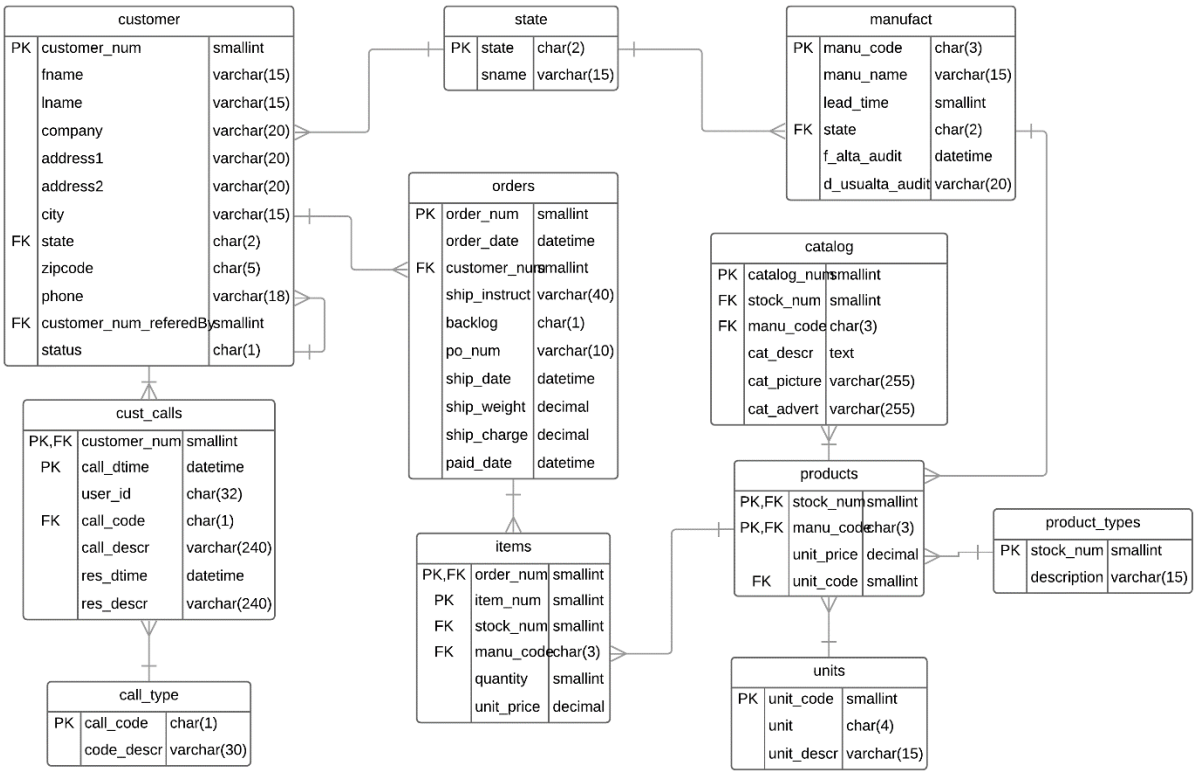
Universidad Tecnológica Nacional – F.R.B.A **Ing. en Sistemas de Información**
Gestión de Datos **1er parcial 14/07/2021**
Legajo: **Apellido y Nombre:**



1. Teoría.

1.1. Explique las dos Reglas de Integridad según Edgar Codd. Indique dos formas distintas de implementar la Regla de Integridad Referencial en un Motor de BD.

Según Codd hay dos reglas de integridad:

-La regla de integridad de las entidades, que establece que dentro de una clave primaria, ningún componente debe aceptar nulos, esto es, si se trata de un atributo, ese atributo no puede ser nulo. Si se trata de varios atributos, ninguno puede ser nulo.

-La regla de integridad referencial, que establece que si un atributo o conjunto de atributos en una tabla referencia a otra (FK) y son no nulos, dicho atributo o conjunto de atributos debe existir en la otra tabla como PK.

Una forma de implementar esta regla es a través de la constraint REFERENCES, que indica qué atributo referencia a qué tabla.

Otra forma (no recomendable) podría ser el uso de un trigger BEFORE para insertar (si no existiera) en la tabla referenciada la FK indicada antes del INSERT en la tabla que la referencia. Forzando así la integridad referencial.

1.2. Explique y ejemplifique al menos 3 objetos de BD que permitan implementar la funcionalidad de integridad de un Motor de DB.

