```
        IF NOT EXISTS (SELECT 1 FROM dbo.customer c WHERE c.customer_num =
@codCliente) BEGIN
            INSERT INTO CUSTOMER (customer_num, fname, lname, state)
                VALUES (@codCliente, @nombre, @apellido, @codProvincia)
        END
        IF NOT EXISTS (SELECT 1 FROM call_type ct WHERE ct.call_code =
@codTipoLlamada) BEGIN
            INSERT INTO call_type (call_code, code_descr)
                VALUES (@codTipoLlamada, @descrTipoLlamada);
        END
        INSERT INTO cust_calls (customer_num, call_dtime, user_id, call_code,
call_descr)
            VALUES (@codCliente, @fechaLlamado, @usuarioId,  @codTipoLlamada,
@descrLlamada)
    END TRY
    BEGIN CATCH
        PRINT 'hubo un error' -- esto es para debug
    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
```

COMMIT TRANSACTION

ROLLBACK TRANSACTION

PARTE 2 - (e) TRIGGER

```
CREATE TRIGGER dbo.customerTr
ON dbo.v_Productos
INSTEAD OF INSERT AS
BEGIN
DECLARE @codCliente          SMALLINT,
        @nombre              VARCHAR(15),
        @apellido            VARCHAR(15),
        @codProvincia        CHAR(2),
        @fechaLlamado        DATETIME,
        @usuarioId           CHAR(32),
        @codTipoLlamada      CHAR(1),
        @descrLlamada        VARCHAR(40),
        @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
    BEGIN TRY
    BEGIN TRANSACTION
        IF (@codCliente IS NULL) OR (@codProvincia IS NULL) OR (@codTipoLlamada IS NULL) OR
(@fechaLlamado IS NULL) BEGIN
            RAISERROR ('El código de cliente, la provincia, el tipo de llamada y la
fecha de llamado son OBLIGATORIAS.', -- Message text.  6, -- Severity.  1 -- State.);
        END
        IF (@codTipollamada NOT IN ('B','D','I','L','O')) BEGIN
            RAISERROR ('Codigo de Llamada inválido.', 16, 1)
        END
        IF NOT EXISTS (SELECT 1 FROM dbo.state where code = @codProvincia) BEGIN
            RAISERROR ('Codigo de Provincia inexistente.', 16, 1);
        END
```

PARTE 1 - (c) QUERY COMPLEJO

```
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
```

Universidad Tecnológica Nacional - FRBA | Ingeniería en Sistemas de Información | Gestión de
Apellido y Nombre Nro. Legajo Examen Parcial 2018

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 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 Codigo de Unidad Inexistente

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 (NO)
```

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

Parte 1		
a	b	c
Nota		

Parte 2	
d	e
Nota	

ARTE 1 - (c) QUER

