

## Teoría.

* 1. Explique las dos Reglas de Integridad según Edgar Codd. Indique dos formas distintas de implementar la Regla de Integridad Referencial en un Motor de BD.
  2. Explique y ejemplifique al menos 3 objetos de BD que permitan implementar la funcionalidad de integridad de un Motor de DB.

1.1

Las reglas de integridad definidas por Edgar Codd son utilizadas para asegurar la integridad de los datos y relacionar entidades en una base de datos relacional, estas son la regla de integridad de las entidades y la regla de integridad referencial.

Regla de integridad de las entidades

Esta regla establece que no se permiten claves primarias (PK) nulas, por lo tanto, ningún componente de la clave primaria de una relación puede aceptar nulos. Ya que las claves primarias son el mecanismo de identificación en el modelo relacional estas no pueden ser nulas. Una base de datos relacional no admite registrar información acerca de algo que no se puede identificar, por lo tanto, todo registro tiene que poder ser identificado unívocamente.

Regla de integridad referencial

Esta regla establece que la base de datos no debe contener valores no nulos de clave foránea para los cuales no exista un valor concordante de clave primaria en la relación referenciada.

Para evitar que se viole esta regla se determinan acciones a llevar a cabo ante operaciones que puedan violar la integridad referencial como, por ejemplo:

¿Qué hacer si se intenta eliminar la clave primaria de un registro referenciado?

**RESTRICT**: no se permite la eliminación del registro “padre” referenciado.

**CASCADE**: se eliminan también los registros que la referencian.

**ANULACIÓN**: se le asigna nulo a todas las claves foráneas que referenciaban al registro padre.

¿Qué hacer si se intenta modificar la clave primaria de un registro referenciado?

**RESTRICT**: no se permite la modificación del registro “padre” referenciado.

**CASCADE**: se modifican también las claves foráneas que la referencian.

**ANULACIÓN**: se le asigna nulo a todas las claves foráneas que referenciaban al registro “padre”.

¿Qué hacer si se intenta insertar un registro con clave foránea que quiera referenciar una clave primaria inexistente?

**RESTRICT**: no se permite la inserción del nuevo registro ya que no existe un registro “padre” con la PK especificada.

Constraints

Para implementar la funcionalidad de integridad de un motor de base de datos se pueden utilizar **constraints**, estas son restricciones que se les puede imponer a los valores de cierto campo de una tabla. Todo valor ingresado debe cumplir las condiciones impuestas por el constraint.

Para asegurar la integridad de entidades se pueden definir **PRIMARY KEY CONSTRAINTS**. La primary key es el atributo o conjunto de atributos que identifican unívocamente a una fila.

Este constraint asegura que los datos pertenecientes a una misma tabla tienen una única manera de identificarse, o sea que cada fila de cada tabla tenga una primary key capaz de identificar unívocamente una fila y esa no puede ser nula

El **FOREING KEY CONSTRAINT** es utilizado para asegurar la integridad referencial de la base de datos, asegura la coherencia entre datos de 2 tablas.

La foreing key es un atributo que referencia a la primary key de otra tabla. Este constraint permite valores nulos, como por ejemplo el caso en el que un cliente no tenga asociada ninguna compra en una tabla compras, pero no permite ingresar valores de FK que no existan como PK en la tabla “padre” referenciada.

Trigger

A través de un **trigger** se podría implementar el control de integridad relacional entre tablas de distintas bases de datos, por ejemplo, ante un DELETE el trigger puede buscar si existe tal PK en otra base de datos antes de que se lleve a cabo el evento. Los motores de bases de datos relacionales no permiten implementar FK contra PKs de tablas de otras bases de datos, un Trigger nos lo permitiría.

Índice UNIQUE

El objeto **índice** **unique** es de clave única, no permite que haya más de una fila por clave. Por ello, puede utilizarse para asegurar la integridad de los datos en un motor de base de datos. Un ejemplo de su aplicación es que el motor crea un índice Unique al utilizar el constraint UNIQUE o al crear una PK, estableciendo la restricción de unicidad de estos, manteniendo la integridad del sistema.

## 2. Query

Mostrar Nombre, Apellido y promedio de orden de compra del cliente referido, nombre Apellido y promedio de orden de compra del cliente referente. De todos aquellos referidos cuyo promedio de orden de compra sea mayor al de su referente. Mostrar la información ordenada por Nombre y Apellido del referido.

