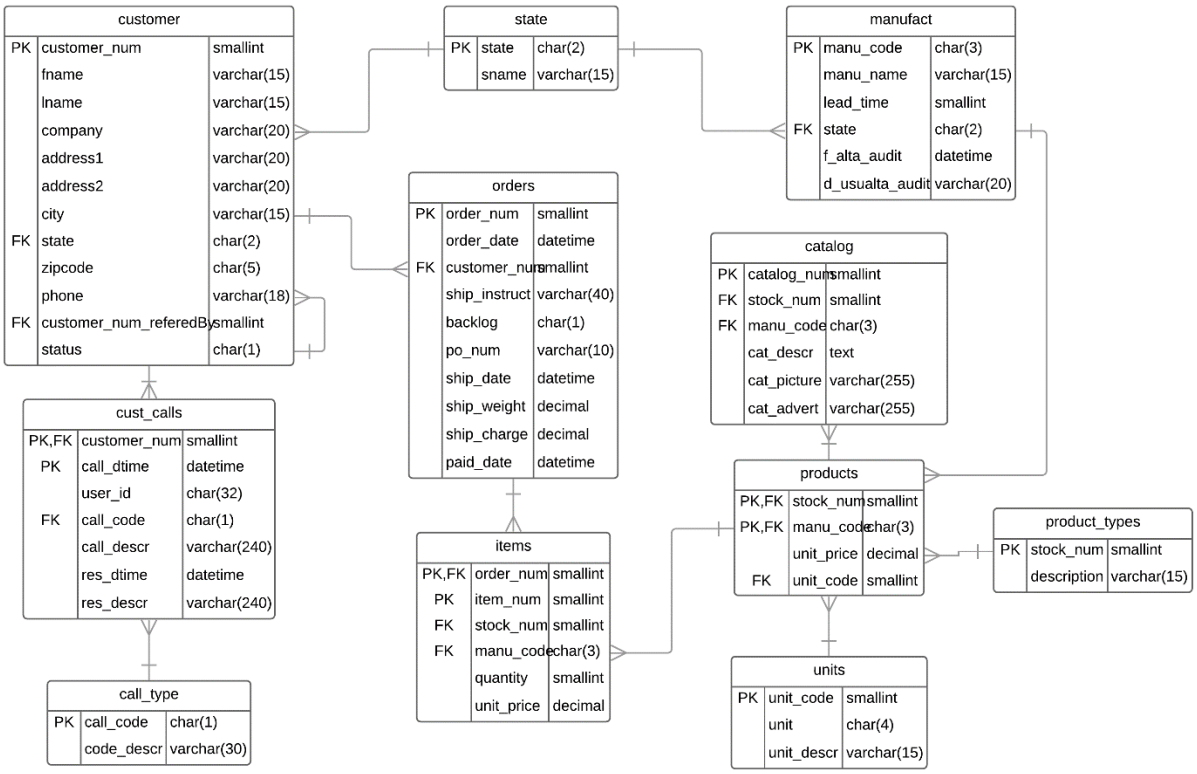
Universidad Tecnológica Nacional – F.R.B.A **Ing. en Sistemas de Información**
Gestión de Datos **1er parcial 14/07/2021**
Legajo: **Apellido y Nombre:**



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.

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

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.

3. Stored 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.

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.

1.1	1.2	2	3	4
12	**13**	**30**	**23**	**22**

RESOLUCION:

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.

Edgar Codd explica que se precisan tener dos reglas de integridad para así poder obtener precisión y por sobre todo consistencia en los datos almacenados en nuestra database.
Estas dos reglas son correspondientes a las Claves primarias y a las claves foráneas.
La clave primaria (PK) es un atributo o un conjunto de atributos para cada tupla de nuestra relación. Esta PK es el identificador único de cada tupla.

. **REGLA DE INTEGRIDAD DE LAS ENTIDADES:** Esta regla dictamina que ningún componente de la PK puede ser nulo. Ya que la tupla representa algo real, y seria contra intuitivo tener algo real que no se pueda identificar como tal.
Por otro lado, tenemos a las claves foráneas (FK) que es un atributo o conjunto de atributos de una relación que deben coincidir con los de la clave primaria de otra relación, siempre y cuando esta FK no tenga un valor nulo (lo cual esta permitido). Es importante entender que la llave foránea comparte dominio con la clave primaria a la que hace referencia, si no habría una inconsistencia.

. REGLA DE INTEGRIDAD DE REFERENCIAL: Esta regla dictamina que en una BBDD no deben haber FK no nulas para los cuales no exista un PK existente referenciada. Esto nos permite mantener datos consistentes.