```
SELECT c1.lname+', '
COALESCE(S
ref.lna
ref.
FROM cust
```

Universidad Tecnológica Nacional - FRBA | Ingeniería en Sistemas de Información | Gestión de Datos
Apellido y Nombre Nro. Legajo Examen Parcial 2018-06-27

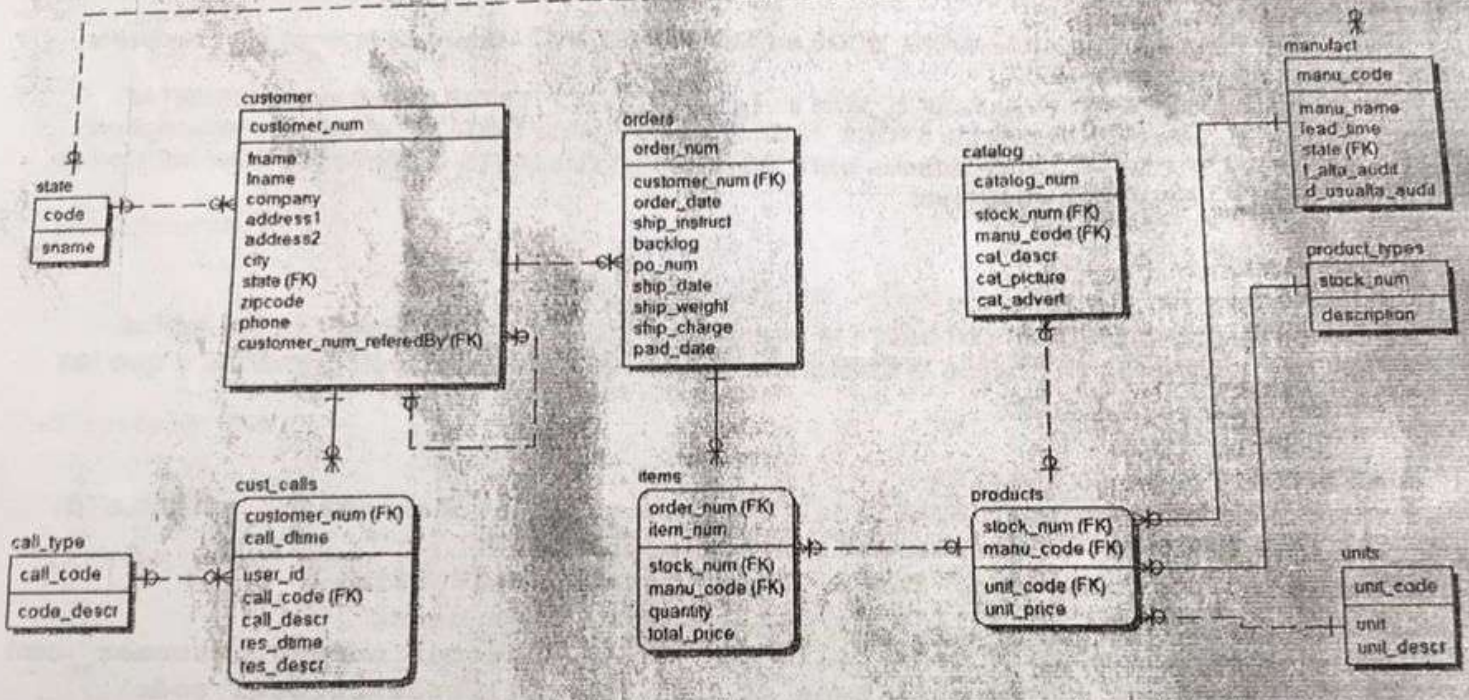
PARTE 1

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.

```
        ELSE BEGIN
            IF NOT EXISTS (SELECT 1 FROM products p
                           WHERE p.stock_num = @stock_num
                             AND p.manu_code = @manu_code) BEGIN

                RAISERROR('No existe el Producto', 16, 2)

            END
            IF NOT EXISTS (SELECT 1 FROM units u
                           WHERE u.unit_code = @unit_code) BEGIN

                INSERT INTO units (unit_code, unit_descr)
                    VALUES (@unit_code, @unit_descr)

            END

            UPDATE products
                SET unit_price = @unit_price,
                    unit_code = @unit_code
                WHERE stock_num = @stock_num
                  AND manu_code = @manu_code;

        END

        COMMIT TRANSACTION

    END TRY

    BEGIN CATCH

        [redacted]

