

* **Describa 3 diferentes formas de implementar el concepto de Dominio definido por Edgar Codd, en un motor de Base de datos Relacional.**
* **Defina que es una transacción y explique su relación con la funcionalidad de consistencia en un motor de BD Relacional.**
* **Query**

**Crear una consulta que devuelva: Para aquellos 3 estados de mayor facturación el Código de estado, número cliente, nombre, apellido, promedio de orden de compra por cliente, total comprado por cliente y total comprado por estado. Para el cálculo de lo facturado solamente tener en cuenta aquellas líneas de ítems que hayan facturado mas de $85.**

**Ordenar la consulta por el monto facturado total por estado en forma descendente y por monto facturado por cliente también en forma descendente.**

**Nota: No se pueden utilizar tablas temporales, ni funciones de usuario.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **state** | **customer\_num** | **fname** | **lname** | **Promedio x orden** | **Monto x Cliente** | **Monto x Estado** |
| CA | 116 | Jean | Parmelee | 4864 | 4864 | 12340 |
| CA | 117 | Arnold | Sipes | 2268 | 4536 | 12340 |
| CA | 106 | George | Watson | 1428 | 2856 | 12340 |
| NJ | 122 | Cathy | O Brian | 4500 | 4500 | 5470 |
| NJ | 119 | Bob | Shorter | 970 | 970 | 5470 |
| OK | 124 | Chris | Putnum | 4143 | 4143 | 4143 |

**4. Store Procedure**

Desarrollar un script para crear la tabla CuentaCorriente con la siguiente estructura:

Id BIGINT IDENTITY, fechaMovimiento DATETIME, customer\_num SMALLINT (FK), order\_num INT (FK), importe DECIMAL(12,2)

Desarrollar un stored procedure que cargue la tabla cuentaCorriente de acuerdo a la información de las tablas orders e ítems.

Por cada OC deberá cargar un movimiento con fechaMovimiento igual al order\_date y el importe = SUM(quantity\*total\_price) de cada orden

Por cada OC pagada cargar además un movimiento con fechaMovimiento igual al paid\_date y el importe = SUM (quantity\*total\_price\*-1) de cada orden

**5. Trigger**

**Dada la tabla CUSTOMER y la tabla CUSTOMER\_AUDIT**

customer\_num smallint not null **(PK)**

update\_date datetime not null **(PK)**

ApeyNom\_NEW varchar(40)

State\_NEW char(2)

customer\_num\_referedBy\_NEW smallint

ApeyNom\_OLD varchar(40)

State\_OLD char(2)

customer\_num\_referedBy\_OLD smallint

update\_user varchar(30) not null

**Ante deletes y updates de los campos lname, fname, state o customer\_num\_refered de la tabla CUSTOMER, auditar los cambios colocando en los campos NEW los valores nuevos y guardar en los campos OLD los valores que tenían antes de su borrado/modificación.**

**En los campos ApeyNom se deben guardar los nombres y apellidos concatenados respectivos.**

**En el campo update\_date guardar la fecha y hora actual y en update\_user el usuario que realiza el update.**

**Verificar en las modificaciones la validez de las claves foráneas ingresadas y en caso de error informarlo y deshacer la operación.**

**Notas: Asumir que ya existe la tabla de auditoría. Las modificaciones pueden ser masivas y en caso de error sólo se debe deshacer la operación actual.**

**Notas**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** |
|  |  |  |  |  |
|  |  |  |  |  |

**5) Trigger**

CREATE TRIGGER auditcs ON customer

after delete, update as

BEGIN

declare @customer\_num int,

@apeynomNew varchar(40), @stateNew char(2),

@customer\_num\_refered\_byNew int,

@apeynomOld varchar(40), @stateOld char(2),

@customer\_num\_refered\_byOld int

declare auditcr CURSOR FOR

SELECT d.customer\_num, i.lname + '' + i.fname, i.state, i.customer\_num\_referedby, d.lname + '' + d.fname, d.state, d.customer\_num\_referedby

