Universidad Nacional de Luján
Departamento de Ciencias Básicas
Programación Aplicada

UTILIZANDO SQL CON FIREBIRD / INTERBASE

Guillermo R. Cherencio
gcherencio@mail.unlu.edu.ar
"del modelo conceptual a la implementación"

El presente documento pretende dar una serie de ejemplos de la utilización de comandos sql utilizando el SGBD FireBird / Interbase de Borland / Inprise, de distribución gratuita (bajo licencia GNU) y que puede ser bajado de http://firebird.sourceforge.net/. También puede consultar en www.Interbase.com . Veamos el siguiente problema:

Mundo Real

Un ferretería desea informatizar la emisión de facturas a clientes, las cuales tienen el siguiente formato:

Fecha:XXXXXXXX **Cliente**:XXXXXXXXXXXXXXXXXXXXXXX **Factura Nº:** XXXX - XXXXXXX
Vendedor:XXXXXXXXXXXXXXXXX **Categoría IVA**: XXXXXXXXXXXX

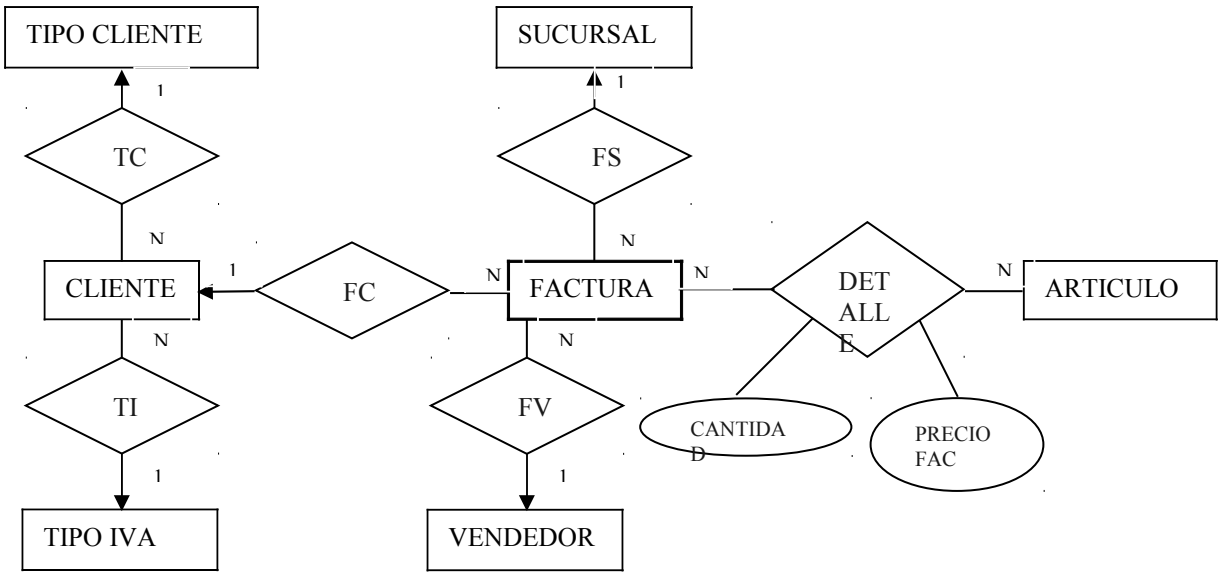
Articulo	Descripcion	Cantidad	Precio	Subtotal
1000012	Motosierra	1	350.50	350.50
2001225	Guadaña	2	50.00	100.00
.....				
.....				
			Total Neto	**450.50**
			IVA XXX %	**XXX.XX**
			Total General	**XXX.XX**

Las ventas se hacen en mostrador a cargo de un vendedor que atiende al cliente, las facturas tienen como identificador al codigo de sucursal más el número de factura (XXXXX - XXXXXXX). Existen 3 categorias de cliente (minorista, mayorista y consumidor final) que poseen distintos % de descuento sobre el precio de los productos. Cada cliente tiene una categoría de iva, dicha categoria de iva implica un % de impuesto determinado al total neto de la factura. Todos los articulos son gravados por el IVA. Responda las siguientes consultas sobre este modelo:

1) ¿Cuál es el vendedor que vendió más (hablando en $)?
2) ¿Cuál fue el cliente que más nos compró (hablando en $)?
3) ¿Cuál fue el total de ventas (hablando en $) por categoría de cliente?

