DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

Bases de datos

Grupo 7

Integrante	LU	Correo electrónico
Lavia, Alejandro	43/11	lavia.alejandro@gmail.com
Simón, Jorge		jorgesm595@gmail.com
Rey, Esteban	657/10	estebanlucianorey@gmail.com
Tripodi, Guido	843/10	guido.tripodi@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Ejercicio 1

1.1. Descripción de problema

Se solicitó la creación de una base de datos en donde una empresa denominada “Entretenimiento Completo S.A.” (ECSA), provee tarjetas de acceso personalizadas a un grupo de parques de diversiones en todo el mundo así como también a eventos especiales.
El mecanismo utilizado es de post-pago: usando dicha tarjeta, el titular de la misma puede acceder a las diversas atracciones de los parques de diversiones o a los eventos especiales.
A fin de mes el titular de la tarjeta recibe una factura con el detalle. El importe es debitado de su medio de pago. La factura es enviada al domicilio de facturación del cliente. Una vez debitado el pago.
Cada tarjeta es personal, ya que lleva además de los datos del titular de la misma y una foto. En caso de extravío la tarjeta deberá ser desactivada y en su lugar se le entregará otra. No puede haber dos tarjetas activas para un mismo cliente. Se guardan los datos personales de los clientes, como dirección, teléfonos, nombre y apellido, etc. En tiempo real, la empresa ECSA informa a los parques de diversiones y a los organizadores de eventos las tarjetas entregadas para que éstos puedan verificar en sus sistemas el ingreso a las atracciones. Las tarjetas poseen una categoría que permite a las empresas realizar algún descuento.
El cambio de categoría se produce luego de haber gastado una cantidad de dinero predeterminada en el año. El ascenso de categoría dura un año, si después de ese año no se mantiene un promedio Y de gasto mensual entonces se baja de categoría. Los parámetros X e Y dependen de la categoría.
Los parques y los eventos tienen un nombre y una ubicación (dirección),y el precio de acceso a los mismos. Este precio es diario. Los eventos tienen además un rango de fechas y son desarrollados por una empresa organizadora.

Ademas, es necesario que nuestra base de datos pueda responder a las siguientes consultas:

- Estadísticas: atracción que más facturó, parque que más facturó, atracción que más facturó por parque.
- Listado de facturas adeudadas
- Para cada cliente las atracciones más visitadas en rango de fechas
- Cambios de categorías de cliente en rango de fechas
- Atracciones con descuento para cada categoría.
- Empresa organizadora de eventos que tuvo mayor facturación.
- Desarrollar un procedimiento almacenado que verifique las categorías y real- ice el cambio de la misma si es necesario.
- Ranking de parques/atracciones con mayor cantidad de visitas en rango de fechas.

1.2. Modelo Entidad Relación

En esta primera etapa realizamos un diseño conceptual del problema utilizando la herramienta de Modelo de Entidad Relación. Este lo construimos con la técnica de Diagrama Entidad relación, a continuación presentamos el diagrama obtenido:

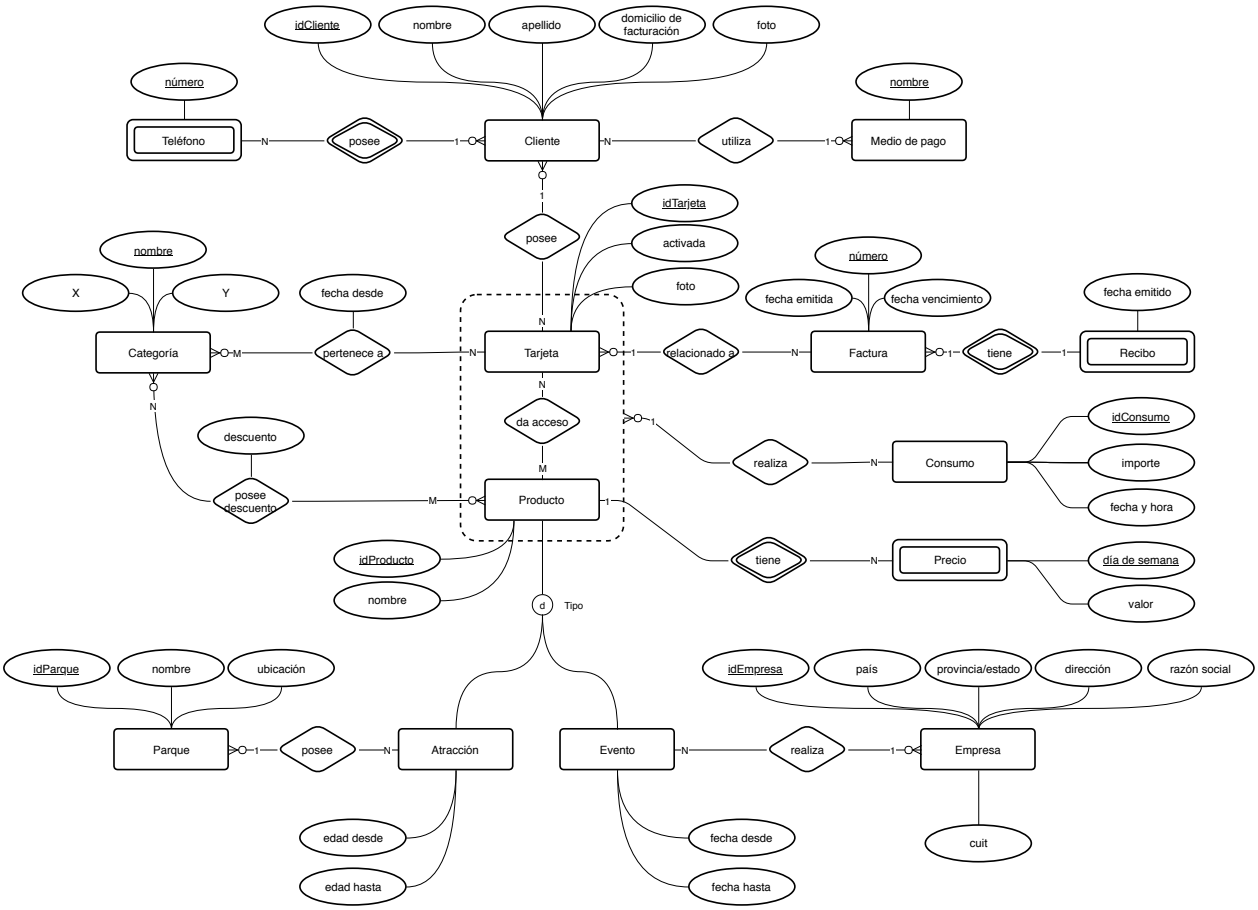

Diagrama Entidad Relación

1.3. Modelo Lógico Relacional

En base a nuestro diseño conceptual construido anteriormente, pasamos a un diseño lógico utilizando el Modelo Lógico Relacional que esta determinado por los siguientes esquemas de relación:

Cliente (idCliente, nombre, apellido, domicilioFact, foto, nombreMedioDePago)

- PK = {idCliente} FK = {nombreMedioDePago}
- Cliente.nombreMedioDePago debe estar en MedioDePago.nombreMedioDePago
- Cliente.foto debe ser una url a la fotografía

Telefono (idCliente, numero)

- PK = {(idCliente, numero)} FK = {idCliente}
- Telefono.idCliente debe estar en Cliente.idCliente

MedioDePago (nombreMedioDePago)

- PK = {nombreMedioDePago}

Tarjeta (idTarjeta, activada, foto, idCliente)

- PK = {idTarjeta} FK = {idCliente, nombreCategoria}
- Tarjeta.idCliente debe estar en Cliente.idCliente
- Tarjeta.idTarjeta debe estar en PerteneceA.idTarjeta

Categoria (nombre, x, y)

- PK = {nombre}
- El atributo **x** debe ser mayor o igual a 0
- El atributo **y** debe ser mayor o igual a 0

Factura (numero, fechaEmitida, fechaVencimiento, idTarjeta)

- PK = {numero} FK = {idTarjeta}
- Factura.idTarjeta debe estar en Tarjeta.idTarjeta
- El atributo **fechaVencimiento** debe ser mayor a **fechaEmitida**

Recibo(numeroFactura, fechaEmitido)

- PK = {numeroFactura} FK = {numeroFactura}
- Recibo.numeroFactura debe estar en Factura.numero

Producto (idProducto, nombre, tipoProducto)

- PK = {idProducto}
- (Producto.idProducto, α) debe estar en (Precio.idProducto, α) para todo $\alpha \in \{L, M, X, J, V, S, D\}$
- El atributo **tipoProducto** pertenece a $\{atraccion, evento\}$.

Precio (idProducto, diaSemana, valor)

- PK = {(idProducto, diaSemana)} FK = {idProducto}
- Precio.idProducto debe estar en Producto.idProducto
- El atributo **valor** debe ser mayor o igual a 0
- El atributo **diaSemana** es un carácter perteneciente a $\{L, M, X, J, V, S, D\}$.

