Materia: ADMINISTRACIÓN DE BASES DE DATOS
Maestro: Gerardo Rodríguez Rojano
Fecha: 20 de marzo de 2019

Objetivos:

- Detener los errores en plpgsql
- Aplicación de Triggers

Fecha de Entrega: 03 de abril de 2019.

Modalidad: En equipos de 2 a 3 integrantes.
Forma de entregar: subir a la plataforma CAMPUS VIRTUAL archivo con formato de texto extensión .sql o .txt con el código solicitado en esta tarea.

Expediente	Nombre Completo
258849	**Michael Brandon Serrato Guerrero**

---

1. Considere el manejo de errores en plpgsql. Explicar en qué situaciones los siguientes errores son lanzados por el sistema postgres:

    - not_null_violation
        - **R:** Es lanzado cuando no está permitido la inserción de valores nulos en ciertas columnas de la tabla (al menos una).
    - unique_violation
        - **R:** Ocurre cuando se intenta duplicar una columna al insertar un nuevo registro, especialmente cuando se trata de una primary key.

- foreign_key_violation
    - **R:** Ocurre cuando se asigna a un registro una clave foranea que no existe en la tabla relacionada.
- restrict_violation
    - **R:** Es un error lanzado cuando se intenta eliminar un registro que tiene dependencias de otros registros externos a la tabla (por claves foraneas).
- undefined_table
    - **R:** Ocurre ya sea por error de sintaxis (cuando el nombre de la tabla no está escrito correctamente y por lo tanto la tabla no existe) o por violación de reglas de acceso a dicha tabla.
- string_data_right_truncation
    - **R:** Es un error lanzado cuando se excede la longitud máxima de una columna al intentar insertar un valor de cadena de caracteres.
- division_by_zero
    - **R:** Es lanzado cuando se intenta realizar la división de un número cualquiera entre cero (debido a que el resultado de esta operación no está determinado computacionalmente).

2. Este sistema cuenta con varias sucursales, donde cada sucursal tiene un presupuesto máximo para pago de salarios mensualmente.

Requerimiento: Crear un activador (trigger) para evitar que a una sucursal se le asignen más empleados que su presupuesto para nómina autorizado.

Ejemplo:

Sucursal: **'S01'**
Presupuesto máximo nómina: **$34'000.0**

Empleados asignados:
- E10, $8'000.0
- E01, $10'000.0
- E05, $10'000.0
- E18, $6'000.0

$34'000.0

En este escenario no se violenta la regla de negocio.

La Base de Datos debe hacer cumplir la regla del negocio de no exceder el presupuesto máximo para pago de salarios por sucursal.

Tabla sucursales

- Una sucursal tiene un máximo de presupuesto para salarios de empleados, por lo que no es posible rebasar éste tope presupuestal con el pago del salario de cada uno de sus empleados.

Tabla empleados

- Un empleado debe estar asignado de una única sucursal.

```
Create table sucursales (
        sucursalNo CHAR(3) PRIMARY KEY,
        sucursalCiudad VARCHAR(40) NOT NULL,
        sucursalColonia VARCHAR(40) NOT NULL,
        sucursalCalle_y_numero VARCHAR(40) NOT NULL,
        sucursalPresupuesto DECIMAL (8,2) DEFAULT 30000 NOT NULL
);

Create table empleados (
        empNo CHAR(3)  PRIMARY KEY,
        sucursalNo CHAR(3) NOT NULL,
        empNombre Varchar(40) NOT NULL,
        empApellidos Varchar(40) NOT NULL,
        empSalario DECIMAL(8,2) DEFAULT 4000 NOT NULL
);

ALTER TABLE empleados ADD CONSTRAINT empleados_sucursalNo_fkey FOREIGN KEY (sucursalNo)
  REFERENCES sucursales(sucursalNo) ON UPDATE CASCADE ON DELETE CASCADE;
```