Diseño Conceptual

Un Diseño Conceptual posible y válido (digo posible porque hay varias manera de hacerlo, no todas válidas, por supuesto):



También se podría haber pensando en una generalización (también llamado tipo – subtipo) en la entidad cliente para representar a los clientes minoristas, mayoristas y consumidor final. Todas las relaciones son obligatorias, los atributos que faltan indicar en el D.E.R. son los siguientes:

Entidad	**Atributo**	**A.I.P.**	**Obligatorio**	**Descripción**
Cliente	Idcli	X	X	Identificador de cliente
Cliente	Nombre		X	Nombre/Razón Social cliente
Cliente	Dirección		X	Dirección (calle,nro,piso,etc) cliente
Cliente	TE			Teléfono cliente
Tipo Iva	Idiva	X	X	Identificador de tipo iva
Tipo Iva	Porc		X	Porcentaje de iva sobre neto
Tipo Cliente	Idtipo	X	X	Identificador de tipo iva
Tipo Cliente	porc_dto		X	Porcentaje de dto sobre precio
Sucursal	Idsuc	X	X	Identificador de sucursal
Sucursal	Nombre		X	Descripción sucursal
Vendedor	Idven	X	X	Identificador de vendedor
Vendedor	Nombre		X	Nombre del vendedor
Factura	Idsuc	X	X	Identificador de sucursal
Factura	Nro	X	X	Numero de factura
Factura	Fecha		X	Fecha de factura
Factura	Neto		X	Neto de factura (calculado)
Factura	Porc		X	% de iva (calculado)
Factura	Iva		X	$ de iva (calculado)
Factura	Total		X	Total de factura (calculado)
Articulo	Idart	X	X	Identificador Articulo
Articulo	Descripcion		X	Descripcion Articulo
Articulo	Precio_base		X	Precio base sin dto del Articulo

La entidad factura es débil y el nro de factura actúa como discriminador, factura hereda el código de sucursal de la entidad fuerte de la cual depende que en este caso es sucursal. La relación DETALLE que representa los articulos facturados, su atributo precio fac se refiere al precio facturado en el momento que se realizó la factura, para evitar el hecho de que, al producirse cambios de precios en los articulos, si éstos habian sido previamente facturados, provocaría el recalculo y variación de las facturas ya emitidas y entregadas a clientes.

Diseño Logico

El Modelo Lógico sería el siguiente: (sale automáticamente del diseño conceptual)

SUCURSAL(**IDSUC**,NOMBRE)
TIPOIVA(**IDIVA**,PORC)
TIPOCLIENTE(**IDTIPO**,PORC_DTO)
CLIENTE(**IDCLI**,NOMBRE,DIRECCION,TE null,IDIVA,IDTIPO)
IDIVA FK TIPOIVA
IDTIPO FK TIPOCLIENTE
VENDEDOR(**IDVEN**,NOMBRE)
ARTICULO(**IDART**,DESCRIPCION,PRECIO_BASE)
FACTURA(**IDSUC,NRO**,FECHA,IDVEN,IDCLI)
IDSUC FK SUCURSAL
IDVEN FK VENDEDOR
IDCLI FK CLIENTE
DETALLE(**IDSUC,NRO,ARTICULO**,CANTIDAD,PRECIO_FAC)
IDSUC,NRO FK FACTURA
ARTICULO FK ARTICULO

Normalización

Aquí hemos omitido los atributos que consideramos calculados en el diseño conceptual. Podemos someter este Modelo Lógico al proceso de normalización y comprobaremos que el mismo se encuentra normalizado.

