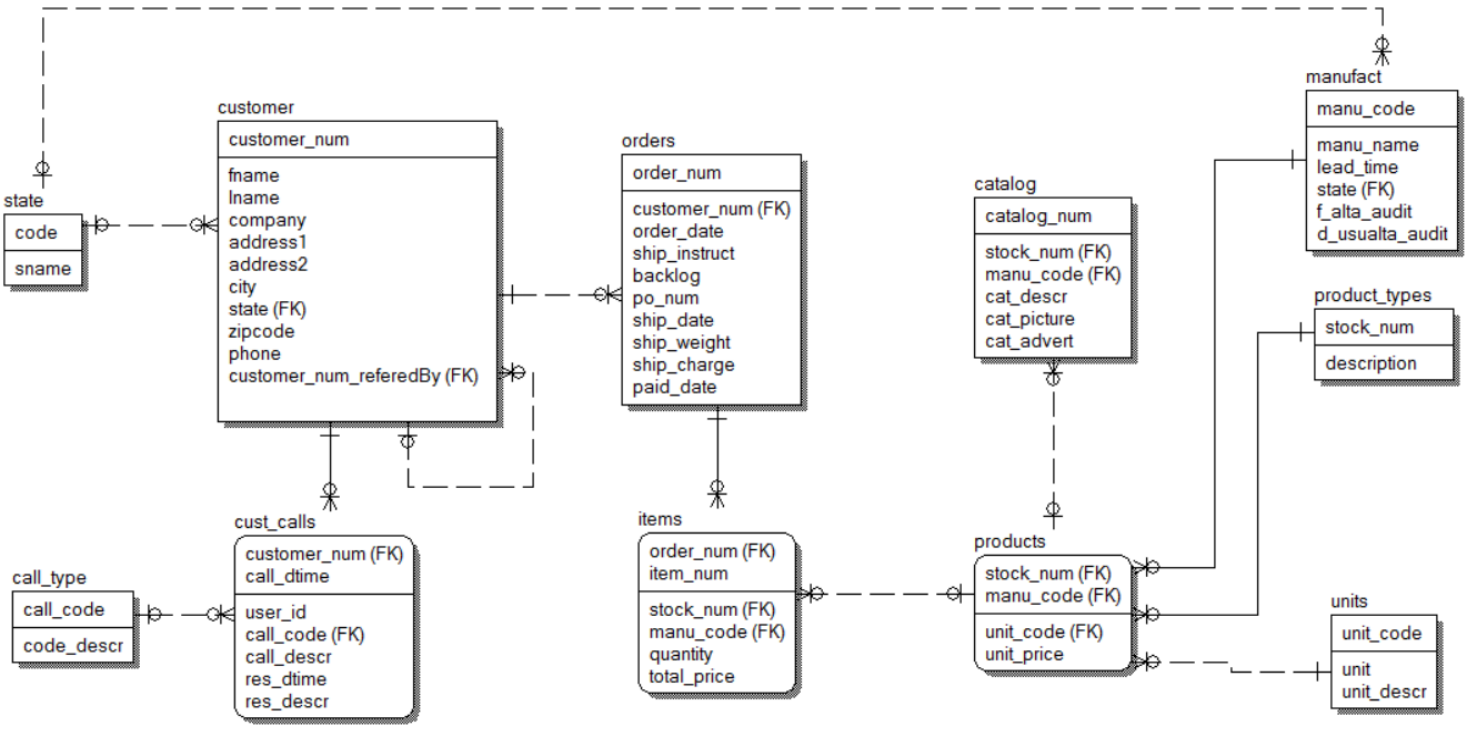
PARTE 1

Teoría

a. **En no más de 15 renglones explique todo lo relacionado con objeto Vista.**
b. **En una carilla explique la Funcionalidad de Seguridad de un RDBMS detallando someramente los objetos relacionados con la misma.**

Práctica



c. **Crear una consulta que devuelva:**
La siguientes cuatro atributos

Apellido, Nombre AS Cliente,
Suma de todo lo comprado por el cliente AS totalCompra,
Apellido, Nombre AS ClienteReferido,
Suma de lo comprado por el referido*0.05 AS totalComision

Consideraciones.