-Un trigger es un objeto que ante algún evento determinado, dispara determinadas acciones. Permite implementar la funcionalidad de integridad de un Motor de DB, por ejemplo, haciendo que al insertar algo en una tabla, se inserten cosas necesarias automáticamente en otras.

-Un índice es una estructura asociada a una tabla en la que se guarda referencia a las posiciones relativas de la o las claves sobre las cuales se creó. Permite implementar la funcionalidad de integridad por el hecho de que al existir una constraint PK o UNIQUE, el motor inserta un registro en la tabla, independientemente de si existe o no una atributo con el mismo valor en la columna del constraint, pero al momento de insertarlo en el índice, detecta el duplicado y hace un rollback de la inserción anterior, disparando un error.

-Una vista es una query a una o varias tablas oculta bajo un nombre determinado, aporta a la integridad de base de datos en el sentido de que el usuario de la vista no tiene acceso a determinadas columnas de la o las tablas y por ende, no puede modificarlas, a su vez, puede agregarse una cláusula WITH CHECK OPTION que limita aún más las modificaciones que dicho usuario puede realizar sobre las tablas de base de dicha vista.

2. Query

Mostrar Nombre, Apellido y promedio de orden de compra del cliente referido, nombre Apellido y promedio de orden de compra del cliente referente. De todos aquellos referidos cuyo promedio de orden de compra sea mayor al de su referente. Mostrar la información ordenada por Nombre y Apellido del referido.

El promedio es el total de monto comprado (p x q) / cantidad de órdenes.

Si el cliente no tiene referente, no mostrarlo.

Notas: No usar Store procedures, ni funciones de usuarios, ni tablas temporales.

```
SELECT h.fname, h.lname, h.promedioReferido, p.fname, p.lname,
SUM(i1.quantity*i1.unit_price)/(SELECT Count(*) FROM orders o4 WHERE
o4.customer_num = p.customer_num) AS promedioReferente
FROM customer p JOIN orders o1 ON p.customer_num = o1.customer_num
                                JOIN items i1 ON i1.order_num = o1.order_num
                                JOIN (SELECT c.customer_num, c.customer_num_referedBy,
c.fname, c.lname, (SUM(i2.quantity*i2.unit_price)/(SELECT Count(*) FROM orders o3
WHERE o3.customer_num = c.customer_num)) AS promedioReferido
                                FROM customer c JOIN orders o2 ON c.customer_num =
o2.customer_num
                                JOIN items i2 ON i2.order_num = o2.order_num
                                                GROUP BY c.customer_num,
c.customer_num_referedBy, c.fname, c.lname) h ON h.customer_num_referedBy =
p.customer_num
GROUP BY h.fname, h.lname, h.promedioReferido, p.fname, p.lname, p.customer_num
HAVING SUM(i1.quantity*i1.unit_price)/(SELECT Count(*) FROM orders o5 WHERE
o5.customer_num = p.customer_num) < h.promedioReferido
ORDER BY h.fname, h.lname
```

3. Store Procedure

Dada la siguiente tabla de auditoria:

```
CREATE TABLE audit_fabricante(
	nro_audit BIGINT IDENTITY PRIMARY KEY,
	fecha DATETIME DEFAULT getDate(),
	accion CHAR(1) CHECK (accion IN ('I','O','N','D')),
	manu_code char(3),
	manu_name varchar(30),
	lead_time smallint,
	state char(2),
	usuario VARCHAR(30) DEFAULT USER,
);
```