    END CATCH

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

END

CLOSE curinsertados

DEALLOCATE curinsertados

END
```

ARCIAL 14-11-2018 · PARTE 2 - (3) TRIGGER

```
CREATE TRIGGER triggerInsert ON ProductosV
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @stock_num SMALLINT,
            @description VARCHAR(15),
            @manu_code CHAR(3),
            @unit_price DECIMAL(8,2),
            @unit_code SMALLINT,
            @unit_descr VARCHAR(15)
    DECLARE CURSOR cursor2 FOR
        SELECT * FROM inserted
    OPEN cursor2
    FETCH NEXT FROM cursor2
       INTO @stock_num, @description, @manu_code, @unit_price, @unit_code, @unit_descr
    BEGIN TRANSACTION
    WHILE (@@FETCH_STATUS = 0) BEGIN
        IF NOT EXISTS (SELECT * FROM manufact m
                       WHERE m.manu_code = @manu_code) BEGIN
            RAISERROR('No existe el fabricante' + CAST(@manu_code AS VARCHAR), 12, 1)
            [redacted]
        END
        IF NOT EXISTS (SELECT * FROM units u
                       WHERE u.unit_code = @unit_code) BEGIN
            INSERT INTO units (unit_code, unit_descr)
                VALUES (@unit_code, @unit_descr)
        END
        IF NOT EXISTS (SELECT * FROM product_types tp
                       WHERE tp.stock_num = @stock_num) BEGIN
            INSERT INTO product_types (stock_num, description)
                VALUES (@stock_num, @description)
        END
        INSERT INTO products SELECT * FROM inserted
        COMMIT TRANSACTION
        FETCH NEXT FROM cursor2
       INTO @stock_num, @description, @manu_code, @unit_price, @unit_code, @unit_descr
    END
    CLOSE cursor2
    DEALLOCATE cursor2
END
```

PARCIAL 14-11-2018 · PARTE 2 - (d) STORED PROCEDURE

```
CREATE TABLE CuentaCorriente (
    id BIGINT IDENTITY,
    fechaMovimiento DATETIME,
    customer_num SMALLINT REFERENCES customer,
    order_num INT REFERENCES orders,
    importe DECIMAL(12,2)
)

CREATE PROCEDURE cargarEnCuentaCorriente
AS
BEGIN
    DECLARE @customer_num SMALLINT,
            @order_num SMALLINT,
            @order_date DATETIME,
            @total DECIMAL(10,2),
            @paid_date DATETIME

    DECLARE CURSOR cursor1 FOR
        SELECT c.customer_num,
               o.order_num,
               o.order_date,
               o.paid_date,
               SUM(i.quantity * i.total_price)
        FROM orders o JOIN items i ON (o.order_num = i.order_num)

    OPEN cursor1

    FETCH NEXT FROM cursor1
        INTO @customer_num, @order_num, @order_date, @paid_date, @total

    WHILE (@@FETCH_STATUS = 0) BEGIN
        INSERT INTO CuentaCorriente (fechaMovimiento, customer_num, order_num, importe)
            VALUES (@order_date, @customer_num, @order_num, @total)
        IF (@paid_date IS NOT NULL) BEGIN
        INSERT INTO CuentaCorriente (fechaMovimiento, customer_num, order_num, importe)
                VALUES (@paid_date, @customer_num, @order_num, @total)
        END

            FETCH NEXT FROM cursor1
                INTO @customer_num, @order_num, @order_date, @paid_date, @total
    END

    CLOSE cursor1

    DEALLOCATE cursor1
END
```

PARCIAL 14-11-2018 · PARTE 1 - (c) QUERY COMPLEJO

- **Opción 1**

```
SELECT c.customer_num,
       c.fname + ' ' + c.lname AS nombreyapellido,
       s.sname,
       SUM(i.quantity * i.total_price) AS montoCliente,
       r.customer_num,
       r.nombreyapellido,
       r.montoReferente
FROM customer c JOIN state s ON (c.state = s.code)
                JOIN orders o ON (c.customer_num = o.customer_num)
                JOIN items i ON (i.order_num = o.order_num)
                LEFT JOIN (SELECT c.customer_num,
                                  c.fname + ' ' + c.lname AS nombreyapellido,
                                  SUM(i.quantity * i.total_price) montoReferente
                           FROM customer c
                               LEFT JOIN orders o ON (c.customer_num =
o.customer_num AND YEAR(o.order_date) = 2015)
                               LEFT JOIN items i ON (i.order_num = o.order_num)
                           GROUP BY c.customer_num,
                                    c.fname + ' ' + c.lname
                          ) r ON (c.customer_num_referedBy = r.customer_num)
WHERE o.order_date BETWEEN '2015-01-01' AND '2015-12-31'
GROUP BY c.customer_num,
         c.fname + ' ' + c.lname,
           s.sname,
           r.customer_num,
           r.nombreyapellido,
           r.montoReferente
HAVING SUM(i.quantity * i.total_price) > COALESCE(r.montoReferente, 0)
ORDER BY 3, 4 DESC
```

PARTE 2 – Practica

d. Stored Procedures (50 puntos)

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 items.

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

e. Triggers (50 puntos)

Dada la vista:

```
Create view ProductosV
(stock_num,description,manu_code,unit_price,unit_code,unit_descr)
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 de insert y update tal que:

Ante un INSERT:
Validar la existencia de claves primarias en las tablas relacionadas (fabricante, unit_code y product_types)
Si NO existe el Fabricante, devolver un mensaje de error y deshacer la transacción para ese registro.
En caso de no existir en Units y Product Types, insertar el registro correspondiente y continuar la operación.

Ante un UPDATE (no se actualizará la clave primaria de Products):
Si existe la combinación producto-fabricante, actualizar el precio y código de unidad previa validación en la tabla units, si no existe la unidad en la tabla, insertarla.
Si NO existe la combinación producto-fabricante devolver un mensaje de error y deshacer la transacción para ese registro.

Tener en cuenta que los UPDATEs e INSERTs pueden ser masivos (inserción de múltiples filas) y sólo se debe deshacer la operación del registro erróneo.

Parte 1		
a	b	c
Nota		

Parte 2	
d	e
Nota	

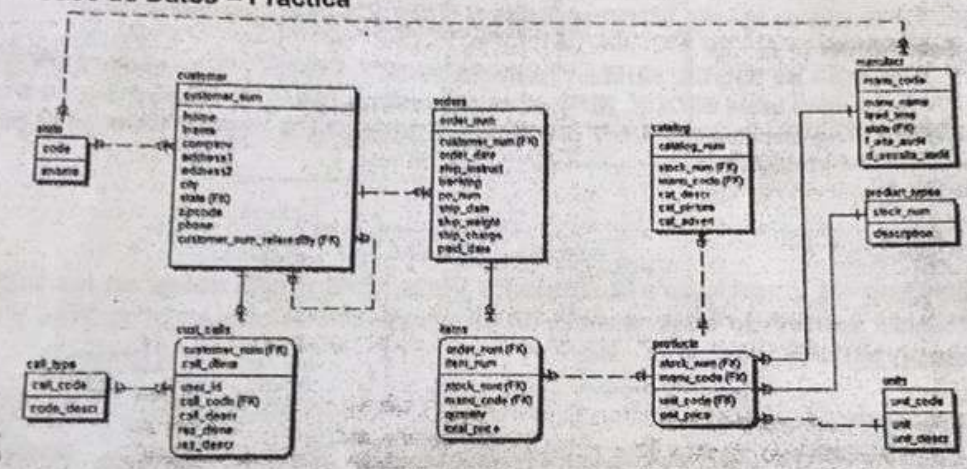
Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Segundo Recuperatorio – 11/07/2018 Apellido y Nombre:

Ing. En Sistemas de Información
Gestión de Datos
Legajo:

Parte 1-

1. **Base de Datos – Teoría (50 pts)**
   a. **Explique el objeto de BD Indice, cuando usarlo, ventajas y desventajas. (25 ptos)**
   b. **Describa brevemente la Arquitectura ANSI/SPARK.(25 ptos)**

2. **Base de Datos – Práctica**



a. **SQL (50 pts)**

Desarrollar una consulta que devuelva por cada fabricante dos filas, una con el producto más comprado y otra con el producto menos comprado, indicando el tipo de registro ('1-masComprado', '2-menosComprado').

Ejemplo

manu_code	menu_name	sname	stock_num	description	totalOrdenes	tipoRegistro
ANZ	Anza	California	5	tennis racquet	3904.80	1-masComprado
ANZ	Anza	California	304	watch	170.00	1-menosComprado
HRO	Hero	California	1	baseball gloves	250750.00	1-masComprado
HRO	Hero	California	309	ear drops	40.00	1-menosComprado

Por cada fabricante deberían haber como 2 filas, en el caso que tengan una solo producto comprado, el mismo se repetirá como masComprado y menosComprado.