- **En el caso que un no tenga OCs deberá mostrar 0 en el campo totalCompra**
- **En el caso que un Cliente no tenga Referidos deberá mostrar al mismo con NULL en las columnas ClienteReferido y totalComision.**
- **Para calcular la comisión del cliente se deberán sumar (cant*precio) de todos los productos comprados por el ClienteReferido cuyo stock_num sea 1,4,5,6,9. La comisión es del 5%.**
- **Se deberá ordenar la salida por el Apellido y Nombre del Cliente.**

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

```
SELECT c1.lname+', '+c1.fname AS CLIENTE,
       COALESCE(SUM(i1.total_price*i1.quantity),0) AS totalCompra,
   ref.lname+', '+ref.fname AS Referido,
   ref.totalCompraReferido*0.05 AS totalComision
FROM customer c1 LEFT JOIN orders o1 ON
(c1.customer_num=o1.customer_num)
     LEFT JOIN items i1 ON (o1.order_num=i1.order_num)
 LEFT JOIN (SELECT c2.customer_num, c2.lname, c2.fname,
                SUM(i2.total_price*i2.quantity) AS totalCompraReferido,
                    c2.customer_num_referedBy
```

```
                FROM customer c2 JOIN orders o2
                            ON (c2.customer_num=o2.customer_num)
                JOIN items i2 ON (i2.order_num=o2.order_num)
                  WHERE c2.customer_num_referedBy IS NOT NULL
                  AND i2.stock_num IN (1,4,5,6,9)
              GROUP BY c2.customer_num, c2.lname, c2.fname,
                       c2.customer_num_referedBy) AS Ref
    ON (Ref.customer_num_referedBy=c1.customer_num)
GROUP BY c1.customer_num, c1.lname, c1.fname, Ref.customer_num,
         Ref.lname, Ref.fname, Ref.totalCompraReferido
ORDER BY 1
```

```
SELECT c1.lname+', '+c1.fname AS CLIENTE,
       COALESCE(SUM(i1.total_price*i1.quantity),0) AS totalCompra,
          ref.lname+', '+ref.fname AS Referido,
          ref.totalCompraReferido*0.05 AS totalComision
FROM customer c1 LEFT JOIN orders o1 ON (c1.customer_num=o1.customer_num)
     LEFT JOIN items i1 ON (o1.order_num=i1.order_num)
        Left join (select c2.lname, c2.fname, c2.customer_num_referedBy,
SUM(i2.total_price*i2.quantity) as totalCompraReferido
                    from customer c2 LEFT JOIN orders o2 ON
(c2.customer_num=o2.customer_num)
                                                LEFT JOIN items i2 ON
(o2.order_num=i2.order_num)
                where i2.stock_num in (1,4,5,6,9)
                group by c2.lname, c2.fname, c2.customer_num_referedBy) ref
                                     on (c1.customer_num=ref.customer_num_referedBy)
group by c1.lname, c1.fname, ref.lname, ref.fname, totalCompraReferido
order by 1
```

```
select cliente.cliente, cliente.totalcompra, ref.referido, ref.totalReferido*0.05
from
( SELECT c1.customer_num, c1.lname+', '+c1.fname AS CLIENTE,
       COALESCE(SUM(i1.total_price*i1.quantity),0) AS totalCompra
   FROM customer c1 LEFT JOIN orders o1 ON (c1.customer_num=o1.customer_num)
        LEFT JOIN items i1 ON (o1.order_num=i1.order_num)
  group by c1.customer_num, c1.lname, c1.fname ) cliente left join
 (select c2.lname+', '+c2.fname referido, c2.customer_num_referedBy,
SUM(i2.total_price*i2.quantity) as totalReferido
       from customer c2 LEFT JOIN orders o2 ON (c2.customer_num=o2.customer_num)
                                LEFT JOIN items i2 ON (o2.order_num=i2.order_num)
   where i2.stock_num in (1,4,5,6,9)
   group by c2.lname, c2.fname, c2.customer_num_referedBy) ref
        on (cliente.customer_num=ref.customer_num_referedBy)
    order by 1
```

