar después de actualizar uno más o registros en una tabla.
- BEFORE DELETE Acciones a realizar antes de eliminar uno más o registros en una tabla.
- AFTER DELETE Acciones a realizar después de eliminar uno más o registros en una tabla.

¿A que se refiere cuando se habla de eventos en TRIGGERS?

Se refiere a los eventos que hacen que se ejecute un trigger son las operaciones de inserción (INSERT), borrado (DELETE) o actualización (UPDATE), ya que modifican los datos de una tabla.
La utilidad principal de un trigger es mejorar la gestión de la base de datos, ya que no requieren que un usuario los ejecute.

RESOLUCION DE LA PARTE PRACTICA

Se procede a la creacion de las tablas y el diagama de las mismas

```
insert into proyecto(nombre_Proy, tipo_Proy) values
('Proyecto1','Tipo1'),
('Proyecto2','Tipo2');

create table provincia
(
    id_prov int primary key auto_increment not null,
    nombre varchar(30),
    id_dep int not null,
    foreign key (id_dep) references departamento(id_dep)
);

insert into provincia(nombre, id_dep) values
('Provincia1', 1),
('Provincia2', 2);

create table persona
(
    id_per int primary key auto_increment not null,
    nombre varchar(20) not null,
    apellido varchar(50) not null,
    fecha_nac date not null,
    edad varchar(11) not null,
```

```
create database EvaluacionMit04;

use EvaluacionMit04;

-- Pregunta 9

create table departamento
(
        id_dep int primary key auto_increment not null,
        nombre varchar(50) not null
);

insert into departamento(nombre) values
('La Paz'),
('Cochabamba');

create table proyecto
(
    id_proyecto int primary key auto_increment not null,
    nombre_Proy varchar (100) not null,
    tipo_Proy varchar(30) not null
);
```

```
create or replace table detalle_proyecto
(
    id_dp int primary key auto_increment not null,
    id_per int not null,
    id_proy int not null,
    foreign key (id_per) references persona(id_per),
    foreign key (id_proy) references proyecto(id_proyecto)
);

insert into detalle_proyecto(id_per, id_proy) values
(1,1),
(2,2);
```

```
create table persona
(
    id_per int primary key auto_increment not null,
    nombre varchar(20) not null,
    apellido varchar(50) not null,
    fecha_nac date not null,
    edad varchar(11) not null,
    email varchar(50) not null,
    id_dep int not null,
    id_prov int not null,
    sexo char(1) not null,
    foreign key (id_dep) references departamento(id_dep),
    foreign key (id_prov) references provincia(id_prov)
);

insert into persona(nombre, apellido, fecha_nac, edad, email, id_dep, id_prov, sexo) values
('Persona1','Apellido1','2001-11-13','21','Persona1@gmail.com',1,1,'M'),
('Persona2','Apellido2','2003-08-17','19','Persona2@gmail.com',2,2,'F');
```

Crear una función que sume los valores de la serie Fibonacci.

- El objetivo es sumar todos los números de la serie fibonacci desde una cadena.

```
create function fibonacci(numero int)

RETURNS TEXT
BEGIN
    DECLARE A INTEGER DEFAULT 0;
    DECLARE B INTEGER DEFAULT 1;
    DECLARE aux INTEGER DEFAULT 0;
    DECLARE count INTEGER DEFAULT 0;
    DECLARE chain TEXT DEFAULT '';
    SET chain =CONCAT(A,',',B);

IF numero=1 THEN SET chain='0';
ELSEIF numero=2 THEN SET chain='0,1';
ELSEIF numero<=0 THEN SET chain='EL NUMERO DEBE SER MAYOR A CERO';
ELSE
        REPEAT

        SET AUX=A+B;
        SET chain = CONCAT(chain,',',AUX);
        SET A=B;
        SET B=AUX;
        SET count=count+1;
        UNTIL count = numero-2 END REPEAT;

END IF;
    RETURN chain;
END;

CREATE OR REPLACE FUNCTION CONTAR_FIBONANCI(SERIE TEXT)
RETURNS INTEGER
BEGIN
    DECLARE SUMA INTEGER DEFAULT 0;
    DECLARE CONT INTEGER DEFAULT 1;
    DECLARE FINAL
    INTEGER DEFAULT CHAR_LENGTH(SERIE);

  REPEAT
    SET SUMA =SUMA + SUBSTRING(SERIE,CONT,1);
    SET CONT=CONT+2;
  until  CONT> FINAL END REPEAT;

    RETURN SUMA;
end;

SELECT fibonacci( numero: 10);
SELECT CONTAR_FIBONANCI( SERIE: fibonacci( numero: 6));
```