El promedio es el total de monto comprado (p x q) / cantidad de órdenes.

Si el cliente no tiene referente, no mostrarlo.

Notas: No usar Store procedures, ni funciones de usuarios, ni tablas temporales.

SELECT

c1.lname AS Referido\_Apellido,

c1.fname AS Referido\_Nombre,

(SELECT SUM(i2.quantity \* i2.unit\_price) / COUNT(DISTINCT o2.order\_num)

FROM items i2 JOIN orders o2 ON (o2.order\_num = i2.order\_num)

WHERE o2.customer\_num = c1.customer\_num ) AS Promedio\_Referido,

referente.lname AS Referente\_Apellido,

referente.fname AS Referente\_Nombre,

referente.Promedio\_Referente

FROM customer c1

JOIN orders o1 ON (o1.customer\_num = c1.customer\_num)

JOIN items i1 ON (i1.order\_num = o1.order\_num)

JOIN (SELECT

c2.customer\_num, lname, fname,

SUM(i3.quantity \* i3.unit\_price) / COUNT(DISTINCT o3.order\_num) AS Promedio\_Referente

FROM customer c2

JOIN orders o3 ON (o3.customer\_num = c2.customer\_num)

JOIN items i3 ON (i3.order\_num = o3.order\_num)

GROUP BY c2.customer\_num, lname, fname

) referente ON (referente.customer\_num = c1.customer\_num\_referedBy)

GROUP BY c1.lname, c1.fname, referente.lname, referente.fname, referente.Promedio\_Referente

HAVING

(SELECT SUM(i2.quantity \* i2.unit\_price) / COUNT(DISTINCT o2.order\_num)

FROM items i2 JOIN orders o2 ON (o2.order\_num = i2.order\_num)

WHERE o2.customer\_num = c1.customer\_num )

>

referente.Promedio\_Referente

ORDER BY c1.lname, c1.fname

## 3. Store Procedure

Dada la siguiente tabla de auditoria:

CREATE TABLE audit\_fabricante(

nro\_audit BIGINT IDENTITY PRIMARY KEY,

fecha DATETIME DEFAULT getDate(),

accion CHAR(1) CHECK (accion IN ('I','O','N','D')),

manu\_code char(3),

manu\_name varchar(30),

lead\_time smallint,

state char(2),

usuario VARCHAR(30) DEFAULT USER,

);

Se pide realizar un proceso de “rollback” que realice las operaciones inversas a las leídas en la tabla de auditoría hasta una fecha y hora enviada como parámetro.

Si es una accion de Insert ("I"), se deberá hacer un Delete.

Si es una accion de Update, se deberán modificar la fila actual con los datos cuya accion sea "O" (Old).

Si la acción es un delete "D", se deberá insertar el registro en la tabla.

Las filas a “Rollbackear” deberán ser tomados desde el instante actual hasta la fecha y hora pasada por parámetro.

En el caso que por cualquier motivo haya un error, se deberá cancelar la operación completa e informar el mensaje de error.

CREATE PROCEDURE deshacer\_operaciones @fechaLimite DATETIME

AS

BEGIN

BEGIN TRY

DECLARE @nro\_audit BIGINT

DECLARE @fecha DATETIME

DECLARE @accion CHAR(1), @manu\_code CHAR(3), @state CHAR(2)

DECLARE @manu\_name VARCHAR(30), @usuario VARCHAR(20)

DECLARE @lead\_time SMALLINT

DECLARE cursor\_audit CURSOR FOR

SELECT nro\_audit, fecha, accion, manu\_code, manu\_name, lead\_time, state, usuario

FROM audit\_fabricante WHERE fecha < @fechaLimite

OPEN cursor\_audit

FETCH NEXT FROM cursor\_audit

INTO @nro\_audit, @fecha, @accion, @manu\_code, @manu\_name, @lead\_time, @state, @usuario

BEGIN TRANSACTION

WHILE @@FETCH\_STATUS = 0

BEGIN

IF(@accion = 'I')

DELETE FROM manufact WHERE manu\_code = @manu\_code

ELSE IF(@accion = 'D')