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

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 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 Codigo de Unidad Inexistente

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.

Parte 1		
a	b	c
Nota		

Parte 2	
d	e
Nota	

Universidad Tecnológica Nacional - FRBA | Ingeniería en Sistemas de Información | Gestión de Datos
Apellido y Nombre Nro. Legajo Examen Parcial 2018-11-14

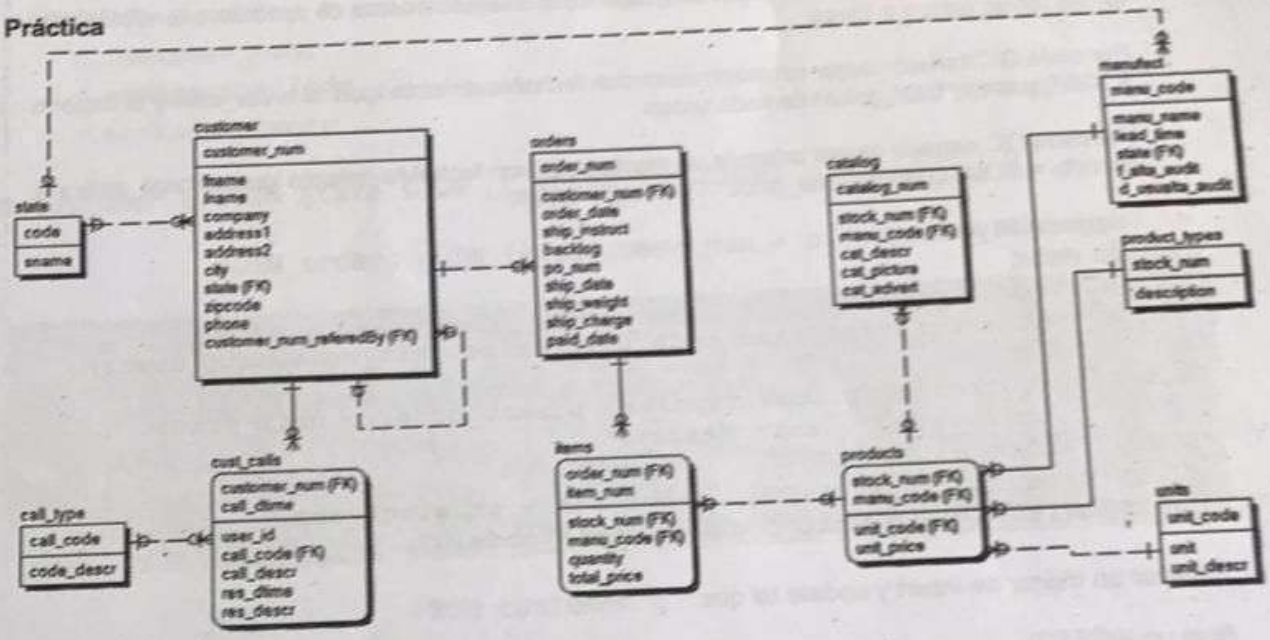
PARTE 1

Teoría (40 puntos)

a. En no más de 15 renglones el concepto y la estructura de una Relación según Edgar Codd.

b. En una carilla explique la Funcionalidad de Concurrencia de un RDBMS detallando someramente los conceptos relacionados con la misma.

Práctica



c. **Crear una consulta que devuelva: (60 puntos)**

Seleccionar aquellos clientes que hayan comprado (precio x cantidad) por un monto TOTAL mayor que sus referentes durante el año 2015.

Mostrar también aquellos Clientes que no han sido referidos por nadie.
Aparte tener en cuenta que el REFERENTE podría no haber comprado nada.

