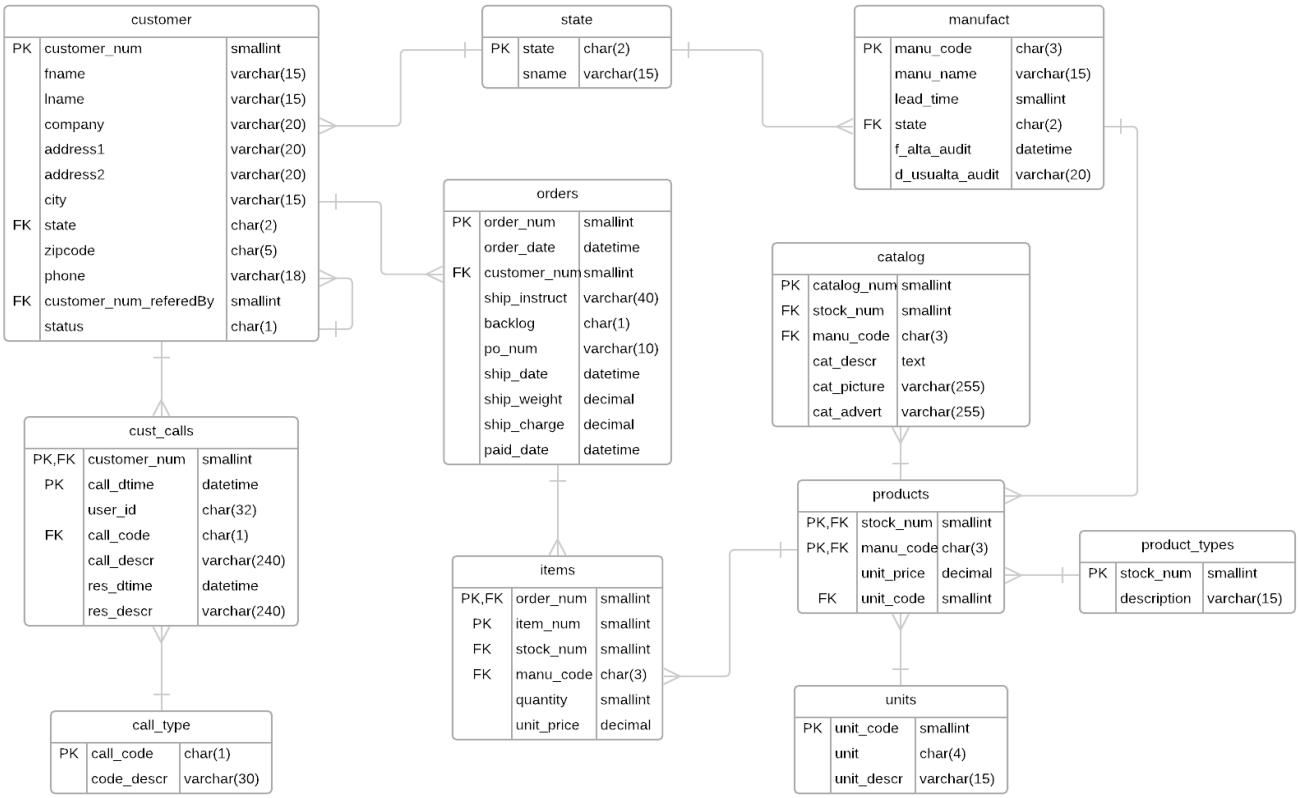
PARTE 1 – Teoría y SQL

a. Explique en menos de 15 renglones qué es Dominio y las diferentes formas de implementarlo en una BD.
b. En una carilla explique Índices: Qué son, para qué sirven, tipos, ventajas, desventajas, y su relación con la funcionalidad de integridad.
c. SQL



Obtener los Tipos de Productos, los montos totales vendidos por cliente y por sus referidos. Mostrar: descripción del Tipo de Producto, Nombre y apellido del cliente, monto total comprado de ese tipo de producto, Nombre y apellido de su cliente referido y el monto total comprado de su referido. Ordenado por Descripción y Nombre del cliente (Padre).
Nota: Si el Cliente no tiene referidos o sus referidos no compraron el mismo producto, mostrar ´--´ como nombre y apellido del referido y 0 (cero) en la cantidad vendida.

Description	Apellido	Nombre	Total comprado	Apellido Referido	Nombre Referido	Total comprado
3 golf balls	Putnum	Chris	1248	--	--	0
golf shoes	Jewell	Fred	600	Putnum	Chris	825
golf shoes	Putnum	Chris	825	--	--	0
metal Woods	Jewell	Fred	230	Putnum	Chris	2070
metal Woods	Jewell	Fred	230	Zuarez	Robert	210
metal Woods	Putnum	Chris	2070	--	--	0
tennis ball	Grant	Alfred	84	Parmelee	Jean	84
tennis ball	Higgins	Anthony	84	--	--	0

```
select tp.description, c.lname, c.fname, sum(i.unit_price*i.quantity) totalPadre
       ,coalesce(r.lname, '--') lNameHijo, coalesce(r.fname, '--') fnamehijo,
        coalesce(r.totalRef,0) totalHijo
  from product_types tp join items i on tp.stock_num = i.stock_num
                        join orders o on o.order_num = i.order_num
                        join customer c on o.customer_num = c.customer_num
                        left join (select c2.customer_num, c2.customer_num_referedBy,
                                          c2.lname, c2.fname, i2.stock_num,
                                          sum(i2.unit_price*i2.quantity) totalRef
                                     from customer c2 join orders o2
                                               on o2.customer_num = c2.customer_num
                                                 join items i2
                                                       on i2.order_num = o2.order_num
                                   group by c2.customer_num, c2.customer_num_referedBy,
                                            c2.lname, c2.fname, i2.stock_num) r
                                   on c.customer_num = r.customer_num_referedBy and
                                      tp.stock_num = r.stock_num
group by tp.description, c.lname, c.fname, r.lname, r.fname, r.totalRef
order by 1, 2
```

PARTE 2 –

d. Crear un procedimiento actualizaPrecios que reciba como parámetro una fecha a partir de la cual procesar los registros de una tabla Novedades que contiene los nuevos precios de Productos con la siguiente estructura/información.