Diseño Fisico

Diseño Físico utilizando FireBird DBMS:

Existen dependencias de existencia entre los distintos objetos que debemos crear en la base de datos, por ejemplo: no podemos crear "cliente" antes que "tipoiva". Esta situación debemos tenerla muy cuenta, tanto para destruir los objetos previamente creados como para crear nuevos. Una vez instalada la base de datos FireBird, el usuario con mayor privilegio es “SYSDBA” y su password por defecto es “masterkey“ (esto puede y debe ser cambiado por motivos de seguridad, nosotros utilizaremos este usuario y password para este apunte, pero Ud. bien podría usar otro); teniendo en ejecución el servidor de base de datos, a partir del subdirectorio \bin (del path en donde instalamos FireBird) podemos ejecutar el siguiente comando en la linea de comandos:

```
isql  -u SYSDBA –p masterkey
```

y luego ejecutar el siguiente comando SQL para crear nuestra base de datos en el directorio actual:

```
SQL>CREATE DATABASE ‘ferreteria.gdb’;
```

por último, podemos salir de isql para luego ejecutar un script que permita crear todos los objetos necesarios dentro de la base de datos y poder poblarla con datos:

```
SQL>QUIT;
```

para ejecutar el script "ferreteria.sql" (del cual se adjunta la copia a continuación y en recuadro) que permitirá la creación de los objetos de la base de datos, podemos ejecutar el siguiente comando desde la linea de comandos:

```
isql  ferreteria.gdb  –u SYSDBA  –p masterkey  -i ferreteria.sql –o salida.txt
```