Mostrar número de cliente, nombre y apellido concatenados, la descripción del estado del cliente, su monto total comprado, del REFERENTE nombre y apellido concatenados y el monto total comprado.

El resultado deberá estar ordenado por descripción de estado y monto total comprado del cliente en forma descendente

customer_num — customer_num_referredBy

101 — NULL

102 — 101

El 101 no fue referido por nadie. Nadie es referente del 101.

El 102 fue referido por el 101. El 101 es referente del 102.

- Opción 2

```
SELECT a.customer_num,
       a.nombreyapellido,
       a.sname,
       a.montoCliente,
       B.nombreyapellido,
       B.montoReferente

FROM (SELECT c.customer_num,
             c.fname + " " + c.lname AS nombreyapellido,
             s.sname,
             SUM(i.quantity * i.total_price) AS montoCliente,
             c.customer_num_referedBy

      FROM customer c JOIN state s ON (c.state = s.code)
                      JOIN orders o ON (c.customer_num = o.customer_num)
                      JOIN items i ON (i.order_num = o.order_num)

      WHERE YEAR(o.order_date) = 2015

      GROUP BY c.customer_num
               c.fname + " " + c.lname,
               s.sname,
               c.customer_num_referedBy

      ) a

       LEFT JOIN (SELECT c.customer_num,
                         c.fname + " " + c.lname AS nombreyapellido,
                         SUM(i.quantity * i.total_price) AS montoReferente

                  FROM customer c LEFT JOIN orders o ON (c.customer_num =
o.customer_num AND YEAR(o.order_date) = 2015)

                                  LEFT JOIN items i ON (i.order_num =
o.order_num)

                  GROUP BY c.customer_num,

                           c.fname + " " + c.lname) b

      ON (a.customer_num_referedBy = B.customer_num)
WHERE a.monto_cliente > COALESCE(b.montoReferente, 0)
ORDER BY 3, 4 DESC
```

- **Opción 3**

```
SELECT c1.customer_num,
       nombreyapellido,
       c1.sname,
       montoTotal,
       c2.fname + ' ' + c2.lname,
       SUM(i.quantity * i.total_price) montoTotalRef

FROM (SELECT c.customer_num,
             c.fname + ' ' + c.lname AS nombreyapellido,
             s.sname,
             c.customer_num_referedBy,
             SUM(i.quantity * i.total_price) AS montoTotal

      FROM customer c JOIN orders o ON (c.customer_num = o.customer_num)
                      JOIN items i ON (i.order_num = o.order_num)
                      JOIN state s ON (c.state = s.code)

      WHERE o.order_date BETWEEN '2015-01-01' AND '2015-12-31'

      GROUP BY c.customer_num
               c.fname,
               c.lname,
               s.sname,
               c.customer_num_referedBy
      ) c1

           LEFT JOIN customer c2 ON (c1.customer_num_referedBy =

c2.customer_num)

           LEFT JOIN orders o ON (c2.customer_num = o.customer_num AND

YEAR(o.order_date) = 2015)

           LEFT JOIN items i ON (i.order_num = o.order_num)

WHERE YEAR(o.order_date) = 2015

GROUP BY c1.customer_num,
         nombreyapellido,
         c1.sname,
         c1.montoTotal,
         c2.fname,
         c2.lname

HAVING montoTotal > COALESCE(SUM(i.quantity * i.total_price), 0)

ORDER BY 3, 4 DESC
```

```
CREATE TRIGGER triggerUpdate
ON ProductosV
INSTEAD OF UPDATE
AS BEGIN
    DECLARE @stock_num SMALLINT,
            @description VARCHAR(15),
            @manu_code CHAR(3),
            @unit_price DECIMAL(8,2),
            @unit_code SMALLINT,
            @unit_descr VARCHAR(15)
    DECLARE CURSOR cursor3 FOR
        SELECT * FROM inserted
    OPEN cursor3
    FETCH NEXT FROM cursor3
      INTO @stock_num, @description, @manu_code, @unit_price, @unit_code, @unit_descr
    BEGIN TRANSACTION
    WHILE (@@FETCH_STATUS = 0) BEGIN
        IF NOT EXISTS (SELECT * FROM units u
                       WHERE u.unit_code = @unit_code) BEGIN
            INSERT INTO units (unit_code, unit_descr)
                VALUES (@unit_code, @unit_descr)
        END
        IF EXISTS (SELECT * FROM products p JOIN manufact m ON (p.manu_code =
m.manu_code)
                   WHERE m.manu_code = @manu_code) BEGIN
            UPDATE products SET unit_price = @unit_price,
                                unit_code = @unit_code
            WHERE manu_code = @manu_code
              AND stock_num = @stock_num
        END
        ELSE BEGIN
            RAISERROR('No existe la combinación producto-fabricante', 12, 1)
            [redacted]
        END
        COMMIT TRANSACTION
        FETCH NEXT FROM cursor3
       INTO @stock_num, @description, @manu_code, @unit_price, @unit_code, @unit_descr
    END
    CLOSE cursor3
    DEALLOCATE cursor3
END
```