-- Función Postgres que REGRESA TRIGGER:

```
CREATE OR REPLACE FUNCTION funcCheckBudget() RETURNS TRIGGER AS $$
DECLARE
        salaries DECIMAL(8, 2) := 0.0;
        budget DECIMAL(8, 2) := 0.0;
BEGIN
        SELECT (sum(empSalario) + NEW.empSalario) INTO salaries
                FROM empleados
                WHERE sucursalNo = NEW.sucursalNo AND empNo != NEW.empNo;

        SELECT s.sucursalPresupuesto INTO budget
                FROM sucursales s
                WHERE s.sucursalNo = NEW.sucursalNo;

        IF salaries > budget THEN
                RAISE NOTICE 'El salario del nuevo empleado excede el presupuesto de la sucursal %.
        Por favor verifique.', NEW.sucursalNo;
                RETURN NULL;
        END IF;

        RETURN NEW;
END
$$ LANGUAGE 'plpgsql';
```

-- Crear el TRGGER en la tabla:

```
CREATE TRIGGER triggerCheckBudget BEFORE INSERT OR UPDATE
        ON empleados
        FOR EACH ROW
        EXECUTE PROCEDURE funcCheckBudget();
```

-- TESTING

Probar tu código con el siguiente escenario.

```
-- INSERT Sucursales
INSERT INTO sucursales VALUES('S01','CELAYA','CENTRO','JUAREZ #45',34000);
INSERT INTO sucursales VALUES('S02','SANTIAGO DE QRO','CENTRO','HIDALGO #70',40000);
```

-- INSERT Empleados

```
INSERT INTO empleados VALUES('E01','S01','ERIKA','MENDOZA CHAVEZ',10000);
INSERT INTO empleados VALUES('E02','S01','LORENA','MADRIGAL CORONA',10000);
INSERT INTO empleados VALUES('E03','S01','JESUS','SALAZAR TREJO',8000);
INSERT INTO empleados VALUES('E04','S01','VILMA','SOSA SUAREZ',7500);
NOTICE:  Se excedió el Presupuesto de la sucursal S01. Verifique.
Query returned successfully: 0 rows affected, 15 msec execution time.
INSERT INTO empleados VALUES('E04','S01','VILMA','SOSA SUAREZ',6001);
NOTICE:  Se excedió el Presupuesto de la sucursal S01. Verifique.

INSERT INTO empleados VALUES('E04','S01','VILMA','SOSA SUAREZ',6000);
INSERT INTO empleados VALUES('E05','S01','AUGUSTO','ZAMORA ESCAMILLA',default);
NOTICE:  Se excedió el Presupuesto de la sucursal S01. Verifique.
Query returned successfully: 0 rows affected, 16 msec execution time.
```

-- SELECTS

SELECT * FROM sucursales;

	sucursalno character(3)	sucursalciudad character varying(40)	sucursalcolonia character varying(40)	sucursalcalle_y_numero character varying(40)	sucursalpresupuesto numeric(8,2)
1	S01	CELAYA	CENTRO	JUAREZ #45	34000.00
2	S02	SANTIAGO DE QRO	CENTRO	HIDALGO #70	40000.00

SELECT * FROM empleados;

	empno character(3)	sucursalno character(3)	empnombre character varying(40)	empapellidos character varying(40)	empsalario numeric(8,2)
1	E01	S01	ERIKA	MENDOZA CHAVEZ	10000.00
2	E02	S01	LORENA	MADRIGAL CORONA	10000.00
3	E03	S01	JESUS	SALAZAR TREJO	8000.00
4	E04	S01	VILMA	SOSA SUAREZ	6000.00

3. Gestión de gastos.

El usuario necesita registrar cada uno de los gastos realizados en la empresa por diferentes conceptos. Los conceptos aquí son: 'A', 'B' y 'C'. Inmediatamente, el sistema mantiene en tiempo real los saldos totales por conceptos de gasto, donde es posible saber cuánto se ha gastado del concepto 'A', 'B' o 'C'.

El sistema recibe las entradas a la tabla de gastos con la siguiente información:

- Concepto del gasto
- Monto del gasto en pesos mexicanos, donde dicho monto no debe ser menor que cero, es decir, el monto no puede ser negativo.