FechaAlta, Manu_code, Stock_num, descTipoProducto, Unit_price, Unit_code

Por cada fila de la tabla Novedades

Si no existe el Fabricante, devolver un error de Fabricante inexistente y descartar la novedad.

Si no existe el stock_num (pero existe el Manu_code) darlo de alta en la tabla Product_types e insertar el nuevo par (stock_num, manu_code) en la tabla Products.

Si existe el Producto actualizar su precio
Si no existe, Insertarlo en la tabla de productos.

Nota: Manejar una transaccion por novedad y errores no contemplados.

```
DROP table ##Novedades;

create table ##Novedades (
Stock_num smallint,
Manu_code char(3),
descTipoProducto varchar(15),
unit_price decimal(6,2),
unit_code  smallint,
fechaAlta datetime);

create procedure ActualizaPrecios @fechaProceso date as
begin
    -- declaro variables y cursor
    declare @stock_num smallint,
            @Manu_code char(3),
            @descTipoProducto varchar(15),
            @unit_price decimal(6,2),
            @unit_code  smallint
                --
    declare cNovedades CURSOR FOR
              Select stock_num, Manu_code, descTipoProducto, unit_price
               from ##Novedades
              where fechaAlta > @fechaProceso;
    --
    open  cNovedades;
    fetch NEXT from cNovedades into @stock_num, @Manu_code, @descTipoProducto,
                                    @unit_price, @unit_code;
    WHILE @@FETCH_STATUS = 0
    Begin
    Begin try
        Begin TRAN
        if not exists (select 1 from manufact where manu_code = @Manu_code)
             throw 50000, 'Fabricante erroneo', 2

        if not exists (select 1 from product_types where stock_num = @stock_num)
            insert into product_types values (@stock_num, @descTipoProducto);

        if not exists (select 1 from products where stock_num = @stock_num and
                                                    manu_code = @Manu_code)
              insert into products values (@stock_num, @Manu_code, @unit_price, @unit_code);
        else
              update products set unit_price = @unit_price
               where stock_num = @stock_num and manu_code = @Manu_code;

        commit;
```

```
    end try
    begin catch
        rollback
    end catch
    fetch NEXT from cNovedades into @stock_num, @Manu_code, @descTipoProducto,
                                    @unit_price, @unit_code;
    End
    --
    close cNovedades
    Deallocate cNovedades
    --
end;
```

e. Triggers

Se desea llevar en tiempo real la cantidad de llamadas/reclamos (Cust_calls) de los Clientes (Customers) que se producen por cada mes del año y por cada tipo (Call_code).

Ante este requerimiento, se solicita realizar un trigger que cada vez que se produzca un Alta o Modificación en la tabla Cust_calls, se actualice una tabla donde se lleva la cuenta por Año, Mes y Tipo de llamada.

Ejemplo. Si se da de alta una llamada, se debe sumar 1 a la cuenta de ese Año, Mes y Tipo de llamada. En caso de ser una modificación y se modifica el tipo de llamada (por ejemplo por una mala clasificación del operador), se deberá restar 1 al tipo anterior y sumarle 1 al tipo nuevo. Si no se modifica el tipo de llamada no se deberá hacer nada.

Tabla ResumenLLamadas
Anio decimal(4) PK,
Mes decimal(2) PK,
Call_code char(1) PK,
Cantidad int

Nota: No se modifica la PK de la tabla de llamadas. Tener en cuenta altas y modificaciones múltiples.

```
create trigger custCallsTr ON Cust_calls
AFTER Insert, update as
begin
  --
  declare @anio decimal(4)
  declare @mes  decimal(2)
  declare @tipo char(1)
  --
  declare curIns CURSOR FOR
        select year(call_time), month(call_time), call_code
          from inserted

  declare curDel CURSOR FOR
        select year(call_time), month(call_time), call_code
          from deleted

    open curIns
    FETCH NEXT FROM curIns into @anio, @mes, @tipo
    while @@FETCH NEXT FROM_status = 0
    begin
      update resumenLlamadas
        set cantidad += 1
       where anio = @anio
        and mes = @mes
        and call_code = @tipo
      --
      FETCH NEXT FROM curIns into @anio, @mes, @tipo
    end
    close curIns
    Deallocate  curIns


    open curDel
    FETCH NEXT FROM curDel into @anio, @mes, @tipo
    while @@FETCH NEXT FROM_status = 0
    begin
      update resumenLlamadas
        set cantidad = cantidad - 1
       where anio = @anio
        and mes = @mes
        and call_code = @tipo
      --
      FETCH NEXT FROM curDel into @anio, @mes, @tipo
    end
    close curDel
    Deallocate  curDel
    --
    COMMIT
 --
end
```