FROM deleted d

LEFT JOIN inserted i on i.customer\_num = d.customer\_num;

OPEN auditcr

FETCH NEXT FROM auditcr

into @customer\_num,

@apeynomNew, @stateNew, @customer\_num\_refered\_byNew,

@apeynomOld, @stateOld, @customer\_num\_refered\_byOld;

WHILE (@@FETCH\_STATUS = 0)

BEGIN

begin try

begin transaction

if not exists(select 1 from inserted)

BEGIN

INSERT INTO customer\_audit(customer\_num, update\_Date, apeynom\_OLD state\_Old, customer\_num\_referedby\_OLD, update\_user)

VALUES (@customer\_num, getDate(), @apeynomOld, @stateOld,

@customer\_num\_refered\_byOld, SYSTEM\_USER)

END

ELSE

BEGIN

if not exists(select 1 from customer

where customer\_num = @customer\_num\_refered\_byNew)

THROW 50001, 'Referente inexistente', 1;

if not exists (select 1 from state where state = @stateNEW)

THROW 50002, 'Estado inexistente', 1;

INSERT INTO customer\_audit(customer\_num, update\_Date,apeynom\_NEW, state\_NEW,customer\_num\_referedby\_NEW, apeynom\_OLD, state\_Old, customer\_num\_referedby\_OLD, update\_user)

values (@customer\_num, getDate(),

@apeynomNew, @stateNew, @customer\_num\_refered\_byNew,

@apeynomOld, @stateOld, @customer\_num\_refered\_byOld, SYSTEM\_USER)

end

COMMIT TRANSACTION

end try

begin catch

ROLLBACK TRANSACTION

end catch

FETCH NEXT FROM auditcr

into @customer\_num,

@apeynomNew, @stateNew, @customer\_num\_refered\_byNew,

@apeynomOld, @stateOld, @customer\_num\_refered\_byOld;

end

CLOSE auditcr

DEALLOCATE auditcr

END;

**4) Stored Procedure**

CREATE TABLE cuentaCorriente(

id BIGINT identity(1,1) primary key,

fechaMovimiento datetime,

customer\_num smallint REFERENCES customer,

order\_num INT references orders,

importe decimal(12,2)

)

CREATE PROCEDURE procedure\_cuenta\_corriente AS BEGIN

INSERT INTO cuentaCorriente(fechaMovimiento, customer\_num, order\_num, importe)

select o.order\_date, o.customer\_num, o.order\_num,

sum(quantity \* unit\_price)

from orders o

join items i on o.order\_num = i.order\_num

groupy by o.order\_num, o.order\_date, o.customer\_num

UNION

select o.paid\_date, o.customer\_num, o.order\_num,

sum(quantity \* unit\_price -1)

from orders o

join items i on o.order\_num = i.order\_num

WHERE o.paid\_date is NOT NULL

GROUP BY o.order\_num, o.paid\_date, o.customer\_num

END

**3) Query**

SELECT c.state, c.customer\_num, c.fname, c.lname,

SUM(i.quantity \* i.unit\_price)/COUNT(DISTINCT o.order\_num) promedioXOrden,

SUM(i.quantity \* i.unit\_price) montoXCliente,

(SELECT SUM(i2.quantity \* i2.unit\_price)

FROM customer c2

JOIN orders o2 ON c2.customer\_num = o2.customer\_num

JOIN items i2 ON o2.order\_num = i2.order\_num

where state = c.state) montoXEstado

FROM customer c

JOIN orders o ON c.customer\_num = o.customer\_num

JOIN items i ON i.order\_num = o.order\_num

WHERE c.state IN (select top 3 c3.state

from customer c3

JOIN orders o3 ON c3.customer\_num = o3.customer\_num

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

GROUP BY c3.state

ORDER BY SUM(i3.quantity \* i3.unit\_price) DESC

)