El sistema genera el número secuencial del gasto y la fecha de registro a la tabla de gastos.

Se necesita crear un activador (trigger) dentro de la base de datos que adicione el gasto capturado a la tabla de conceptos de gastos y actualice la fecha-hora de última modificación de la siguiente manera:

Tabla de Conceptos del Gasto

Concepto de Gasto	Monto ($)	Fecha-hora última modificación
A	0.0	2019-03-22 14:30:36
B	0.0	2019-03-22 14:30:42
C	0.0	2019-03-22 14:30:49

En este momento de la tabla la 'fecha-hora de última modificación' expresa el momento cuando se insertó cada concepto en la tabla. Luego indicará el momento en que se actualice el monto.

1. Ingresar a la tabla de **registro de gastos** el concepto 'A' de gasto, su monto de $150.0, en la fecha-hora del sistema.

2. El sistema adiciona el gasto al concepto 'A'.

Tabla de Conceptos del Gasto

Concepto de Gasto	Monto ($)	Fecha-hora última modificación
A	150.0	2019-03-22 14:32:26
B	0.0	2019-03-22 14:30:42
C	0.0	2019-03-22 14:30:49

En este momento de la tabla la 'fecha-hora de última modificación' expresa el momento cuando el gasto de tipo ‘A’ fue modificado. Los conceptos de gasto ‘B’ y ‘C’ están intactos.

```
CREATE TABLE gastos (
  gastoNo SERIAL PRIMARY KEY,
  gastoConcepto CHAR(1) NOT NULL,  -- A, B, C, D, ...
```

```
  gastoMonto DECIMAL(8,2) DEFAULT 0.0,
  gastoMontoFechaHra TIMESTAMP DEFAULT current_timestamp
);

CREATE TABLE conceptosDeGasto (
  conceptoClave CHAR(1) PRIMARY KEY,
  conceptoGastosTot DECIMAL(8,2) DEFAULT 0.0,
  conceptoFechaHraUltModif TIMESTAMP DEFAULT current_timestamp
);

SELECT * FROM gastos;
SELECT * FROM conceptosDeGasto;
```

-- Función Postgres que REGRESA TRIGGER

```
CREATE OR REPLACE FUNCTION funcAddExpenseConcept() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.gastoMonto < 0 THEN
                RAISE EXCEPTION 'El monto del gasto no puede ser un valor negativo (menor a cero).';
                RETURN NULL;
        END IF;

        UPDATE conceptosDeGasto
        SET conceptoGastosTot = (conceptoGastosTot + NEW.gastoMonto), conceptoFechaHraUltModif
= now()
        WHERE conceptoClave = NEW.gastoConcepto;

        RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
```

-- Crear el TRGGER en la tabla

```
CREATE TRIGGER triggerAddExpenseConcept BEFORE INSERT
        ON gastos
        FOR EACH ROW
        EXECUTE PROCEDURE funcAddExpenseConcept();
```

-- TESTING

Probar tu código con el siguiente escenario.

-- INSERT Conceptos de gasto

```
INSERT INTO conceptosDeGasto VALUES('A',default,default);
INSERT INTO conceptosDeGasto VALUES('B',default,default);
INSERT INTO conceptosDeGasto VALUES('C',default,default);
```

-- SELECTS

```
SELECT * FROM gastos;
```

	gastono integer	gastoconcepto character(1)	gastomonto numeric(8,2)	gastomontofechahra timestamp without time zone

```
SELECT * FROM conceptosDeGasto ORDER BY conceptoClave;
```

	conceptoclave character(1)	conceptogastostot numeric(8,2)	conceptofechahraultmodif timestamp without time zone
1	A	0.00	2019-04-04 00:12:53.751
2	B	0.00	2019-04-04 00:12:53.751
3	C	0.00	2019-04-04 00:12:53.751