Atraccion (idProducto, edadDesde, edadHasta, idParque)

- PK = {idProducto} FK = {idProducto, idParque}
- Atraccion.idProducto debe estar en Producto.idProducto
- Atraccion.idParque debe estar en Parque.idParque
- El atributo **edadDesde** debe ser menor o igual que el atributo **edadHasta**

Evento (idProducto, fechaDesde, fechaHasta, idEmpresa)

- PK = {idProducto} FK = {idProducto, idEmpresa}
- Evento.idProducto debe estar en Producto.idProducto
- Evento.idEmpresa debe estar en Empresa.idEmpresa
- El atributo **fechaDesde** debe ser menor o igual que el atributo **fechaHasta**

Parque (idParque, nombre, ubicacion)

- PK = {idParque}

Empresa (idEmpresa, cuit, razonSocial, pais, direccion, provinciaEstado)

- PK = {idEmpresa}
- Empresa.cuit es una cadena de 10 caracteres numéricos

Consumo (idConsumo, importe, fecha, hora, idProducto, idTarjeta)

- PK = {idConsumo} FK = {(idProducto, idTarjeta)}
- (Consumo.idProducto, Consumo.idTarjeta) debe estar en (DaAcceso.idProducto, DaAcceso.idTarjeta)

PerteneceA (nombreCategoria, idTarjeta, fechaDesde)

- PK = {(nombreCategoria, idTarjeta)} FK = {nombreCategoria, idTarjeta}
- PerteneceA.nombreCategoria debe estar en Categoria.nombre
- PerteneceA.idTarjeta debe estar en Tarjeta.idTarjeta

PoseeDescuento (nombreCategoria, idProducto, descuento)

- PK = {(nombreCategoria, idProducto)} FK = {nombreCategoria, idProducto}
- PoseeDescuento.nombreCategoria debe estar en Categoria.nombre
- PoseeDescuento.idProducto debe estar en Producto.idProducto
- El atributo **descuento** debe ser mayor a 0

DaAcceso (idTarjeta, idProducto)

- PK = {(idTarjeta, idProducto)} FK = {idTarjeta, idProducto}
- DaAcceso.idTarjeta debe estar en Tarjeta.idTarjeta
- DaAcceso.idProducto debe estar en Producto.idProducto

1.4. Queries

1.4.1. Estadísticas: atracción que mas facturó

```
-- Atracción que más facturo
---------------------------
SELECT
    producto.nombre AS Atraccion,
    SUM(consumo.importe) AS facturacionTotal
FROM
    mydb.Consumo consumo,
    mydb.Producto producto,
    mydb.Atraccion atraccion,
    mydb.Factura factura,
    mydb.Parque parque
WHERE
    producto.idProducto = atraccion.idProducto
        AND parque.idParque = atraccion.idParque
        AND producto.idProducto = consumo.idProducto
        AND factura.idTarjeta = consumo.idTarjeta
        AND factura.fechaEmitida > consumo.fechaYhora
        AND MONTH(consumo.fechaYhora) > (MONTH(factura.fechaEmitida) - 1)
        AND YEAR(consumo.fechaYhora) = YEAR(consumo.fechaYhora)
GROUP BY consumo.idProducto
HAVING SUM(consumo.importe) >= ALL (SELECT
        SUM(consumo1.importe)
    FROM
        mydb.Consumo consumo1,
        mydb.Atraccion atraccion1,
        mydb.Producto producto1,
        mydb.Factura factura1
    WHERE
        consumo1.idProducto != consumo.idProducto
            AND producto1.idProducto = atraccion1.idProducto
            AND consumo1.idProducto = producto1.idProducto
            AND consumo1.idTarjeta = factura1.idTarjeta
            AND factura1.fechaEmitida > consumo1.fechaYhora
            AND MONTH(consumo1.fechaYhora) > (MONTH(factura1.fechaEmitida) - 1)
            AND YEAR(consumo1.fechaYhora) = YEAR(consumo1.fechaYhora)
    GROUP BY consumo1.idProducto);
```

