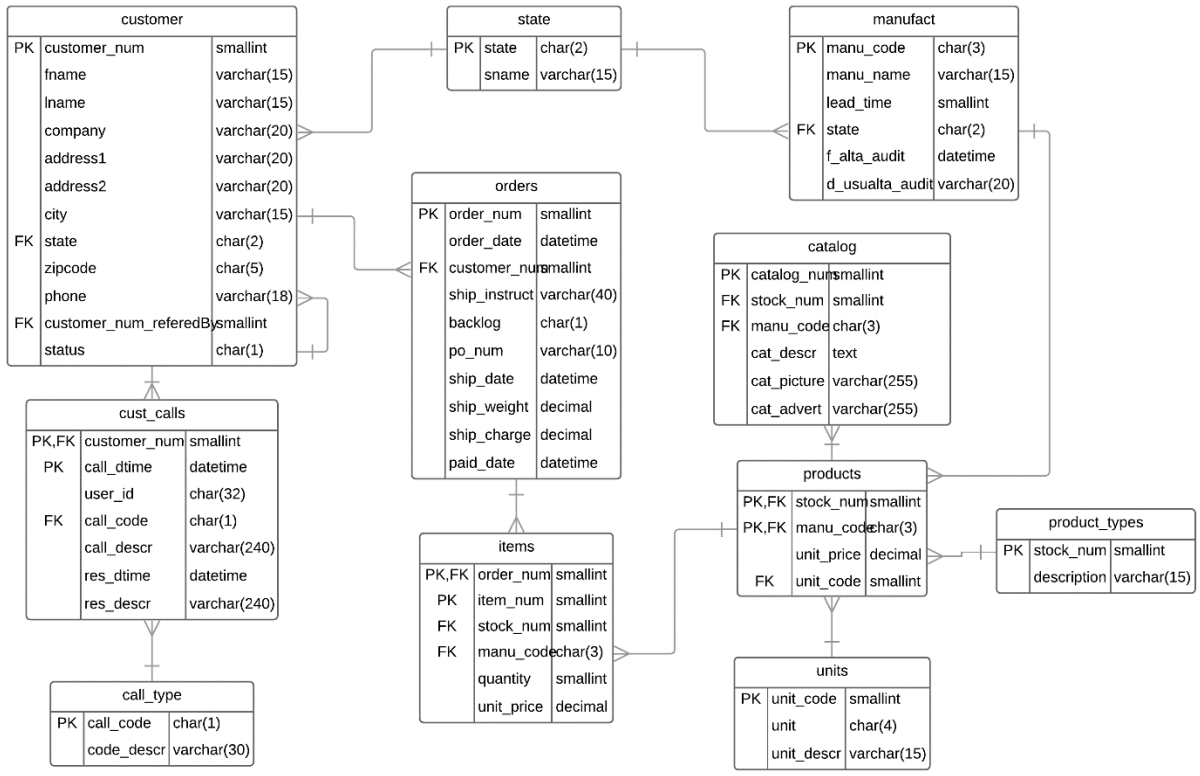

1. **Describa 3 diferentes formas de implementar el concepto de Dominio definido por Edgar Codd, en un motor de Base de datos Relacional.**

RTA:

El concepto de dominio no solo se refiere a los valores posibles que puede tomar un atributo y el tipo de esto, estos valores también están agrupados por ciertas relaciones o características. Por ejemplo, 1, 2, 4, 30 y 31 son valores numéricos enteros que pueden pertenecer a diferentes dominios disjuntos, 1, 2 y 4 pueden ser parte del dominio 'cantidad de ruedas de un vehículo' ,y 30 y 31 parte del dominio 'días en un año'.

Por ello, para implementar el concepto de dominio debe definirse estos grupos de valores con sus tipos, pero también con sus restricciones y nombres apropiados.

Forma 1: Tipos de datos

Restringen las características que pueden tener los datos a almacenar, por ejemplo 'int', 'decimal(n,m)' y 'smallint' son diferentes formas de definir un valor numérico con sus características.

No tiene sentido que use un 'decimal' para restringir la cantidad de ruedas de un auto y tampoco tiene sentido utilizar un 'int' para un valor monetario pequeño que podría incluir centavos.

Forma 2: Restricciones o Constraints

Indican ciertas reglas que deben cumplir los datos, por ejemplo NULL/NOT NULL, CHECK o las FKs.

Estas restricciones me permiten definir características de un dominio, por ejemplo no tiene sentido permitir nulos en el campo 'nro_teléfono' de un cliente de una central telefónica, siendo este el campo más importante.