Otra solución: los dos triggers en uno

```
CREATE TRIGGER productosV_Tr ON ProductosV

INSTEAD OF INSERT, UPDATE

AS BEGIN

    DECLARE @stock_num    SMALLINT,        @unit_price   DECIMAL(6,2),
            @description  VARCHAR (15),    @unit_code    SMALLINT,
            @manu_code    CHAR(3),         @unit_descr   VARCHAR(15)

    DECLARE curinsertados CURSOR FOR
        SELECT stock_num, description, manu_code, unit_price, unit_code, unit_descr
        FROM inserted

    OPEN curinsertados

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

    WHILE (@@FETCH_STATUS = 0)

    BEGIN

        BEGIN TRY

            BEGIN TRAN

        IF NOT EXISTS (SELECT TOP 1 NULL FROM deleted) BEGIN

            IF NOT EXISTS (SELECT 1 FROM manufact m
                           WHERE m.manu_code = @manu_code) BEGIN

                RAISERROR('No existe el fabricante', 16, 1)

            END

            IF NOT EXISTS (SELECT 1 FROM units u
                           WHERE u.unit_code = @unit_code) BEGIN

                INSERT INTO units (unit_code, unit_descr)
                    VALUES (@unit_code, @unit_descr)

            END

            IF NOT EXISTS (SELECT 1 FROM product_types pt
                           WHERE pt.stock_num = @stock_num) BEGIN

                INSERT INTO product_types (stock_num, description)
                    VALUES (@stock_num, @description)

            END

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

        END
```

Primer Recuperatorio – 28/11/2018 – PARTE PRÁCTICA

1.c SQL (60 pts)

Seleccionar código de fabricante, nombre fabricante, cantidad de órdenes del fabricante, cantidad total vendida del fabricante, promedio de las cantidades vendidas de todos los Fabricantes de todos aquellos fabricantes cuyas 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	promedioDeTodoslosFabricantes
ANZ	Anza	11	11081.80	3972.85
SHM	Shimara	4	5677.91	3972.85

2.a Stored Procedure (45 pts)

Crear un procedimiento procBorraOC que reciba un número de orden de compra por parámetro y realice la eliminación de la misma y sus ítems.

Deberá manejar una transacción y deberá manejar excepciones ante algún error que ocurra.

El procedimiento deberá guardar en una tabla de auditoria auditOC los siguientes datos order_num, order_date, customer_num, cantidad_items, total_orden (SUM(total_price)), cant_productos_comprados (SUM(quantity)), cantidad de ítems.

Ante un error deberá almacenar en una tablas erroresOC, order_num, order_date, customer_num, error_ocurrido VARCHAR(50).

2.e Triggers (55 pts)

En el caso que el cliente (customer_n...