1.4.2. Estadísticas: parque que más facturó

```
-- Parque que más facturo
------------------------
SELECT
    parque.nombre AS Parque,
    SUM(consumo.importe) AS facturacionTotal
FROM
    Consumo consumo,
    Producto producto,
```

```
    Atraccion atraccion,
    Factura factura,
    Parque parque
WHERE
    producto.idProducto = atraccion.idProducto
        AND parque.idParque = atraccion.idParque
        AND producto.idProducto = consumo.idProducto
        AND factura.idTarjeta = consumo.idTarjeta
        AND factura.fechaEmitida > consumo.fechaYhora
        AND MONTH(consumo.fechaYhora) > (MONTH(factura.fechaEmitida) - 1)
        AND YEAR(consumo.fechaYhora) = YEAR(consumo.fechaYhora)
GROUP BY consumo.idProducto
HAVING SUM(consumo.importe) >= ALL (SELECT
        SUM(consumo1.importe)
    FROM
        Consumo consumo1,
        Atraccion atraccion1,
        Producto producto1,
        Factura factura1
    WHERE
        consumo1.idProducto != consumo.idProducto
            AND producto1.idProducto = atraccion1.idProducto
            AND consumo1.idProducto = producto1.idProducto
            AND consumo1.idTarjeta = factura1.idTarjeta
            AND factura1.fechaEmitida > consumo1.fechaYhora
            AND MONTH(consumo1.fechaYhora) > (MONTH(factura1.fechaEmitida) - 1)
            AND YEAR(consumo1.fechaYhora) = YEAR(consumo1.fechaYhora)
    GROUP BY consumo1.idProducto);
```

1.4.3. Estadísticas: atracción que más facturó por parque.

```
-- Atracción con más facturación por parque
SELECT
    parque.nombre AS nombreParque,
    producto.nombre AS nombreAtraccion
FROM
    mydb.Consumo consumo,
    mydb.Factura factura,
    mydb.Producto producto,
    mydb.Atraccion atraccion,
    mydb.Parque parque
WHERE
    producto.idProducto = atraccion.idProducto
        AND parque.idParque = atraccion.idParque
        AND producto.idProducto = consumo.idProducto
        AND factura.idTarjeta = consumo.idTarjeta
        AND factura.fechaEmitida > consumo.fechaYhora
        AND MONTH(consumo.fechaYhora) > (MONTH(factura.fechaEmitida) - 1)
        AND YEAR(consumo.fechaYhora) = YEAR(consumo.fechaYhora)
        AND producto.idProducto IN
        (SELECT
            producto1.idProducto AS Atraccion
```

```
        FROM
            mydb.Consumo consumo1,
            mydb.Factura factura1,
            mydb.Producto producto1,
            mydb.Atraccion atraccion1,
            mydb.Parque parque1
        WHERE
            producto1.idProducto = atraccion1.idProducto
                AND parque1.idParque = atraccion1.idParque
                AND producto1.idProducto = consumo1.idProducto
                AND factura1.idTarjeta = consumo1.idTarjeta
                AND factura1.fechaEmitida > consumo1.fechaYhora
                AND MONTH(consumo1.fechaYhora) > (MONTH(factura1.fechaEmitida) - 1)
                AND YEAR(consumo1.fechaYhora) = YEAR(consumo1.fechaYhora)
                AND parque1.idParque = parque.idParque
        GROUP BY parque1.nombre , consumo1.idProducto
        HAVING SUM(consumo1.importe) >= ALL (SELECT
                SUM(consumo.importe) AS facturacionTotal
            FROM
                mydb.Consumo consumo,
                mydb.Factura factura,
                mydb.Producto producto,
                mydb.Atraccion atraccion,
                mydb.Parque parque2
            WHERE
                producto.idProducto = atraccion.idProducto
                    AND parque.idParque = atraccion.idParque
                    AND producto.idProducto = consumo.idProducto
                    AND factura.idTarjeta = consumo.idTarjeta
                    AND factura.fechaEmitida > consumo.fechaYhora
                    AND MONTH(consumo.fechaYhora) > (MONTH(factura.fechaEmitida) - 1)
                    AND YEAR(consumo.fechaYhora) = YEAR(consumo.fechaYhora)
                    AND parque2.idParque = parque.idParque
            GROUP BY parque.nombre , consumo.idProducto))
        GROUP BY parque.idParque, producto.idProducto
```

1.4.4. Listado de facturas impagas

```
SELECT numero as NumeroDeFactura , fechaVencimiento
FROM mydb.Factura, mydb.Recibo
WHERE mydb.Factura.fechaVencimiento <= now()
      AND mydb.Recibo.idFactura != mydb.Factura.idFactura;
```