Otro ejemplo es la restricción de que una columna pueda tomar determinados valores, por ejemplo que el tipo de documento solo pueda ser DNI o CUIT con '…CHECK (tipo_doc in ('DNI','CUIT'))` '

Forma 3: Nombres

El motor no puede validar que los nombres de las columnas tengan sentido en relación al contenido del dominio, pero estos deben especificarlo con claridad.

Por ejemplo, una columna llamada 'partes_del_cuerpo' indica que posibles valores para ella pueden ser 'mano', 'pie', etc, pero no tendrían sentido valores como 'servilleta' o 'cacatúa'.

De esta forma, aplicando nombres correctos, los contraints apropiados e indicando los tipos de datos acordes, se restringen y describen las características del dominio definido.

2. Explique cómo se relaciona el objeto Índice con la funcionalidad de integridad en un motor de Base de datos relacional.

RTA:

El objeto índice Unique es de clave única, es decir, no permite que haya más de una fila por clave. Por ello, puede utilizarse para asegurar la integridad de los datos en dónde se aplica, 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 , facilitando, además de la búsqueda de la fila, la restricción de unicidad de estos, manteniendo la coherencia del sistema.

Otro ejemplo de aplicación similar a la utilización de PKs compuestas es la creación de un índice unique sobre dos columnas cuyo par no deseo que aparezca más de una vez, por ejemplo, si no tiene sentido en mi sistema más de una orden de compra por día creo un índice unique sobre ambas columnas:
CREATE UNIQUE INDEX ix_orders ON orders(order_num,order_date)

---

3. Query

Seleccionar código de fabricante, nombre fabricante, cantidad de órdenes del fabricante, Monto Total Vendido del fabricante sum(quantity*total_price) y el promedio de las Montos vendidos de todos los Fabricantes.

SOLAMENTE MOSTRAR aquellos fabricantes cuyos Montos de ventas totales sean mayores al PROMEDIO de las ventas de TODOS los fabricantes.

Mostrar el resultado ordenado por cantidad total vendida en forma descendente.

IMPORTANTE: No se pueden usar procedures, ni Funciones de usuario.

manu_code	manu_name	CantOrdenes	Total vendido	Promedio de Todos
ANZ	Anza	11	11081.80	3972.85
SHM	Shimara	4	5677.91	3972.85

RTA:

```sql
SELECT m.manu_code, m.manu_name,
       count(distinct i.order_num) CantOrdenes,
       sum(i.quantity * i.unit_price)'Total vendido',
       (SELECT sum(i2.quantity * i2.unit_price) / count(distinct i2.manu_code) FROM items
i2) 'Promedio de Todos'
FROM manufact m
       left join items i on (i.manu_code = m.manu_code)
GROUP BY m.manu_code, m.manu_name
HAVING sum(i.quantity * i.unit_price) >
       (SELECT sum(i2.quantity * i2.unit_price) / count(distinct i2.manu_code) FROM items
i2)
ORDER BY sum(i.quantity * i.unit_price) DESC```
```

---

4. Stored Procedure

Crear un procedimiento que reciba como parámetro una FECHA.

Este deberá guardar en la tabla VENTASxMES el Monto total y cantidad total de productos vendidos para el Año y mes (yyyymm) de la fecha ingresada con la siguiente particularidad.

Asuma que existen 3 tipos de unidades de productos y las cantidades deberán ser "ajustadas" según su tipo:

1 unid (queda igual)

2 par (multiplicar por 2)

3 doc (multiplicar por 12)

Tabla VENTASxMES

anioMes	**decimal(6)**
stock_num	**smallint**
manu_code	**char(3)**
Cantidad	**int**
Monto	**decimal(10,2)**

El procedimiento debe manejar TODO el proceso en una transacción y deshacer todo en caso de error.