luego de esta ejecución, tendremos en salida.txt la salida de los comandos sql ejecutados, para salir de isql, podemos tipear el comando QUIT:

```
SQL>QUIT;
```

```
/*
#
# Script demo Implementacion con FireBird DBMS
#
# Programacion Aplicada
# Universidad Nacional de Lujan
#
# Lic. Guillermo Cherencio
#
# Modo de Ejecucion:       isql ferreteria.gdb -u SYSDBA -p masterkey -o salida.txt
# Previamente se debe crear la base de datos ferreteria.gdb
# Se utiliza el usuario SYSDBA password masterkey, usuario por defecto en FireBird
# sino cambiar por el que corresponda.
# Este script puede re-ejecutarse
# Primero borro todos los objetos que pudieran quedar de ejecuciones anteriores
# Luego creo los objetos necesarios lograr la implementacion de esta base de datos
*/
SET ECHO ON;

/* Borro Triggers */
DROP TRIGGER trg_bidetalle;
DROP TRIGGER trg_budetalle;
DROP TRIGGER trg_bufactura;
DROP TRIGGER trg_budetalle2;
DROP TRIGGER trg_bdfactura;
/* Borro Procedures */
DROP PROCEDURE sp_precio_fac_ok;
DROP PROCEDURE sp_mov_fac;
DROP PROCEDURE sp_mov_fac;
/* Borro Exceptions */
DROP EXCEPTION ex_precio_fac_inc;
DROP EXCEPTION ex_suc_nro_inc;
/* Borro Views */
DROP VIEW tot_cli_max;
DROP VIEW tot_cli;
DROP VIEW tot_ven_max;
DROP VIEW tot_ven;
DROP VIEW vfactura;
```

```
DROP VIEW vfactura1;
DROP VIEW vdetalle;
COMMIT;
/* Borro Tablas */
DROP TABLE detalle;
DROP TABLE factura;
DROP TABLE articulo;
DROP TABLE vendedor;
DROP TABLE cliente;
DROP TABLE tipocliente;
DROP TABLE tipoiva;
DROP TABLE sucursal;
DROP TABLE login;
DROP TABLE detalle_his;
DROP TABLE factura_his;
/* Borro Generators */
DROP GENERATOR LOG_ID;
COMMIT;

/* Creo Tablas */
CREATE TABLE sucursal(idsuc NUMERIC(3,0) DEFAULT 1 NOT NULL,
        nombre VARCHAR(40) NOT NULL,
        PRIMARY KEY (idsuc));

ALTER TABLE sucursal ADD CONSTRAINT suc_idsuc  CHECK ( idsuc > 0 );

COMMIT;

SHOW TABLE sucursal;

CREATE TABLE tipoiva(idiva NUMERIC(1,0) DEFAULT 1 NOT NULL,
        porc NUMERIC(5,2) NOT NULL,
        PRIMARY KEY (idiva));

ALTER TABLE tipoiva ADD CONSTRAINT ti_idiva  CHECK ( idiva > 0 );

ALTER TABLE tipoiva ADD CONSTRAINT ti_porc  CHECK ( porc > 0 );

COMMIT;

SHOW TABLE tipoiva;

CREATE TABLE tipocliente(idtipo NUMERIC(1,0) DEFAULT 1 NOT NULL,
        porc NUMERIC(5,2) NOT NULL,
        PRIMARY KEY (idtipo));

ALTER TABLE tipocliente ADD CONSTRAINT tc_idtipo  CHECK ( idtipo > 0 );

ALTER TABLE tipocliente ADD CONSTRAINT tc_porc  CHECK ( porc > 0 );

COMMIT;

SHOW TABLE tipocliente;

CREATE TABLE cliente(idcli NUMERIC(5,0) DEFAULT 1 NOT NULL,
        nombre VARCHAR(40) NOT NULL,
        direccion VARCHAR(40) NOT NULL,
        te VARCHAR(40),
```

```
            idiva NUMERIC(1,0) NOT NULL,
            idtipo NUMERIC(1,0) NOT NULL,
            PRIMARY KEY (idcli));

ALTER TABLE cliente ADD CONSTRAINT cli_idcli  CHECK ( idcli > 0 );

ALTER TABLE cliente ADD CONSTRAINT cli_idiva  CHECK ( idiva > 0 );

ALTER TABLE cliente ADD CONSTRAINT cli_idtipo CHECK ( idtipo > 0 );

ALTER TABLE cliente ADD CONSTRAINT fk_cli_tipoiva FOREIGN KEY (idiva) REFERENCES tipoiva;

ALTER TABLE cliente ADD CONSTRAINT fk_cli_tipocliente FOREIGN KEY (idtipo) REFERENCES tipocliente;

COMMIT;

SHOW TABLE cliente;

CREATE TABLE vendedor(idven NUMERIC(3,0) DEFAULT 1 NOT NULL,
            nombre VARCHAR(40) NOT NULL,
            PRIMARY KEY (idven));

ALTER TABLE vendedor ADD CONSTRAINT ven_idven  CHECK ( idven > 0 );

COMMIT;

SHOW TABLE vendedor;

CREATE TABLE articulo(idart NUMERIC(5,0) DEFAULT 1 NOT NULL,
            descripcion VARCHAR(40) NOT NULL,
            precio_base NUMERIC(5,2) NOT NULL,
            PRIMARY KEY (idart));

ALTER TABLE articulo ADD CONSTRAINT art_idart  CHECK ( idart > 0 );

ALTER TABLE articulo ADD CONSTRAINT art_precio  CHECK ( precio_base > 0 );

COMMIT;

SHOW TABLE articulo;

CREATE TABLE factura(idsuc NUMERIC(3,0) DEFAULT 1 NOT NULL,
            nro NUMERIC(8,0) DEFAULT 1 NOT NULL,
            fecha DATE NOT NULL,
            idven NUMERIC(3,0) DEFAULT 1 NOT NULL,
            idcli NUMERIC(5,0) DEFAULT 1 NOT NULL,
            PRIMARY KEY (idsuc,nro));

ALTER TABLE factura ADD CONSTRAINT fac_idsuc  CHECK ( idsuc > 0 );

ALTER TABLE factura ADD CONSTRAINT fac_nro CHECK ( nro > 0 );

ALTER TABLE factura ADD CONSTRAINT fac_idven  CHECK ( idven > 0 );

ALTER TABLE factura ADD CONSTRAINT fac_idcli  CHECK ( idcli > 0 );

ALTER TABLE factura ADD CONSTRAINT fk_fac_ven FOREIGN KEY (idven) REFERENCES vendedor;
```

```
ALTER TABLE factura ADD CONSTRAINT fk_fac_suc FOREIGN KEY (idsuc) REFERENCES sucursal;

ALTER TABLE factura ADD CONSTRAINT fk_fac_cli FOREIGN KEY (idcli) REFERENCES cliente;

COMMIT;

SHOW TABLE factura;

CREATE TABLE detalle(idsuc NUMERIC(3,0) DEFAULT 1 NOT NULL,
            nro NUMERIC(8,0) DEFAULT 1 NOT NULL,
            idart NUMERIC(5,0) DEFAULT 1 NOT NULL,
            cantidad NUMERIC(3,0) DEFAULT 1 NOT NULL,
            precio_fac NUMERIC(5,2) NOT NULL,
            PRIMARY KEY (idsuc,nro,idart));

ALTER TABLE detalle ADD CONSTRAINT det_idsuc  CHECK ( idsuc > 0 );

ALTER TABLE detalle ADD CONSTRAINT det_nro  CHECK ( nro > 0 );

ALTER TABLE detalle ADD CONSTRAINT det_idart  CHECK ( idart > 0 );

ALTER TABLE detalle ADD CONSTRAINT det_cantidad  CHECK ( cantidad > 0 );

ALTER TABLE detalle ADD CONSTRAINT det_precio  CHECK ( precio_fac > 0 );

ALTER TABLE detalle ADD CONSTRAINT fk_det_fac FOREIGN KEY (idsuc,nro) REFERENCES factura;

ALTER TABLE detalle ADD CONSTRAINT fk_det_art FOREIGN KEY (idart) REFERENCES articulo;

COMMIT;

SHOW TABLE detalle;
```