1.4.5. Para cada cliente las atracciones más visitadas en rango de fechas

```
delimiter $
create procedure atraccionMasVisitadaPorCliente(in fechaDesde datetime, in fechaHasta datetime)
begin
SELECT
    cliente.nombre AS nombreCliente,
```

```
    cliente.apellido AS apellidoCliente,
    producto.nombre AS nombreAtraccion
FROM
    mydb.Consumo consumo,
    mydb.Tarjeta tarjeta,
    mydb.Producto producto,
    mydb.Cliente cliente
WHERE
    producto.tipoProducto LIKE 'atraccion'
        AND cliente.idCliente = tarjeta.idCliente
        AND tarjeta.idTarjeta = consumo.idTarjeta
        AND producto.idProducto = consumo.idProducto
        AND producto.idProducto IN (
        SELECT
            producto1.idProducto AS AtraccionesDeMaximasVisitasXCliente
        FROM
            mydb.Consumo consumo1,
            mydb.Tarjeta tarjeta1,
            mydb.Producto producto1,
            mydb.Cliente cliente1
        WHERE
            producto1.tipoProducto LIKE 'atraccion'
                AND cliente1.idCliente = tarjeta1.idCliente
                AND tarjeta1.idTarjeta = consumo1.idTarjeta
                AND producto1.idProducto = consumo1.idProducto
                AND consumo1.fechaYhora >= fechaDesde
                AND consumo1.fechaYhora <= fechaHasta
             AND cliente1.idCliente = cliente.idCliente
        GROUP BY cliente1.nombre , consumo1.idProducto
        HAVING COUNT(consumo1.idProducto) >= ALL (SELECT
                COUNT(consumo2.idProducto) AS cantidadVisitasAtraccionXCliente
            FROM
                mydb.Consumo consumo2,
                mydb.Tarjeta tarjeta2,
                mydb.Producto producto2,
                mydb.Cliente cliente2
            WHERE
                producto2.tipoProducto LIKE 'atraccion'
                    AND cliente2.idCliente = tarjeta2.idCliente
                    AND tarjeta2.idTarjeta = consumo2.idTarjeta
                    AND producto2.idProducto = consumo2.idProducto
                    AND cliente2.idCliente = cliente.idCliente
            GROUP BY cliente2.idCliente , consumo2.idProducto)


            ) GROUP BY cliente.idCliente, producto.idProducto;

            end
$
```

1.4.6. Cambios de categorías de cliente en rango de fechas

```
delimiter $
create procedure CambioCategoriaClientePorFechas(in fechaInicio datetime, in fechaFin datetime)
begin
SELECT cliente.nombre as Nombre, cliente.apellido as Apellido,
pertenece.nombreCategoria as Categoria, pertenece.fechaDesde as Fecha
FROM mydb.Cliente cliente, mydb.perteneceA pertenece, mydb.Tarjeta tarjeta
WHERE cliente.idCliente = tarjeta.idCliente and
     pertenece.fechaDesde >= fechaInicio and
     pertenece.fechaDesde <= fechaFin
;
end
```

1.4.7. Atracciones con descuento para cada categoría

```
SELECT
    poseeDesc.nombreCategoria AS Categoria,
    producto.nombre AS Atraccion,
    poseeDesc.descuento AS descuento
FROM
    mydb.PoseeDescuento poseeDesc,
    mydb.Producto producto
WHERE
    producto.idProducto = poseeDesc.idProducto
        AND producto.tipoProducto LIKE 'atraccion';
```

1.4.8. Empresa organizadora de eventos que tuvo mayor facturación.

```
SELECT
    empresa.razonSocial,
    SUM(consumo.importe) AS facturacionTotal
FROM
    Empresa empresa,
    Consumo consumo,
    mydb.Evento evento,
    mydb.Factura factura
WHERE
    evento.idProducto = consumo.idProducto
        AND evento.idEmpresa = empresa.idEmpresa
        AND factura.idTarjeta = consumo.idTarjeta
        AND factura.fechaEmitida > consumo.fechaYhora
        AND MONTH(consumo.fechaYhora) > (MONTH(factura.fechaEmitida) - 1)
        AND YEAR(consumo.fechaYhora) = YEAR(consumo.fechaYhora)
GROUP BY empresa.razonSocial
HAVING SUM(consumo.importe) >= ALL (SELECT
        SUM(consumo1.importe)
    FROM
        Empresa empresa1,
        Consumo consumo1,
```

```
        Evento evento1,
        Factura factura1
    WHERE
        evento1.idProducto = consumo1.idProducto
            AND evento1.idEmpresa = empresa1.idEmpresa
            AND factura1.idTarjeta = consumo1.idTarjeta
            AND factura1.fechaEmitida > consumo1.fechaYhora
            AND MONTH(consumo1.fechaYhora) > (MONTH(factura1.fechaEmitida) - 1)
            AND YEAR(consumo1.fechaYhora) = YEAR(consumo1.fechaYhora)
    GROUP BY empresa1.razonSocial)
```