INSERT INTO manufact

(manu\_code, manu\_name, lead\_time, state, f\_alta\_audit, d\_usualta\_audit)

VALUES (@manu\_code, @manu\_name, @lead\_time, @state, @fecha, @usuario)

ELSE IF (@accion = 'N')

UPDATE manufact

SET

manu\_code = (SELECT manu\_code

FROM audit\_fabricante

WHERE nro\_audit = @nro\_audit AND accion = 'O'),

manu\_name = (SELECT manu\_name

FROM audit\_fabricante

WHERE nro\_audit = @nro\_audit AND accion = 'O'),

lead\_time = (SELECT lead\_time

FROM audit\_fabricante

WHERE nro\_audit = @nro\_audit AND accion = 'O'),

state = (SELECT state

FROM audit\_fabricante

WHERE nro\_audit = @nro\_audit AND accion = 'O'),

f\_alta\_audit = (SELECT f\_alta\_audit

FROM audit\_fabricante

WHERE nro\_audit = @nro\_audit AND accion = 'O'),

d\_usualta\_audit = (SELECT d\_usualta\_audit

FROM audit\_fabricante

WHERE nro\_audit = @nro\_audit AND accion = 'O')

WHERE manu\_code = @manu\_code

END

COMMIT TRANSACTION

END TRY

BEGIN CATCH

RAISERROR('Error al deshacer operaciones de fabricantes.', 16, 1)

ROLLBACK TRANSACTION

END CATCH

CLOSE cursor\_audit

DEALLOCATE cursor\_audit

END

## 4. Triggers

El responsable del área de ventas nos informó que necesita cambiar el sistema para que a partir de ahora no se borren físicamente las órdenes de compra sino que el borrado sea lógico.

Nuestro gerente solicitó que este requerimiento se realice con triggers pero sin modificar el código del sistema actual.

Para ello se agregaron 3 atributos a la tabla ORDERS, flag\_baja (0 false / 1 baja lógica), fecha\_baja (fecha de la baja), user\_baja (usuario que realiza la baja).

Se requiere realizar un trigger que cuando se realice una baja que involucre uno o más filas de la tabla ORDERS, realice la baja lógica de dicha/s fila/s.

Solo se podrán borrar las órdenes que pertenezcan a clientes que tengan menos de 5 órdenes. Para los clientes que tengan 5 o más ordenes se deberá insertar en una tabla BorradosFallidos el customer\_num, order\_num, fecha\_baja y user\_baja.

Nota: asumir que ya existe la tabla BorradosFallidos y la tabla ORDERS está modificada.

Ante algún error informarlo y deshacer todas las operaciones.

CREATE TRIGGER borrado\_de\_OC ON orders

INSTEAD OF DELETE AS

BEGIN

BEGIN TRY

DECLARE @order\_num SMALLINT

DECLARE @customer\_num SMALLINT

DECLARE @user VARCHAR(15) = SUSER\_SNAME()

DECLARE @fecha DATETIME = GETDATE()

DECLARE cursor\_borrado\_logico CURSOR FOR

SELECT order\_num, customer\_num FROM deleted

OPEN cursor\_borrado\_logico

FETCH NEXT FROM cursor\_borrado\_logico

INTO @order\_num, @customer\_num

WHILE @@FETCH\_STATUS = 0

BEGIN

IF(SELECT COUNT(DISTINCT order\_num) FROM orders WHERE customer\_num = @customer\_num) < 5

UPDATE orders

SET

flag\_baja = 1,

fecha\_baja = @fecha,

user\_baja = @user

WHERE order\_num = @order\_num

ELSE

INSERT INTO BorradosFallidos (customer\_num, order\_num, fecha\_baja, user\_baja)

VALUES (@customer\_num, @order\_num, @fecha, @user)

FETCH NEXT FROM cursor\_borrado\_logico

INTO @order\_num, @customer\_num

END

END TRY

BEGIN CATCH

RAISERROR('Ocurrió un error al realizar el borrado lógico.', 16, 1)

END CATCH

CLOSE cursor\_borrado\_logico

DEALLOCATE cursor\_borrado\_logico

END

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1.1** | **1.2** | **2** | **3** | **4** |
| **12** | **13** | **30** | **23** | **22** |
|  |  |  |  |  |