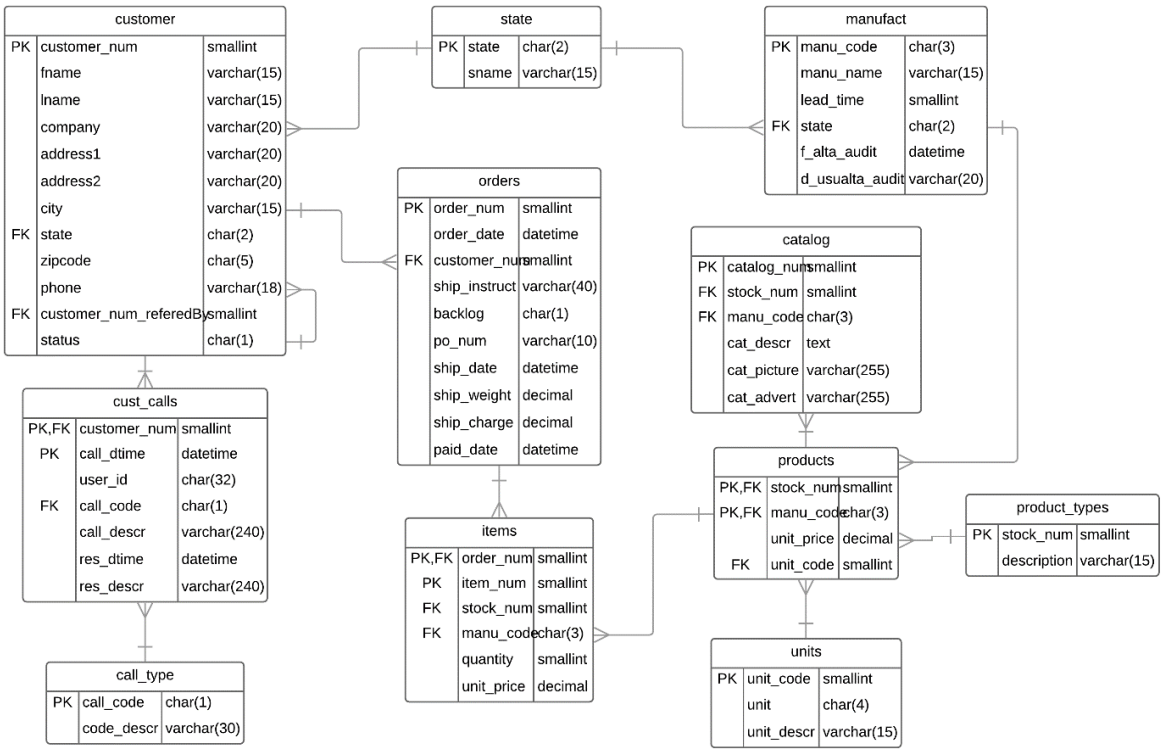


- **Describa 3 diferentes formas de implementar el concepto de Dominio definido por Edgar Codd, en un motor de Base de datos Relacional.**

- **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)