PARTE 2

d. **Stored Procedures**

Desarrollar un stored procedure maneje la inserción o modificación de un producto determinado.
Parámetros de Entrada STOCK_NUM, MANU_CODE, UNIT_PRICE, UNIT_CODE, DESCRIPTION

Si existe el producto en la tabla PRODUCTS actualizar los atributos que no pertenecen a la clave primaria.
Si no existe el producto en la tabla PRODUCTS Insertar fila en la tabla, previamente validar lo siguiente:

- **EXISTENCIA de MANU_CODE en Tabla MANUFACT - Informando Error por Fabricante Inexistente.**
- **EXISTENCIA de STOCK_NUM en Tabla PRODUCT_TYPES - Si no existe Insertar un registro en la tabla STOCK_NUM, si existe realizar UPDATE del atributo 'description'.**
- **EXISTENCIA del atributo UNIT_CODE en la Tabla UNITS - Informando Error por Código de Unidad Inexistente.**

```
create or alter Procedure Producto_PR @STOCK_NUM smallint, @MANU_CODE char,
                                      @UNIT_PRICE decimal(6,2), @UNIT_CODE smallint,
                                     @DESCRIPTION varchar as
begin
      if exists (select 1 from products where stock_num = @STOCK_NUM and
manu_code=@manu_code)
         begin -- update
             update products
                set unit_price = @UNIT_PRICE,
                          unit_code = @UNIT_CODE
              where stock_num = @STOCK_NUM and manu_code=@manu_code;
         end
         else -- es un insert
         begin
             if not exists (select 1 from manufact where manu_code = @MANU_CODE)
                      throw 50000, 'Fabricante INEXISTENTE', 1

          if not exists (select 1 from units where unit_code = @UNIT_CODE)
                      throw 50000, 'Codigo de unidad INEXISTENTE', 1

                if not exists (select 1 from product_types where stock_num =
@stock_num)
                     insert into product_types (stock_num, description) values
(@stock_num,@DESCRIPTION);
                else
                     update product_types set description = @DESCRIPTION
                         where stock_num = @STOCK_NUM;

             insert into products (STOCK_NUM, MANU_CODE, UNIT_PRICE, UNIT_CODE )
                    values (@STOCK_NUM, @MANU_CODE, @UNIT_PRICE, @UNIT_CODE);
         end
end;
```

e. **Triggers**

Realizar un trigger de Insert sobre la siguiente Vista, insertando datos en las tablas correspondientes validando la existencia de sus respectivas claves primarias y que las mismas no sean NULAS.

```
CREATE VIEW v_Productos
(codCliente, nombre, apellido, codProvincia, fechaLlamado, usuarioId,
codTipoLlamada, descrLlamada, descrTipoLlamada)
SELECT c.customer_num, fname, lname, state, call_dtime,
       user_id, cc.call_code, call_descr, code_descr
FROM customer c JOIN cust_calls cc  ON (c.customer_num=cc.customer_num)
                JOIN call_type ct ON (cc.call_code=ct.call_code)
WHERE ct.call_code IN ('B','D','I','L','O')
AND state IN (SELECT code FROM state)
WITH CHECK OPTION
```

Tener en cuenta en el trigger la inserción de múltiples filas.