Se pide realizar un proceso de “rollback” que realice las operaciones inversas a las leídas en la tabla de auditoría hasta una fecha y hora enviada como parámetro.
Si es una accion de Insert ("I"), se deberá hacer un Delete.
Si es una accion de Update, se deberán modificar la fila actual con los datos cuya accion sea "O" (Old).
Si la acción es un delete "D", se deberá insertar el registro en la tabla.
Las filas a “Rollbackear” deberán ser tomados desde el instante actual hasta la fecha y hora pasada por parámetro.
En el caso que por cualquier motivo haya un error, se deberá cancelar la operación completa e informar el mensaje de error.

```
CREATE PROCEDURE rollback_procedure @fechaHasta DATETIME
AS
BEGIN
BEGIN TRANSACTION
DECLARE @fecha DATETIME, @accion CHAR(1), @manu_code char(3),
 @manu_name varchar(30), @lead_time smallint, @state char(2), @usuario
VARCHAR(30)

DECLARE cursor_audit CURSOR FOR
SELECT fecha, accion, manu_code, manu_name, lead_time, state, usuario
FROM audit_fabricante WHERE  fecha > @fechaHasta AND fecha < Getdate()

OPEN cursor_audit
FETCH NEXT FROM cursor_audit into @fecha, @accion, @manu_code, @manu_name,
@lead_time, @state, @usuario

WHILE (@@FETCH_STATUS = 0)
	BEGIN
		BEGIN TRY
			IF (@accion = 'I')
				DELETE FROM manufact WHERE manu_code =
@manu_code;
			IF (@accion = 'O')
				UPDATE manufact
				SET manu_name = @manu_name, lead_time = @lead_time
				, state = @state, f_alta_audit = @fecha, d_usualta_audit =
@usuario
```

```
                    WHERE manu_code = @manu_code
                IF(@accion = 'D')
                    INSERT INTO manufact (manu_code, manu_name, lead_time,
state, f_alta_audit, d_usualta_audit)
                    VALUES (@manu_code, @manu_name, @lead_time, @state,
@fecha, @usuario)
            END TRY

            BEGIN CATCH
                PRINT 'Ha ocurrido el siguiente error: ' + ERROR_MESSAGE();
                PRINT 'No se ha realizado ninguna operación.'
                ROLLBACK TRANSACTION
            END CATCH
        FETCH NEXT FROM cursor_audit into @fecha, @accion, @manu_code,
@manu_name, @lead_time, @state, @usuario
        END
COMMIT TRANSACTION
CLOSE cursor_audit
DEALLOCATE cursor_audit
END
```

4. Triggers

El responsable del área de ventas nos informó que necesita cambiar el sistema para que a partir de ahora no se borren físicamente las órdenes de compra sino que el borrado sea lógico.

Nuestro gerente solicitó que este requerimiento se realice con triggers pero sin modificar el código del sistema actual.

Para ello se agregaron 3 atributos a la tabla ORDERS, flag_baja (0 false / 1 baja lógica), fecha_baja (fecha de la baja), user_baja (usuario que realiza la baja).

Se requiere realizar un trigger que cuando se realice una baja que involucre uno o más filas de la tabla ORDERS, realice la baja lógica de dicha/s fila/s.

Solo se podrán borrar las órdenes que pertenezcan a clientes que tengan menos de 5 órdenes. Para los clientes que tengan 5 o más ordenes se deberá insertar en una tabla BorradosFallidos el customer_num, order_num, fecha_baja y user_baja.

Nota: asumir que ya existe la tabla BorradosFallidos y la tabla ORDERS está modificada.

Ante algún error informarlo y deshacer todas las operaciones.

```
CREATE TRIGGER bajasDeOrdenes ON orders
INSTEAD OF DELETE
AS
BEGIN
BEGIN TRANSACTION
DECLARE @customer_num smallint, @order_num smallint
DECLARE cursor_bajas CURSOR FOR SELECT order_num, customer_num  FROM
deleted
```

```
OPEN cursor_bajas
FETCH NEXT FROM cursor_bajas INTO @order_num, @customer_num
WHILE(@@FETCH_STATUS = 0)
	BEGIN

	BEGIN TRY
		IF ((SELECT Count(*) FROM orders WHERE customer_num =
@customer_num) < 5)
			UPDATE orders SET flag_baja = 1, fecha_baja = GETDATE(),
user_baja = USER_NAME()
			WHERE order_num = @order_num
		ELSE
			INSERT INTO BorradosFallidos (customer_num, order_num,
fecha_baja, user_baja)
			VALUES(@customer_num, @order_num, GETDATE(),
USER_NAME())
	END TRY
	BEGIN CATCH
		PRINT 'Ha ocurrido el siguiente error: ' + ERROR_MESSAGE();
		PRINT 'No se ha realizado ninguna operación.'
		ROLLBACK TRANSACTION
	END CATCH

	FETCH NEXT FROM cursor_bajas INTO @order_num, @customer_num
	END
COMMIT TRANSACTION
CLOSE cursor_bajas
DEALLOCATE cursor_bajas
END
```

1.1	1.2	2	3	4
12	13	30	23	22