1.4.9. Procedimiento que verifica las categorías y realiza el cambio de ser necesario.

```
CREATE PROCEDURE `cambiarAllCategorias` ()
BEGIN
        DECLARE v_finished INTEGER DEFAULT 0;
    DECLARE v_idTarjeta INTEGER DEFAULT 0;
    DECLARE v_conPromMensual INT DEFAULT 0;
    DECLARE v_conAcumAnual INT DEFAULT 0;
    DECLARE v_catCorresp_nombre VARCHAR(50) DEFAULT NULL;
    DECLARE v_catActual_nombre VARCHAR(50) DEFAULT NULL;
    DECLARE v_catActual_x INT DEFAULT 0;
    DECLARE v_catActual_y INT DEFAULT 0;
    DECLARE v_catCorresp_x INT DEFAULT 0;
    DECLARE v_allIdsTarjetas CURSOR FOR SELECT idTarjeta FROM Tarjeta WHERE activada = 1;
    -- De una sola query calculo el promedio mensual en el año actual y el gasto acumulado

        -- declare NOT FOUND handler
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_finished = 1;

        OPEN v_allIdsTarjetas;

        for_each_idTarjeta: LOOP

        FETCH v_allIdsTarjetas INTO v_idTarjeta;
        IF v_finished = 1 THEN         LEAVE for_each_idTarjeta; END IF;

        -- Obtengo el consumo promedio por mes y el total gastado en el anio
        SELECT SUM(importe), AVG(importe) INTO v_conAcumAnual, v_conPromMensual
        FROM  Consumo c
        WHERE c.idTarjeta = v_idTarjeta AND YEAR(c.fechaYhora) = YEAR(CURDATE())
        GROUP BY YEAR(fechaYhora) DESC;

        -- La categoria actual es aquella que en la relacion perteneceA
        -- posee la fecha mas reciente
        SELECT  pa1.nombreCategoria, x, y INTO v_catActual_nombre , v_catActual_x , v_catActual_y
        FROM  PerteneceA pa1, Categoria c
        WHERE pa1.nombreCategoria = c.nombreCategoria
                AND pa1.idTarjeta = v_idTarjeta
                AND pa1.fechaDesde = (SELECT MAX(pa2.fechaDesde)
                                      FROM PerteneceA pa2
                                      WHERE pa2.idTarjeta = v_idTarjeta);
```

```
        -- La categoria correspondiente va a ser
        -- aquella a la que se le haya superado el limite x
        SELECT  nombreCategoria, x
        INTO v_catCorresp_nombre , v_catCorresp_x
        FROM  Categoria
        WHERE x = (SELECT MAX(x) FROM Categoria c  WHERE c.x <= v_conAcumAnual);

        -- Si el consumo mensual es adecuado para la categoria actual,
        -- y la categoria correspondiente no es mayor a la actual,
        -- entonces la categoria correspondiente es la actual
        -- En cualquier otro caso, la categoria se resuelve por el parametro x

        IF NOT (v_catActual_y <= v_conPromMensual and v_catCorresp_x <= v_catActual_x )THEN
                INSERT INTO PerteneceA VALUES (v_idTarjeta,v_catCorresp_nombre, CURDATE());
        END IF;

    END LOOP for_each_idTarjeta;

        CLOSE v_allIdsTarjetas;
END
```

1.4.10. Ranking de parques/atracciones con mayor cantidad de visitas en rango de fechas.

```
delimiter $
CREATE PROCEDURE atraccionMayorVisita(IN fechaDesde datetime, IN fechaHasta datetime)
BEGIN
SELECT producto.nombre as ParqueAtraccion, count(*) as cantidadVisitado
FROM Atraccion atraccion, Tarjeta tarjeta, Producto producto, Consumo consumo
WHERE atraccion.idProducto = producto.idProducto
    AND consumo.fechaYhora >= fechaDesde
    AND consumo.fechaYhora <= fechaHasta
    AND producto.idProducto = consumo.idProducto
GROUP BY consumo.idProducto
HAVING count(*) >= ALL (SELECT count(*) as count
       FROM mydb.Atraccion atraccion, mydb.Tarjeta tarjeta, Producto producto, Consumo consumo
       WHERE atraccion.idProducto = producto.idProducto
         AND consumo.fechaYhora >= fechaDesde
         AND consumo.fechaYhora <= fechaHasta
         AND producto.idProducto = consumo.idProducto
       GROUP BY consumo.idProducto)
;
END
```