// Trigger version con Transaccion GLOBAL

```
CREATE or ALTER TRIGGER dbo.contactosTr
   ON  dbo.v_Productos
   INSTEAD OF INSERT AS
BEGIN
       --
       declare @codCliente       smallint
       declare @nombre           varchar(15)
       declare @apellido         varchar(15)
       declare @codProvincia     char(2)

       declare @fechaLlamado     datetime
       declare @usuarioId        char(32)
       declare @codTipoLlamada   char(1)
       declare @descrLlamada     varchar(40)
       declare @descrTipoLlamada varchar(30)

    DECLARE contactos_cur CURSOR FOR select * from inserted;
       open contactos_cur;
       --
       begin try
           FETCH NEXT FROM contactos_cur
             INTO @codCliente, @nombre, @apellido, @codProvincia, @fechaLlamado,
                  @usuarioId,  @codTipoLlamada, @descrLlamada, @descrTipoLlamada;

           WHILE @@FETCH_STATUS = 0
           BEGIN
              if not exists (select 1 from dbo.customer c
                             where c.customer_num = @codCliente)
                  insert into customer (customer_num, fname, lname,state)
                             values (@codCliente, @nombre, @apellido, @codProvincia)

              if not exists (select 1 from call_type ct
                                 where ct.call_code = @codTipoLlamada)
                     insert into call_type (call_code,code_descr)
                           values (@codTipoLlamada, @descrTipoLlamada)

              insert into cust_calls (customer_num, call_dtime, user_id,
                                                    call_code, call_descr)
                     values (@codCliente, @fechaLlamado, @usuarioId,
                                 @codTipoLlamada, @descrLlamada)

                  FETCH NEXT FROM contactos_cur
                  INTO @codCliente, @nombre, @apellido, @codProvincia, @fechaLlamado,
                       @usuarioId,  @codTipoLlamada, @descrLlamada, @descrTipoLlamada;
           END
       end try
       begin catch
           rollback;
           print 'Nro. Error:' + cast(ERROR_NUMBER() as varchar);
           print 'mensaje:' + ERROR_MESSAGE();
       end catch
       CLOSE contactos_cur
       DEALLOCATE contactos_cur
END
```

- **Trigger en versión con TRANSACCIONES por fila**

```
CREATE or ALTER TRIGGER dbo.contactosTr
   ON  dbo.v_Productos
   INSTEAD OF INSERT AS
BEGIN
       --
       declare @codCliente       smallint
       declare @nombre           varchar(15)
       declare @apellido         varchar(15)
       declare @codProvincia     char(2)

       declare @fechaLlamado     datetime
       declare @usuarioId        char(32)
       declare @codTipoLlamada   char(1)
       declare @descrLlamada     varchar(40)
       declare @descrTipoLlamada varchar(30)

    DECLARE contactos_cur CURSOR FOR select * from inserted;
       open contactos_cur;
       --
       FETCH NEXT FROM contactos_cur
             INTO @codCliente, @nombre, @apellido, @codProvincia, @fechaLlamado,
                  @usuarioId,  @codTipoLlamada, @descrLlamada, @descrTipoLlamada;
    --
       WHILE @@FETCH_STATUS = 0
       BEGIN
           if @@TRANCOUNT <= 0 -- si ya esta seteada la transaccion, no la creo
               begin transaction;
        begin try
                  if not exists (select 1 from dbo.customer c where c.customer_num =
@codCliente)
                     insert into customer (customer_num, fname, lname,state) values
(@codCliente, @nombre, @apellido, @codProvincia)

                  if not exists (select 1 from call_type ct where ct.call_code =
@codTipoLlamada)
               insert into call_type (call_code,code_descr) values (@codTipoLlamada,
@descrTipoLlamada)

                  insert into cust_calls (customer_num, call_dtime,    user_id,
call_code,       call_descr)
                                  values (@codCliente, @fechaLlamado, @usuarioId,
@codTipoLlamada, @descrLlamada)
              commit tran;
                  --
        end try
           begin catch
              rollback tran;
                 print 'Nro. Error:' + cast(ERROR_NUMBER() as varchar);
              print 'mensaje:' + ERROR_MESSAGE();
           end catch
              FETCH NEXT FROM contactos_cur
                  INTO @codCliente, @nombre, @apellido, @codProvincia, @fechaLlamado,
                       @usuarioId,  @codTipoLlamada, @descrLlamada, @descrTipoLlamada;
       END
       CLOSE contactos_cur
       DEALLOCATE contactos_cur
END
```

Parte 1		
a	b	c
Nota		

Parte 2	
d	e
Nota	