Implementamos los atributos calculados:

Calculo subtotal en detalle de factura

```
/*
Implementamos los atributos calculados utilizando Views:
Calculo subtotal en detalle de factura
*/
CREATE VIEW vdetalle (idsuc,nro,idart,cantidad,precio_fac,subtotal) AS
SELECT  idsuc,
            nro,
            idart,
            cantidad,
            precio_fac,
            (cantidad * precio_fac)
FROM detalle;

COMMIT;

SHOW VIEW vdetalle;
```

Calculo el neto de la factura

```
/*
Calculo el neto de la factura
*/
CREATE VIEW vfactura1 (nro,idsuc,fecha,idven,idcli,neto) AS
SELECT  f.nro,
        f.Idsuc,
        f.fecha,
        f.idven,
        f.idcli,
        sum(d.subtotal)
FROM    factura f, vdetalle d
WHERE  f.nro = d.nro and f.idsuc = d.idsuc
GROUP BY f.nro,f.idsuc,f.fecha,f.idven,f.idcli;

COMMIT;

SHOW VIEW vfactura1;
```

Calculo $ iva, % iva, total de factura

```
/*
Calculo $ iva, % iva, total de factura
*/
CREATE VIEW vfactura (nro,idsuc,fecha,idven,idcli,neto,porc,iva,total) AS
SELECT  f.nro,
        f.Idsuc,
        f.fecha,
        f.idven,
        f.idcli,
        f.neto,
        ti.porc,
        (f.neto * (ti.porc / 100)),
        (f.neto + (f.neto * (ti.porc / 100)))
FROM    vfactura1 f, cliente c, tipoiva ti
WHERE  f.idcli = c.idcli and c.idiva = ti.idiva;

COMMIT;

SHOW VIEW vfactura;
```

Ingresamos datos de prueba:

Teniendo en cuenta la dependencia de objetos y el concepto de integridad (ya sea tanto referencial, de dominio o de entidad) podemos hacer el ingreso de datos para "poblar" esta base de datos. También es muy recomendable hacer un ingreso de datos con el objetivo de tratar de violar la integridad de la misma para verificar que todos los ojetos fueron creados con las restricciones y controles necesarios para mantener íntegra a la base de datos a pesar de los intentos por ingresar datos incorrectos o inconsistentes.

```
/*
Ingresamos datos de prueba:
        Teniendo en cuenta la dependencia de objetos y el concepto de integridad (ya sea tanto referencial,
de dominio o de entidad) podemos hacer el ingreso de datos para "poblar" esta base de datos.  También es
muy recomendable hacer un ingreso de datos con el objetivo de tratar de violar la integridad de la misma
para verificar que todos los ojetos fueron creados con las restricciones y controles necesarios para mantener
íntegra a la base de datos a pesar de los intentos por ingresar datos incorrectos o inconsistentes.
*/
INSERT INTO sucursal VALUES(1,'LUJAN');
COMMIT;
SELECT * FROM sucursal;
```

```
INSERT INTO tipoiva VALUES(1,10.50);
INSERT INTO tipoiva VALUES(2,21);
COMMIT;
SELECT * FROM tipoiva;
INSERT INTO tipocliente VALUES(1,10);
INSERT INTO tipocliente VALUES(2,15);
INSERT INTO tipocliente VALUES(3,20);
COMMIT;
SELECT * FROM tipocliente;
INSERT INTO cliente VALUES(1,'JOSE','MITRE 231','02323-111111',2,1);
INSERT INTO cliente VALUES(2,'ANA','SAN MARTIN 123','02323-222222',1,3);
COMMIT;
SELECT * FROM cliente;
INSERT INTO vendedor VALUES(1,'PEDRO');
INSERT INTO vendedor VALUES(2,'ANDREA');
COMMIT;
SELECT * FROM vendedor;
INSERT INTO articulo VALUES(100,'PINZA',8.50);
INSERT INTO articulo VALUES(200,'TENAZA',9.50);
INSERT INTO articulo VALUES(300,'MARTILLO',6.25);
INSERT INTO articulo VALUES(400,'MAZA',9.15);
INSERT INTO articulo VALUES(500,'PICO',15.55);
INSERT INTO articulo VALUES(600,'SERRUCHO',21.15);
COMMIT;
SELECT * FROM articulo;
INSERT INTO factura VALUES(1,10001,'01-MAR-2003',1,2);
INSERT INTO detalle VALUES(1,10001,100,2,6.8);
INSERT INTO detalle VALUES(1,10001,200,3,7.6);
INSERT INTO detalle VALUES(1,10001,300,1,5);
INSERT INTO detalle VALUES(1,10001,400,1,7.32);
COMMIT;
INSERT INTO factura VALUES(1,10002,' 02-MAR-2003',1,1);
INSERT INTO detalle VALUES(1,10002,100,1,7.1);
INSERT INTO detalle VALUES(1,10002,200,1,8.1);
COMMIT;
INSERT INTO factura VALUES(1,10003,'today',1,1);
INSERT INTO detalle VALUES(1,10003,100,1,7.2);
INSERT INTO detalle VALUES(1,10003,200,1,8.2);
COMMIT;
SELECT * FROM factura;
SELECT * FROM detalle;
```

Controlamos que el precio facturado no sea menor al precio base del mismo menos el descuento de acuerdo al tipo de cliente en cuestion:

```
/*
Controlamos que el precio facturado no sea menor al precio base del mismo menos el descuento de acuerdo
al tipo de cliente en cuestion:

Solucion:
Debemos controlar los eventos de insert y de update sobre la tabla detalle, debido a que
FireBird no me permite crear un mismo trigger object para controlar ambos eventos (insert y
update) al mismo tiempo, estoy obligado a crear dos trigger events, en este caso, trg_bidetalle
y trg_budetalle (trigger before insert detalle y trigger before update detalle).
La validacion de ambos triggers es identica, para no escribir ambas veces el mismo codigo, podemos
hacer reutilizacion del mismo escribiendo un procedimiento comun a ambos triggers que realice el
```