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

- Constraints: Las constraints o restricciones son limitaciones que le ponemos al dominio que puede tomar un atributo de cierta tabla. Pudiendo así declarar a un campo como llave foránea que referencia a otra tabla, y ya en ese momento le estas diciendo que puede ser null o un valor de las PKs de la segunda tabla. Otro caso es el de declarar a un campo como PK, en este caso vamos a cumplir la Regla de integridad de las entidades.

```
-- EJEMPLO
CREATE TABLE avion(
	nro_serie int PRIMARY KEY,
	id_motor int REFERENCES motor(id_motor)
)
```

- Secuencias: Las secuencias son muy importantes a la hora de tener entidades en nuestra DB que no tengan un campo único que lo identifiquen del resto. En estos casos podemos usar las secuencias para auto rellenar la PK con un id secuencial que podría ser por ejemplo una secuencia que empiece en el int 1 y que vaya sumando de a uno. Es muy importante que la BBDD posea este feature, caso contrario podríamos emularlo por nuestra cuenta, pero debido a la concurrencia de usuarios sería un trabajo arduo.

```
-- EJEMPLO
CREATE TABLE avion(
	id_avion int IDENTITY(1,1),
	id_motor int REFERENCES motor(id_motor)
)
```

- Tablas: Es la unidad básica para almacenar datos dentro de esta es donde tendremos las distintas filas con su correspondiente clave primarias. Si no fuera por la existencia del objeto de tablas no se podría implementar la funcionalidad de integridad sencillamente porque ambas reglas de integridad dependen de las filas de x tabla.

2) Query:

```
SELECT
    referido.lname apellido_referido,
    referido.fname nombre_referido,
    SUM(i.quantity * i.unit_price)/COUNT(DISTINCT o.order_num) promedio_referido,
    referente.lname apellido_referente,
    referente.fname nombre_referente,
    referente.promedio promedio_referente
FROM customer referido
    JOIN orders o ON o.customer_num=referido.customer_num
    JOIN items i ON o.order_num=i.order_num
    JOIN (
    SELECT
        c.customer_num,
        c.lname,
        c.fname,
        SUM(i2.quantity * i2.unit_price)/COUNT(DISTINCT o2.order_num) promedio
    FROM customer c
        JOIN orders o2 ON o2.customer_num=c.customer_num
        JOIN items i2 ON o2.order_num=i2.order_num
    GROUP BY c.customer_num, c.lname, c.fname
) referente ON referente.customer_num=referido.customer_num_referedBy
WHERE referido.customer_num_referedBy IS NOT NULL
GROUP BY referido.customer_num, referido.lname, referido.fname,
        referente.customer_num, referente.fname, referente.lname, referente.prome
dio
HAVING
    SUM(i.quantity * i.unit_price)/COUNT(DISTINCT o.order_num)
    >
    referente.promedio
ORDER BY nombre_referido, apellido_referido
```

3) Stored Procedure:

```
create procedure rollbackFabricantes @fechaRollback datetime as
begin
	begin try
 -- declaro variables y cursor
	declare @accion CHAR(1),
	@manu_code char(3),
	@manu_name varchar(30),
	@lead_time smallint,
	@state char(2),
	@usuario VARCHAR(30),
	@fecha DATETIME
	--
	declare cursorAudit CURSOR FOR
	Select accion, manu_code, manu_name, lead_time,state, usuario
	from audit_fabricante
	where fecha > @fechaRollback
	ORDER BY fecha DESC;	-- Para deshacer los cambios como si fuera una pila,
elegimos deshacer primero los ultimos realizados. LIFO
	--
	open cursorAudit;
	fetch NEXT from cursorAudit into @accion, @manu_code, @manu_name, @lead_time,
@state, @usuario
	Begin TRAN;
	WHILE @@FETCH_STATUS = 0
	Begin
		if @accion = 'I'
			begin
				DELETE FROM manufact WHERE manu_code=@manu_code;
			end
		if @accion = 'O'
			begin
				UPDATE manufact SET manu_name=@manu_name,
lead_time=@lead_time, state=@state WHERE manu_code=@manu_code;
			end
		if @accion = 'D'
			begin
				INSERT INTO manufact
(manu_code,manu_name,lead_time,state,d_usualta_audit,f_alta_audit)
					VALUES(@manu_code, @manu_name, @lead_time, @lead_time,
@usuario, GETDATE());
					-- IMPORTANTE, inserto el usuario nuevo que da el alta,
y tomo la fecha actual para el mismo alta.
			end
		fetch NEXT from cursorAudit into @accion, @manu_code, @manu_name,
@lead_time, @state, @usuario, @fecha
	End
	--
	close cursorAudit
	Deallocate cursorAudit
	commit;
	end try
	begin catch
			rollback;
			SELECT ERROR_MESSAGE() AS ErrorMessage;
	end catch
--
end;
```

4) Trigger:

```
create trigger softDeleteOrders ON orders
INSTEAD OF DELETE as
begin
 --
        begin try;
        declare @order_num smallint
        declare @customer_num smallint
        declare @cust_order_qty smallint
        --
        declare cursor_soft CURSOR FOR
        select order_num, customer_num
              from deleted
              WHERE flag_baja = 0;

        open cursor_soft;
        FETCH NEXT FROM cursor_soft into @order_num;
        begin tran;
        while @@FETCH_STATUS = 0
               begin
                      set @cust_order_qty = (SELECT COUNT(*) FROM orders WHERE
customer_num=@customer_num)
                      if @cust_order_qty < 5
                             UPDATE orders
                             SET flag_baja = 1, fecha_baja = GETDATE(), user_baja = USER
                             WHERE order_num=@order_num;
                      else
                      INSERT INTO borrados_fallidos
(customer_num,order_num,fecha_baja,user_baja)
                             VALUES (@customer_num,@order_num,GETDATE(), USER);
                      FETCH NEXT FROM cursor_soft into @order_num
               end
        close cursor_soft
        Deallocate cursor_soft
        COMMIT;
        end try
        begin catch
              rollback;
              SELECT ERROR_MESSAGE() AS ErrorMessage;
        end catch
        --
--
end
```