1.5. Triggers

Para cumplir con algunas restricciones del modelo propuesto implementamos los siguientes triggers

1.5.1. Atracción

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Atraccion_BEFORE_INSERT` BEFORE INSERT ON `Atraccio
BEGIN

if (new.edadDesde < 0 or ( new.edadDesde > new.edadHasta)) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo edadDesde debe ser menor o igual a edadHa
end if;

END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Atraccion_BEFORE_UPDATE` BEFORE UPDATE ON `Atraccio
BEGIN

if (new.edadDesde < 0 or ( new.edadDesde > new.edadHasta)) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo edadDesde debe ser menor o igual a edadHa
end if;

END
```

1.5.2. Categoria

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Categoria_BEFORE_INSERT` BEFORE INSERT ON `Categori
BEGIN

if (new.x < 0 or new.y < 0) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Los atributo x e y deben ser mayores o igual a 0';
end if;

END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Categoria_BEFORE_UPDATE` BEFORE UPDATE ON `Categori
BEGIN

if (new.x < 0 or new.y < 0) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Los atributo x e y deben ser mayores o igual a 0';
end if;
END
```

1.5.3. Cliente

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Cliente_BEFORE_INSERT` BEFORE INSERT ON `Cliente` F
BEGIN
```

```
if !(new.foto REGEXP "^http:\/\/[.A-Za-z0-9-]+.[a-zA-Z]\/") then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo foto debe ser una url a una fotografia';
end if;

END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Cliente_BEFORE_UPDATE` BEFORE UPDATE ON `Cliente` F
BEGIN

if !(new.foto REGEXP "^http:\/\/[.A-Za-z0-9-]+.[a-zA-Z]\/") then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo foto debe ser una url a una fotografia';
end if;

END
```

1.5.4. Empresa

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Empresa_BEFORE_INSERT` BEFORE INSERT ON `Empresa` F
BEGIN

if !(new.cuit REGEXP "^[0-9]{10}$") then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo cuit debe estar formado por 10 digitos nu
end if;

END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Empresa_BEFORE_UPDATE` BEFORE UPDATE ON `Empresa` F
BEGIN

if !(new.cuit REGEXP "^[0-9]{10}$") then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo cuit debe estar formado por 10 digitos nu
end if;

END
```

1.5.5. Evento

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Evento_BEFORE_INSERT` BEFORE INSERT ON `Evento` FOR
BEGIN
if (new.fechaDesde > new.fechaHasta) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo fechaDesde debe ser menor o igual a fecha
end if;
END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Evento_BEFORE_UPDATE` BEFORE UPDATE ON `Evento` FOR
BEGIN
if (new.fechaDesde > new.fechaHasta) then
```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo fechaDesde debe ser menor o igual a fecha
end if;
END
```

1.5.6. Factura

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Factura_BEFORE_INSERT` BEFORE INSERT ON `Factura` F
BEGIN

if !(new.fechaVencimiento >= new.fechaEmitida) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo fechaVencimiento debe ser mayor a fechaEm
end if;

END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Factura_BEFORE_UPDATE` BEFORE UPDATE ON `Factura` F
BEGIN

if !(new.fechaVencimiento >= new.fechaEmitida) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo fechaVencimiento debe ser mayor a fechaEm
end if;

END
```

1.5.7. PoseeDescuento

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`PoseeDescuento_BEFORE_INSERT` BEFORE INSERT ON `Pos
BEGIN
if (new.descuento <= 0) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo descuento debe ser mayor a 0';
end if;
END

\begin{minted}{sql}
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`PoseeDescuento_BEFORE_UPDATE` BEFORE UPDATE ON `Pos
BEGIN
if (new.descuento <= 0) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo descuento debe ser mayor a 0';
end if;
END
```

1.5.8. Precio

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Precio_BEFORE_INSERT` BEFORE INSERT ON `Precio` FOR
BEGIN
if (new.valor < 0 or (new.diaSemana != 'L' AND
new.diaSemana != 'M' AND
new.diaSemana != 'X' AND
```