grueso del trabajo y nos devuelva 0 cuando el precio sea incorrecto y 1 cuando el precio sea correcto. De paso, es una buena oportunidad para ver una aplicacion de los store procedures.
Creo Excepcion para el manejo de error de ambos triggers
Creo Procedure a ser utilizado por ambos triggers para el control de precios
Creo trigger events que hacen uso del procedimiento y de la excepcion
*/

```
CREATE EXCEPTION ex_precio_fac_inc 'Precio a Facturar demasiado barato';

SET TERM !! ;
CREATE PROCEDURE sp_precio_fac_ok (idsuc NUMERIC(3,0),nro NUMERIC(8,0),idart NUMERIC(5,0),precio_fac
NUMERIC(5,2))
RETURNS (precio_ok NUMERIC(1,0))
AS
        DECLARE VARIABLE preciobase NUMERIC(5,2);
        DECLARE VARIABLE pordto     NUMERIC(5,2);
BEGIN
        SELECT ar.precio_base FROM articulo ar
        WHERE ar.idart = :idart
        INTO :preciobase;

        SELECT tc.porc FROM tipocliente tc,cliente cl,factura fc
        WHERE :idsuc = fc.idsuc and :nro = fc.nro and
                cl.idcli = fc.idcli and tc.idtipo = cl.idtipo
        INTO :pordto;

        IF (:precio_fac < (:preciobase - (:preciobase * (:pordto / 100)) ) ) THEN
                precio_ok = 0;
        ELSE
                precio_ok = 1;
        EXIT;
END !!
SET TERM ; !!


SET TERM !! ;
CREATE TRIGGER trg_bidetalle FOR detalle BEFORE INSERT
AS
        DECLARE VARIABLE precio_ok  NUMERIC(1,0);
BEGIN
        EXECUTE PROCEDURE sp_precio_fac_ok NEW.idsuc,NEW.nro,NEW.idart,NEW.precio_fac
RETURNING_VALUES :precio_ok;
        IF (:precio_ok = 0) THEN
                EXCEPTION ex_precio_fac_inc;
END !!
SET TERM ; !!


SET TERM !! ;
CREATE TRIGGER trg_budetalle FOR detalle BEFORE UPDATE
AS
        DECLARE VARIABLE precio_ok  NUMERIC(1,0);
BEGIN
        EXECUTE PROCEDURE sp_precio_fac_ok NEW.idsuc,NEW.nro,NEW.idart,NEW.precio_fac
RETURNING_VALUES :precio_ok;
        IF (:precio_ok = 0) THEN
                EXCEPTION ex_precio_fac_inc;
END !!
```

```
SET TERM ; !!
```

Controlamos que no se realicen cambios en la sucursal o nro de factura, tanto en facturas como en detalle:

```
/*
Controlamos que no se realicen cambios en la sucursal o nro de factura, tanto en facturas como en detalle

Solicion:
Crear excepcion para manejo de error, advertir la imposibilidad de modificar estas columnas
Crear trigger para controlar evento update sobre tabla factura
Crear trigger para controlar evento update sobre tabla detalle
En ambos trigger controlar las columnas sucursal y nro de factura y utilizar la excepcion
previamente creada
*/

CREATE EXCEPTION ex_suc_nro_inc 'No se puede modificar ni Sucursal ni Nro. Factura';

SET TERM !! ;
CREATE TRIGGER trg_budetalle2 FOR detalle BEFORE UPDATE
AS
BEGIN
        IF ( NEW.idsuc <> OLD.idsuc OR NEW.nro <> OLD.nro ) THEN
                EXCEPTION ex_suc_nro_inc;
END !!
SET TERM ; !!

SET TERM !! ;
CREATE TRIGGER trg_bufactura FOR factura BEFORE UPDATE
AS
BEGIN
        IF ( NEW.idsuc <> OLD.idsuc OR NEW.nro <> OLD.nro ) THEN
                EXCEPTION ex_suc_nro_inc;
END !!
SET TERM ; !!
```