Dada la siguiente tabla CURRENT_STOCK

```
create table CURRENT_STOCK (
        stock_num           smallint not null,
        manu_code           char(3) not null,
        CURRENT_AMOUNT      integer default 0,
        created_date        datetime not null, -- fecha de creación del registro
        updated_date        datetime not null, -- última fecha de actualización del registro
        PRIMARY KEY (stock_num, manu_code)
);
```

Crear un trigger que ante un insert, update o delete en la tabla ITEMS actualice la cantidad CURRENT_AMOUNT de la tabla CURRENT_STOCK de forma tal que siempre contenga el stock actual del par (stock_num, manu_code).

Si la operación es un INSERT se restará la cantidad QUANTITY al CURRENT_AMOUNT.
Si la operación es un DELETE se sumará la cantidad QUANTITY al CURRENT_AMOUNT.
Si la operación es un UPDATE se sumará la cantidad QUANTITY nueva y se restará la anterior al CURRENT_AMOUNT.

Si no existe el par (stock_num, manu_code) en la tabla CURRENT_STOCK debe insertarlo en la tablA CURRENT_STOCK con el valor inicial de 0 (cero) mas/menos la operación a realizar.

Tener en cuenta que las operaciones (INSERTs, DELETEs, UPDATEs) pueden ser masivas.

SQL (60 pts)

- *Solución 1*

```
SELECT m.manu_code,
       m.manu_name,
       COUNT(distinct i.order_num) CantOrdenes,
       SUM(quantity * total_price) total,
       (SELECT SUM(quantity * total_price) / COUNT(distinct manu_code)
        FROM items i)  promedioDeTodoslosFabricantes
FROM manufact m JOIN items i ON (m.manu_code = i.manu_code)
GROUP BY m.manu_code, m.manu_name
HAVING SUM(quantity * total_price) > (SELECT SUM(quantity * total_price) /
                                             COUNT(distinct manu_code)
                                      FROM items i)
```

- *Solución 2*

```
SELECT m.manu_code,
       m.manu_name,
       COUNT(distinct i.order_num) CantOrdenes,
       SUM(quantity * total_price) total,
       (SELECT AVG(A.total) total
        FROM (SELECT manu_code, SUM(quantity * total_price) total
              FROM items i
              GROUP BY manu_code)a)  promedioDeTodoslosFabricantes
FROM manufact m JOIN items i ON (m.manu_code = i.manu_code)
GROUP BY m.manu_code, m.manu_name
HAVING SUM(quantity * total_price) > (SELECT avg(A.total) total
                                      FROM (SELECT manu_code,
                                                   SUM(quantity * total_price) total
                                            FROM items i
```

2.e **Triggers (55 pts)**

```
CREATE TRIGGER itemsTR ON items
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
--
    DECLARE @stock_num smallint, @manu_code char(3), @quantityI smallint, @quantityD
smallint
    DECLARE Actualizados CURSOR FOR
        SELECT COALESCE(i.stock_num, d.stock_num) stock_num,
               COALESCE(i.manu_code, d.manu_code) manu_code,
               i.quantity, d.quantity
        FROM inserted i FULL JOIN deleted d ON (i.order_num = d.order_num AND
i.item_num = d.item_num)
    OPEN Actualizados
    FETCH Actualizados
        INTO @stock_num, @manu_code, @quantityI, @quantityD
    WHILE @@FETCH_STATUS=0
    BEGIN
        IF NOT EXISTS (SELECT 1 FROM CURRENT_STOCK p WHERE p.manu_code = @manu_code
AND p.stock_num = @stock_num)
            INSERT INTO CURRENT_STOCK (stock_num, manu_code, Current_Amount,
created_date, updated_date)
                VALUES (@stock_num, @manu_code, 0, GETDATE(), GETDATE());
        --
        UPDATE CURRENT_STOCK
            SET Current_Amount = Current_Amount - COALESCE(@quantityI, 0) +
COALESCE(@quantityD, 0),
                updated_date = getdate()
            WHERE stock_num = @stock_num AND manu_code = @manu_code;

        FETCH Actualizados
            INTO @stock_num, @manu_code, @quantityI, @quantityD;
    END

    CLOSE Actualizados
    DEALLOCATE Actualizados
END
```