RTA:

```
DROP TABLE VENTASXMES
CREATE TABLE VENTASxMES(
    anioMes decimal(6),
    stock_num smallint,
    manu_code char(3),
    Cantidad int,
    Monto decimal(10,2)
);
GO

DROP PROCEDURE llenadoVentasPorMes
GO
```

```
CREATE PROCEDURE llenadoVentasPorMes @fechaEjecucion DATETIME AS
BEGIN
	-- declaración de variables
	DECLARE @unit char(4), @stock_num smallint, @manu_code char(3), @cantidad int,
@montoTotal decimal(10,2),@cantidadResultante int;

	-- declaración de cursor
	DECLARE ventaCursor
	CURSOR FOR
		SELECT u.unit, p.stock_num, p.manu_code, sum(i.quantity) cantidad, sum
(i.quantity * i.unit_price) montoTotal
		FROM orders o
			left join items i ON  (o.order_num = i.order_num)
			left join products p ON (i.stock_num = p.stock_num AND i.manu_code=
p.manu_code)
			left join units u ON (u.unit_code = i.unit_price)
		WHERE YEAR(o.order_date) = YEAR(@fechaEjecucion) AND
			MONTH(o.order_date) = MONTH(@fechaEjecucion)
		GROUP BY u.unit, p.stock_num, p.manu_code

	-- apertura de cursor
	OPEN ventaCursor
	FETCH NEXT FROM ventaCursor INTO @unit, @stock_num, @manu_code , @cantidad ,
@montoTotal;


	BEGIN TRY
		BEGIN TRANSACTION
			WHILE @@FETCH_STATUS = 0
			BEGIN
				IF @unit NOT IN ('unid', 'par', 'doc') throw 50000, 'Unidad
errónea', 2

				SET @cantidadResultante = CASE @unit
					WHEN 'unid' THEN @cantidad
					WHEN 'par' THEN @cantidad * 2
					WHEN 'doc' THEN @cantidad * 12
				END

				INSERT INTO VENTASxMES (anioMes, stock_num, manu_code,
Cantidad, Monto)
				VALUES(Year(@fechaEjecucion)*100 + MONTH(@fechaEjecucion),
					@stock_num, @manu_code, @cantidadResultante,
@montoTotal)

				FETCH NEXT FROM ventaCursor INTO @unit, @stock_num, @manu_code
, @cantidad , @montoTotal;
			END
		COMMIT TRANSACTION
	END TRY
	BEGIN CATCH
		ROLLBACK TRANSACTION
	END CATCH

	-- cierre de cursor
	CLOSE ventaCursor
	DEALLOCATE ventaCursor
END
GO
```

5. Triggers

Dada la vista:

```
Create view ProductosV
 AS SELECT p.stock_num, pt.description, p.manu_code, p.unit_price,
           p.unit_code, u.unit_descr
FROM products p JOIN product_types pt ON p.stock_num = pt.stock_num
                JOIN units u ON p.unit_code = u.unit_code;
```

Realizar un trigger que realice los INSERTS en esta vista.
En caso que ya exista el Producto, informar el mensaje "Clave duplicada".
Si no existe el fabricante o el tipo de producto informar el error.
Si no existe la unidad insertarla en la tabla correspondiente.
Tener en cuenta que los INSERTs pueden ser masivos y sólo se debe deshacer la operación del registro erróneo.

RTA:

```
DROP TRIGGER insertsProductosV
GO
```

```
CREATE TRIGGER insertsProductosV ON ProductosV
INSTEAD OF INSERT
AS
BEGIN
	-- Declarado de variables
	DECLARE @stock_num smallint ,@description varchar(15), @manu_code char(3),
@unit_price decimal,
	@unit_code smallint,@unit_descr varchar(15);

	-- Creación de cursor
	DECLARE insertedCursor
	CURSOR FOR
		SELECT stock_num, description, manu_code, unit_price, unit_code, unit_descr
		FROM inserted

	-- Apertura de cursor
	OPEN insertedCursor

	FETCH NEXT FROM insertedCursor INTO @stock_num, @description, @manu_code,
@unit_price, @unit_code, @unit_descr;

	WHILE @@FETCH_STATUS = 0
	BEGIN
		BEGIN TRY
			BEGIN TRANSACTION
				IF EXISTS(SELECT * from products where stock_num = @stock_num
AND manu_code = @manu_code)
					THROW 50000, 'Clave duplicada', 2

				IF @manu_code NOT IN (SELECT manu_code FROM manufact)
					THROW 51000, 'Fabricante no existente', 2

				IF @stock_num NOT IN (SELECT stock_num FROM product_types)
					THROW 51000, 'Tipo de Producto no existente', 2

				IF @unit_code NOT IN (SELECT unit_code FROM units)
					INSERT INTO units (unit_descr) VALUES (@unit_descr)

				INSERT INTO products
(stock_num,manu_code,unit_price,unit_code) VALUES
					(@stock_num, @manu_code, @unit_price, @unit_code)

			COMMIT TRANSACTION
		END TRY
		BEGIN CATCH
			ROLLBACK TRANSACTION
		END CATCH

		FETCH NEXT FROM insertedCursor INTO @stock_num, @description, @manu_code,
@unit_price, @unit_code, @unit_descr;
	END

	-- Cierre de cursor
	CLOSE insertedCursor
	DEALLOCATE insertedCursor
END
GO
```