	`fibonacci(10)`
1	0,1,1,2,3,5,8,13,21,34

	`CONTAR_FIBONANCI(fibonacci(7))`
1	20

Manejo de vistas.

Crear una consulta SQL para lo siguiente.

La consulta de la vista debe reflejar como campos:

- nombres y apellidos concatenados
- la edad
- fecha de nacimiento.
- Nombre del proyecto

Obtener todas las personas del sexo femenino que hayan nacido en el departamento de El Alto en donde la fecha de nacimiento sea:

1. fecha_nac = '2000-10-10'

```sql
INSERT INTO departamento(nombre)
VALUES('EL ALTO'),
      ('SAN FIERRO');

insert provincia(nombre, id_dep)
Values ('Provinvcia3',3),
 ('Provincia4',4);

INSERT INTO persona(nombre, apellido, fecha_nac, edad, email, id_dep, id_prov, sexo)
VALUES ('CARMEN','CALLE VARGAS','2002-10-10','21','CARMEN@GMAIL.COM',3,3,'F'),
       ('GABRIELA','CASAS MENDOZA','2000-10-10','23','GABRIELA@GMAIL.COM',4,4,'F');

insert into proyecto(nombre_Proy, tipo_Proy) values
 ('Proyecto3','Tipo3'),
('Proyecto4','Tipo4');

INSERT INTO DETALLE_PROYECTO(id_per, id_proy)
VALUES (3,3),
       (4,4);

select *
from persona;
```

	id_per	nombre	apellido	fecha_nac	edad	email	id_dep	id_prov	sexo
1	1	Persona1	Apellido1	2001-11-13	21	Persona1@gmail.com	1	1	M
2	2	Persona2	Apellido2	2003-08-17	19	Persona2@gmail.com	2	2	F
3	3	CARMEN	CALLE VARGAS	2002-10-10	21	CARMEN@GMAIL.COM	3	3	F
4	4	GABRIELA	CASAS MENDOZA	2000-10-10	23	GABRIELA@GMAIL.COM	4	4	F

```sql
CREATE OR REPLACE VIEW SEARCH AS
select *
from (SELECT CONCAT(per.nombre, ' ', per.apellido) AS NAME_AND_LASTNAME,
             per.edad AS EDAD,
             per.fecha_nac AS FECHA_DE_NACIMIENTO,
             pro.nombre_Proy AS PROJECT_NAME
      FROM persona as per
               INNER JOIN departamento as dep ON per.id_dep = dep.id_dep
               INNER JOIN detalle_proyecto as dp ON per.id_per = dp.id_per
               INNER JOIN proyecto as pro ON dp.id_proy = pro.id_proyecto
      WHERE per.sexo = 'F'
        AND dep.nombre = 'EL ALTO'
        AND per.fecha_nac = '2002-10-10') PDDPP;
```

Output | evaluacionhito4.search

	NAME_AND_LASTNAME	EDAD	FECHA_DE_NACIMIENTO	PROJECT_NAME
1	CARMEN CALLE VARGAS	21	2002-10-10	Proyecto3

Manejo de TRIGGERS I.
Crear TRIGGERS Before or After para INSERT y UPDATE aplicado a la tabla PROYECTO
Debera de crear 2 triggers minimamente.
Agregar un nuevo campo a la tabla PROYECTO.
El campo debe llamarse ESTADO