```
new.diaSemana != 'J' AND
new.diaSemana != 'V' AND
new.diaSemana != 'S' AND
new.diaSemana != 'D')) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo VALOR debe ser mayor a 0 y diaSemana debe
pertencer a {L,M,X,J,V,S,D}';
end if;
END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Precio_BEFORE_UPDATE` BEFORE UPDATE ON `Precio` FOR
BEGIN
if (new.valor < 0 or (new.diaSemana != 'L' AND
new.diaSemana != 'M' AND
new.diaSemana != 'X' AND
new.diaSemana != 'J' AND
new.diaSemana != 'V' AND
new.diaSemana != 'S' AND
new.diaSemana != 'D')) then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El atributo VALOR debe ser mayor a 0 y diaSemana debe
pertencer a {L,M,X,J,V,S,D}';
end if;
END
```

1.5.9. Producto

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Producto_BEFORE_INSERT` BEFORE INSERT ON `Producto`
BEGIN

if !(new.tipoProducto = 'evento' or new.tipoProducto = 'atraccion') then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El tipo de producto debe ser atraccion o evento';
end if;

END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Producto_BEFORE_UPDATE` BEFORE UPDATE ON `Producto`
BEGIN

if !(new.tipoProducto = 'evento' or new.tipoProducto = 'atraccion') then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El tipo de producto debe ser atraccion o evento';
end if;

END


CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Producto_AFTER_INSERT` AFTER INSERT ON `Producto` F
BEGIN
INSERT INTO `Precio` (`idProducto`, `diaSemana`, `valor`) VALUES (new.idProducto, 'D', 9999);
INSERT INTO `Precio` (`idProducto`, `diaSemana`, `valor`) VALUES (new.idProducto, 'L', 9999);
INSERT INTO `Precio` (`idProducto`, `diaSemana`, `valor`) VALUES (new.idProducto, 'M', 9999);
INSERT INTO `Precio` (`idProducto`, `diaSemana`, `valor`) VALUES (new.idProducto, 'X', 9999);
INSERT INTO `Precio` (`idProducto`, `diaSemana`, `valor`) VALUES (new.idProducto, 'J', 9999);
```

```
INSERT INTO `Precio` (`idProducto`, `diaSemana`, `valor`) VALUES (new.idProducto, 'V', 9999);
INSERT INTO `Precio` (`idProducto`, `diaSemana`, `valor`) VALUES (new.idProducto, 'S', 9999);
END
```

2. Conclusiones

A lo largo del diseño e implementación del modelo de datos nos encontramos con estas etapas como las mas significativas que nos genero contratiempos.

- Modelización de entidades
- Refinamientos en el MER y MR
- Implementación de queries

Modelización de entidades

A la hora de realizar el modelo entidad relación, de forma inicial se planteo la existencia de una entidad débil ItemDeFactura que se relacionaría de la forma muchos a uno con la entidad Factura (N a 1) dado un consumo determinado. Esto lo que provocaba era redundancia de datos de consumo sin necesidad. Se observó que como solo se requería el grupo de consumos que agrupaba una factura (y no por ejemplo la posición dentro de la factura de cada consumo), la entidad no tenia sentido.

Refinamientos en el MER y MR

El pasaje del MER al MR fue sistemático, pero a medida que se conocían requerimientos adicionales del problema se produjeron varios refinamientos sobre el MER de forma iterativa, lo que subsecuentemente también ocasiono cambios en el MR. En principio consideramos que Tarjeta solo pertenecía a una categoría pero debido a las consultas que debían implementarse nos dimos cuenta que se necesita información histórica acerca de las categorías a las que perteneció. También nos encontramos con ciertas restricciones que se habían pasado por alto a la hora de diseñar.

Implementación de queries

Las queries relacionadas con la obtención de los máximos representaron las de mayor dificultad a la hora de programar. Errores de datos duplicados por errores en los GROUP BY o la falta de HAVING en la query fueron algunos de nuestros errores más comunes. Otro tipo de error a la hora de programar en SQL fueron los nombres de las variables y renombres de tablas: las variables de las store procedures se denotan con ”v_.ªl inicio por ejemplo, para no confundir al interprete con las columnas propias de las tablas.

3. Aclaraciones para correr las implementaciones

Para montar el proyecto se deberá tener instalado el motor de base de datos Mysql Server 5.7 Server compatibles con MySQL en su versión más nueva, trabajamos con la interfaz MySQL Workbench versión 6.3.

Para crear la base de datos se deberá correr **tp1.sql** el cual contiene el DDL. Tanto las queries de los distintos ejercicios como los store procedures se encuentran en la carpeta ”queries”. En el caso de las store procedures, los archivos contienen el código para eliminarlas y crearlas nuevamente a cada una. Se generaron inserts de prueba para el testeo de la base: los mismos se encuentran en el archivo **mockDataInserts.sql** en la carpeta raiz del proyecto.