GROUP BY c.state, c.customer\_num, c.fname, c.lname

HAVING SUM(i.quantity \* i.unit\_price) > 85

ORDER BY montoXestado DESC, montoXCliente DESC

**TEORIA**

**2.**

**Defina que es una transacción y explique su relación con la funcionalidad de consistencia en un motor de BD Relacional.**

Una transaccion es un conjunto de sentencias SQL que se ejecutan atomicamente en una unidad logica de trabajo. Partiendo de que una transaccion lleva la bas de datos de un estado correcto a otro estado correcto, el motor posee mecanismos de manera de garantizar que la operacion completa se ejecute o falle, no permitiendo que queden datos inconsistentes.   
Cada sentencia de alteracion de datos como insert, update o delete (algunas usadas en los ejercicios practicos de arriba) es una transaccion en si misma que es llamada singleton transaction

un ejemplo de transaction podria ser:

--aca se asume que la base de datos esta en un estado consistente

BEGIN TRANS;

insert 123

update abcd

insert 321

commit;

--aca se asume que la base de datos esta en un estado consistente

En cuanto a su relacion con la funcionalidad de consistencia los motores de db tienen varios mecanismos para asegurarse la consistencia de los datos.. en realidad, consistencia es un concepto muy parecido al de integridad, una busqueda en google nos diria que son lo mismo.. pero aca se puede hacer una divison , porque la integridad al nivel del mundo relacional esta definida por codd por dos reglas : regla integridad entidades y regla integridad referencial pero la consistencia la podemos plantear como que los datos de nuestra base de datos tienen que estar correctos en funcion a un determinado caso de negocio, que no tiene nada que ver con la integridad tradicional.

ej: si grabo en el sistema un ticket (cabecera, nroticket, cliente, hora, consumidor final,el detalle: caramelo, bombon etc,) cuando esto se graba, quiero que se grabe una fila por cada cosa : cabecera, nro ticket, cliente, y demas datos/operaciones como calcular el stock.

Todos estos movimientos se tienen que hacer como una unidad atomica de ejecucion, es decir, se tienen que ejecutar juntos.. si el sistema no graba todo eso, el dato me queda inconsistente si tu ticket tiene 10 renglones , en el detalle tiene que haber 10 movimientos de stock digamos.  
esto, es **consistencia.**Con este ejemplo podemos ver que la integridad, esta mas atado a las reglas de integridad de Edgar Codd y consistencia se ata mas al negocio porque la base de datos por si sola puede asegurar la integridad de las datos, gracias a las reglas de integridad que yo creo.. como los constraints , que se chequean todo el tiempo. Ahora la consistencia, no depende de las reglas de integridad.  
La **transaccion**, es el concepto mas importante que hace referencia a la consistencia de los datos

ejemplo en el BEGINS TRANS , COMMIT , END TRANS

**1.**

**Describa 3 diferentes formas de implementar el concepto de Dominio definido por Edgar Codd, en un motor de Base de datos Relacional.**

El dominio es el conjunto de valores posibles que puede tomar una columna (campo o atributo) de una tabla. Los dominios son la menor unidad de semántica de información desde el punto de vista del modelo, son atómicos, o sea que no se pueden "descomponer". En consecuencia un dominio es un conjunto de valores escalares, todos del mismo tipo. Observación: como nulo no es un valor, los dominios no contienen nulos.  
Ademas, es la menor unidad semantica de informacion, son atomicos (no se pueden descomponer sin perder significado)y sus conjuntos de valores escalares son de igual tipo.

3 Formas claras de implementarlo podrian ser las siguientes:

- Nombre de columna: definir un nombre de columna claramente para que todos entiendan que contiene ese campo  
- Tipos de datos: tales como INT, DATETIME, varchar, entre otros  
- Constraints: son las encargadas de restringir los valores posibles dentro del tipo de dato  
algunas constraints son:  
 . NULL/NOT NULL  
 . DEFAULT  
 . CHECK  
 . PRIMARY KEY, UNIQUE  
 . CLAVES FORANEAS (integridad referencial)