Output | evaluacionhito4.proyecto

8 rows | Tx: Auto | DDL

	id_proyecto	nombre_Proy	tipo_Proy	ESTADO
1	1	Proyecto1	Tipo1	<null>
2	2	Proyecto2	Tipo2	<null>
3	3	Proyecto3	Tipo3	<null>
4	4	Proyecto4	Tipo4	<null>
5	5	EDUCACION PERSONAS ESPECIALES	EDUCACION	<null>
6	6	PLANTACION DE ARBOLES	FORESTACION	<null>
7	7	LOS AZTECAS	CULTURA	<null>
8	8	ARBOLESS	EDUCACION	ACTIVO

```
ALTER TABLE PROYECTO ADD (ESTADO VARCHAR(30));

INSERT INTO PROYECTO(NOMBRE_PROY, TIPO_PROY)
VALUES ('EDUCACION PERSONAS ESPECIALES','EDUCACION'),
       ('PLANTACION DE ARBOLES','FORESTACION'),
       ('LOS AZTECAS','CULTURA');

CREATE OR REPLACE TRIGGER UPDATE_TIP_PROY
BEFORE UPDATE ON PROYECTO
FOR EACH ROW
    BEGIN
        IF  NEW.TIPO_PROY='EDUCACION'OR  NEW.TIPO_PROY ='FORESTACION' OR  NEW.TIPO_PROY= 'CULTURA'
            THEN SET NEW.ESTADO='ACTIVO';
        ELSE
            SET NEW.ESTADO='INACTIVO';
        END IF;
    END;

CREATE OR REPLACE TRIGGER ADD_TIP_PROY
BEFORE  INSERT ON proyecto
FOR EACH ROW
    BEGIN
        IF  new.tipo_Proy='EDUCACION'OR  new.tipo_Proy ='FORESTACION' OR  new.tipo_Proy= 'CULTURA'
            THEN SET new.ESTADO='ACTIVO';
        ELSE
            SET new.ESTADO='INACTIVO';
        END IF;
    end;

INSERT INTO proyecto(nombre_Proy, tipo_Proy)
VALUES ('ARBOLESS','EDUCACION');

SELECT * FROM proyecto;
```

Manejo de Triggers II.
El trigger debe de llamarse calculaEdad.
El evento debe de ejecutarse en un BEFORE INSERT.
Cada vez que se inserta un registro en la tabla PERSONA, el trigger debe de calcular la edad en función a la fecha de nacimiento.
Adjuntar el código SQL generado y una imagen de su correcto funcionamiento.

```
CREATE OR REPLACE TRIGGER calculaEdad
 BEFORE INSERT ON persona
 FOR EACH ROW
    BEGIN
        SET NEW.EDAD= TIMESTAMPDIFF(YEAR,NEW.FECHA_NAC,CURDATE());
    end;

INSERT INTO  persona(nombre, apellido, fecha_nac, email, id_dep, id_prov, sexo)
VALUES  ('MARIA','GALARSA ORTEGA','1992-12-15','MARIA@GMAIL.COM',3,3,'F');
INSERT INTO  persona(nombre, apellido, fecha_nac, email, id_dep, id_prov, sexo)
VALUES ('JONATHAN','ALARCON GUTIERREZ', '2003-10-24', 'johnlag10@email', 4, 4, 'M');

select *
from persona;
```

Output | evaluacionhito4.persona

	id_per	nombre	apellido	fecha_nac	edad	email	id_dep	id_prov	sex
1	1	Persona1	Apellido1	2001-11-13	21	Persona1@gmail.com	1	1	M
2	2	Persona2	Apellido2	2003-08-17	19	Persona2@gmail.com	2	2	F
3	3	CARMEN	CALLE VARGAS	2002-10-10	21	CARMEN@GMAIL.COM	3	3	F
4	4	GABRIELA	CASAS MENDOZA	2000-10-10	23	GABRIELA@GMAIL.COM	4	4	F
5	5	MARIA	GALARSA ORTEGA	1992-12-15	29	MARIA@GMAIL.COM	3	3	F
6	6	JONATHAN	ALARCON GUTIERREZ	2003-10-24	19	johnlag10@email	4	4	M