-- INSERT gastos

```
INSERT INTO gastos VALUES(default, 'A', 150, default);
INSERT INTO gastos VALUES(default, 'B', 10, default);
INSERT INTO gastos VALUES(default, 'C', 15, default);

INSERT INTO gastos VALUES(default, 'A', 100, default);
INSERT INTO gastos VALUES(default, 'B', 20, default);
INSERT INTO gastos VALUES(default, 'C', 25, default);

INSERT INTO gastos VALUES(default, 'C', -25, default);
ERROR:  el nuevo registro para la relación «gastos» viola la restricción «check» «gastos_gastomonto_check»
DETAIL:  La fila que falla contiene (7, C, -25.00, 2019-03-22 14:33:26.021168).
********** Error **********

"A";250.00;"2019-03-22 14:30:58.921409"
"B";30.00;"2019-03-22 14:30:58.921409"
"C";40.00;"2019-03-22 14:30:58.921409"

INSERT INTO gastos VALUES(default, 'A', 45, default);
SELECT * FROM conceptosDeGasto ORDER BY conceptoClave;
```

	conceptoclave character(1)	conceptogastostot numeric(8,2)	conceptofechahraultmodif timestamp without time zone
1	A	295.00	2019-04-04 00:44:36.616
2	B	30.00	2019-04-04 00:43:06.109
3	C	40.00	2019-04-04 00:43:17.973

4. Sistema de seguimiento a los porcentajes de venta por artículo. En esta parte del sistema interesa ir contabilizando el total de artículos vendidos en las diferentes ventas, contabilizar el total vendido por artículo y realizar un cálculo en tiempo real del porcentaje vendido por artículo con respecto al total de artículos vendidos. **Se necesita crear un activador (trigger) dentro de la base de datos que realice dicha contabilidad y cálculo del porcentaje de venta por artículo.**

La siguiente tabla muestra el catálogo de cada artículo de la empresa y su porcentaje de venta con relación al total de las ventas, donde se asume que todavía no hay registro de ventas. Esta es una tabla resumida donde sólo se muestran las columnas de clave artículo y porcentaje de venta.

Catálogo de artículos

Artículo	Porcentaje de Venta / Total Vendido

Luego de arrancar el registro de las ventas, la siguiente tabla muestra las ventas registradas:

Ventas

No. De Venta	Artículo	Cantidad

Por lo tanto, el sistema llevará los porcentajes de venta por artículo hasta ese momento:

- Cantidad de artículos vendidos: v1(8) v2(3) v3(6) Total artículos vendidos = 17
- Cantidad de artículos vendidos por clave '0101': 1+4 = 5 % ventas = 29.4
- Cantidad de artículos vendidos por clave '0102': 3+2 = 5 % ventas = 29.4
- Cantidad de artículos vendidos por clave '0401': 4+1+2 = 7 % ventas = 41.2

Catálogo de artículos

Articulo	ıtaje de Venta / Total Vendido

```
DROP TABLE IF EXISTS ventas;
DROP TABLE IF EXISTS articulos;

CREATE TABLE articulos(
  artclave CHAR(4) PRIMARY KEY,
  artItemsVendidos INT DEFAULT 0,
  artPorcentajeVenta DECIMAL(7,4) DEFAULT 0.0
);

CREATE TABLE ventas (
  ventaNumero INT,
  artclave CHAR(4) NOT NULL,
  ventaCantidad INT NOT NULL CHECK(ventaCantidad > 0),
  PRIMARY KEY(ventaNumero, artclave)
);
ALTER TABLE ventas ADD CONSTRAINT ventas_artclave_fkey FOREIGN KEY(artclave) REFERENCES
articulos(artclave)
  ON UPDATE CASCADE ON DELETE CASCADE;
```

Recordar:

```
SELECT (1*100.0) /8;
R.
12.5000000000000000   genera un dato NUMERIC

-- Función ROUND
SELECT ROUND(67.456,2) AS "Round upto 2 decimal";
SELECT ROUND(33.33,2) AS "Round upto 2 decimal";
-----------------------------------------------------------------------
```

-- Función Postgres que REGRESA TRIGGER

```
CREATE OR REPLACE FUNCTION funcCalculateSalesPercentage() RETURNS TRIGGER AS $$
DECLARE
        total INT := 0;
        TmpRow record;
BEGIN
        SELECT sum(ventaCantidad) INTO total FROM ventas;

        FOR TmpRow IN (SELECT artclave, sum(ventaCantidad) sales FROM ventas GROUP BY
        artclave) LOOP
                IF NOT EXISTS(SELECT * FROM articulos WHERE artclave = TmpRow.artclave)
        THEN
                        INSERT INTO articulos VALUES(TmpRow.artclave, default, default);
                END IF;

                UPDATE articulos
                SET artItemsVendidos = TmpRow.sales, artPorcentajeVenta = ROUND((TmpRow.sales *
        100.0) / total, 4)
                WHERE artclave = TmpRow.artclave;
        END LOOP;

        RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
```

-- Crear el TRGGER en la tabla

```
CREATE TRIGGER triggerCalculateSalesPercentage AFTER INSERT
        ON ventas
        FOR EACH STATEMENT
        EXECUTE PROCEDURE funcCalculateSalesPercentage();
```

-- TESTING

Probar tu código con el siguiente escenario.

```
DELETE FROM articulos;
DELETE FROM ventas;

INSERT INTO articulos VALUES('0101',default);
INSERT INTO articulos VALUES('0102',default);
INSERT INTO articulos VALUES('0401',default);

SELECT * FROM articulos ORDER BY artclave;
"0101"; 0; 0.0000
"0102"; 0; 0.0000
"0401"; 0; 0.0000
-----------------------

SELECT * FROM ventas ORDER BY ventaNumero;
-- empty set --
-----------------------------------------------

INSERT INTO ventas VALUES(1,'0101',1);
SELECT * FROM articulos ORDER BY artclave;
"0101"; 1; 100.0000
"0102"; 0;   0.0000
"0401"; 0;   0.0000

SELECT * FROM ventas ORDER BY ventaNumero;
1; "0101"; 1
-------------------------------------------

INSERT INTO ventas VALUES(1,'0102',3);
SELECT * FROM articulos ORDER BY artclave;
"0101"; 1; 25.0000
"0102"; 3; 75.0000
"0401"; 0;  0.0000

SELECT * FROM ventas ORDER BY ventaNumero;
```

1; "0101"; 1
1; "0102"; 3

INSERT INTO ventas VALUES(1,'0401',4);
SELECT * FROM articulos ORDER BY artclave;
"0101"; 1; 12.5000
"0102"; 3; 37.5000
"0401"; 4; 50.0000

SELECT * FROM ventas ORDER BY ventaNumero;
1; "0101"; 1
1; "0102"; 3
1; "0401"; 4

INSERT INTO ventas VALUES(2,'0102',2);
SELECT * FROM articulos ORDER BY artclave;
"0101"; 1; 10.0000
"0102"; 5; 50.0000
"0401"; 4; 40.0000

SELECT * FROM ventas ORDER BY ventaNumero;
1; "0101"; 1
1; "0102"; 3
1; "0401"; 4
2; "0102"; 2

INSERT INTO ventas VALUES(2,'0401',1);
SELECT * FROM articulos ORDER BY artclave;
"0101"; 1; 9.0909
"0102"; 5; 45.4545
"0401"; 5; 45.4545

SELECT * FROM ventas ORDER BY ventaNumero;
1; "0101"; 1
1; "0102"; 3
1; "0401"; 4
2; "0102"; 2
2; "0401"; 1

-- CAPTURAS DE PANTALLA DEL RESULTADO FINAL DE LAS TABLAS:

	ventanumero integer	artclave character(4)	ventacantidad integer
1	1	0101	1
2	1	0102	3
3	1	0401	4
4	2	0102	2
5	2	0401	1

	artclave character(4)	artitemsvendidos integer	artporcentajeventa numeric(7,4)
1	0101	1	9.0909
2	0102	5	45.4545
3	0401	5	45.4545

F I N