Manejo de TRIGGERS III.
Crear otra tabla con los mismos campos de la tabla persona(Excepto el primary key id_per).
No es necesario que tenga PRIMARY KEY.
Cada vez que se haga un INSERT a la tabla persona estos mismos valores deben insertarse a la tabla copia.
Para resolver esto deberá de crear un trigger before insert para la tabla PERSONA.
Adjuntar el código SQL generado y una imagen de su correcto funcionamiento.

```sql
CREATE TABLE PERSONA_2
(
    NOMBRE VARCHAR(20),
    APELLIDOS VARCHAR(50),
    FECHA_NAC DATE,
    EDAD INT,
    EMAIL VARCHAR(50),
    ID_DEP INT NOT NULL ,
    ID_PROV INT NOT NULL,
    GENERO VARCHAR(1),
    FOREIGN KEY (ID_PROV) REFERENCES  provincia(id_prov),
    FOREIGN KEY (ID_DEP) REFERENCES departamento(id_dep)
);
```

```sql
CREATE OR REPLACE TRIGGER PERSONA_2
BEFORE INSERT ON persona
FOR EACH ROW
    BEGIN
        INSERT INTO PERSONA_2(NOMBRE, APELLIDOS, FECHA_NAC, EDAD, EMAIL, ID_DEP, ID_PROV, GENERO)
        VALUES(NEW.NOMBRE,NEW.APELLIDO,NEW.FECHA_NAC,NEW.EDAD,NEW.EMAIL,NEW.ID_DEP,NEW.ID_PROV,NEW.SEXO);
    end;

INSERT INTO persona(nombre, apellido, fecha_nac, edad, email, id_dep, id_prov, sexo)
VALUES ('PEDRO','ALIAGA ORTEGA','2000-12-15',22,'ALEJANDRA@GMAIL.COM',3,3,'M'),
       ('ISSEI', 'HYODOU', '2001-10-14', 21, 'redragon00@gmail.com', 4, 4, 'M');

SELECT * FROM PERSONA_2;
```

	NOMBRE	APELLIDOS	FECHA_NAC	EDAD	EMAIL	ID_DEP	ID_PROV	GENERO
1	PEDRO	ALIAGA ORTEGA	2000-12-15	21	ALEJANDRA@GMAIL.COM	3	3	M
2	ISSEI	HYODOU	2001-10-14	21	redragon00@gmail.com	4	4	M

Crear una consulta SQL que haga uso de todas las tablas.

- La consulta generada convertirlo a VISTA

```sql
CREATE OR REPLACE VIEW BANKAI AS
SELECT
    CONCAT(per.nombre,per.apellido) AS NOMBRE_Y_APELLIDOS,
    per.edad AS EDAD,
    dep.nombre AS DEPARTAMENTO,
    pvc.nombre AS PROVINCIA,
    CONCAT(pro.nombre_Proy,': ',tipo_Proy) AS PROYECTO
FROM persona as per
    INNER JOIN departamento as dep ON per.id_dep = dep.id_dep
    INNER JOIN detalle_proyecto as dp ON per.id_per = dp.id_per
    INNER JOIN proyecto as pro ON dp.id_proy = pro.id_proyecto
    INNER JOIN provincia as pvc ON per.id_prov = pvc.id_prov;

SELECT * FROM (BANKAI);
```

Output | evaluacionhito4.bankai

4 rows | Tx: Auto | DDL | CSV

	NOMBRE_Y_APELLIDOS	EDAD	DEPARTAMENTO	PROVINCIA	PROYECTO
1	Persona1Apellido1	21	La Paz	Provincia1	Proyecto1: Tipo1
2	Persona2Apellido2	19	Cochabamba	Provincia2	Proyecto2: Tipo2
3	CARMENCALLE VARGAS	21	EL ALTO	Provinvcia3	Proyecto3: Tipo3
4	GABRIELACASAS MENDOZA	23	SAN FIERRO	Provincia4	Proyecto4: Tipo4

LA
PRESENTACION A
LLEGADO A SU
FIN...