Controlamos que si se pretende borrar una factura, también se borre todo su detalle y quede registrado en una tabla de "login" la acción realizada:

```
/*
Controlamos que si se pretende borrar una factura, también se borre todo su detalle y quede registrado en
una tabla de "login" la acción realizada:

Solucion:
Debemos crear una tabla de "login" en donde guardar la informacion de control, en esta tabla guardaremos
la fecha-hora del cambio, un texto relacionado con el cambio y generaremos en forma automatica una clave
numerica.  En este tipo de tablas hay que tener especial cuidado con su clave primaria para evitar que la
misma se repita o sea demasiado extensa (no minima), para ello, la generacion automatica de un id por
parte del SGBD suele ser la mejor opcion.
Crear trigger para controlar evento delete sobre la tabla factura y este debe borrar todas
las lineas de detalle de dicha factura antes de borrar la factura propiamente dicha, luego
guardar en la tabla de login la accion realizada
*/
```

```
CREATE GENERATOR LOG_ID;
SET GENERATOR LOG_ID TO 0;

CREATE TABLE login (
        id integer not null primary key,
        fecha timestamp not null,
        descripcion varchar(255) not null);

SET TERM !! ;
CREATE TRIGGER trg_bdfactura FOR factura BEFORE DELETE
AS
        DECLARE VARIABLE nro_fac NUMERIC(8,0);
        DECLARE VARIABLE mensaje VARCHAR(255);
BEGIN
        IF ( OLD.idsuc IS NOT NULL AND OLD.nro IS NOT NULL ) THEN
          BEGIN
                SELECT COUNT(*) FROM detalle
                WHERE idsuc = OLD.idsuc and nro = OLD.nro
                INTO :nro_fac;
                IF ( :nro_fac > 0 ) THEN
                  BEGIN
                        mensaje = USER || ' Borro Factura. ' || OLD.idsuc || ' ' || OLD.nro || ' con ' || nro_fac
|| ' lineas de detalle';
                        DELETE FROM detalle WHERE idsuc = OLD.idsuc AND nro = OLD.nro;
                        INSERT INTO login VALUES(GEN_ID(LOG_ID,1),'now',:mensaje);
                  END
          END
END !!
SET TERM ; !!
```

Resolvemos las consultas:

El vendedor que vendio mas: (o los vendedores que vendieron mas), podemos pensar esta consulta de la siguiente manera:

1. Calcular el total de venta por vendedor
2. Sacar el máximo del punto 1, para saber el monto de mayor venta
3. Realizar un join entre 1 y 2 por igualdad en el monto de venta; de esta manera, me quedaré con la o las tuplas que tengan el mayor monto.
4. Realizar join entre tabla vendedor y 1 por igualdad de vendedor
5. Mostrar el atributo nombre de la tabla vendedor

Calcular el total de venta por vendedor

```
CREATE VIEW tot_ven (idven,tven) AS
SELECT  F.IDVEN,
        SUM(F.TOTAL)
FROM    VFACTURA F
GROUP BY F.IDVEN;
COMMIT;

SHOW VIEW tot_ven;
```

Sacar el máximo de la consulta anterior

```
CREATE VIEW tot_ven_max (tmax) AS
SELECT  MAX(TVEN)
FROM    tot_ven;
COMMIT;
```

```
SHOW VIEW tot_ven_max;
```

Realizo los distintos joins y muestro nombre del vendedor de mayor venta:

```
SELECT  vv.nombre
FROM    vendedor vv, tot_ven tv, tot_ven_max tvm
WHERE   tvm.tmax = tv.tven and
        vv.idven = tv.idven;
```

El cliente que mas nos compro: (o bien, el o los clientes a los cuales le vendimos mas), podemos pensar esta consulta de la siguiente manera:

1. Calcular el total de venta por cliente
2. Sacar el máximo del punto 1, para saber el monto de mayor venta
3. Realizar un join entre 1 y 2 por igualdad en el monto de venta; de esta manera, me quedaré con la o las tuplas que tengan el mayor monto.
4. Realizar join entre tabla cliente y 1 por igualdad de vendedor
5. Mostrar el atributo nombre de la tabla cliente

Calcular el total de venta por cliente

```
CREATE VIEW tot_cli (idcli,tven) AS
SELECT  F.IDCLI,
        SUM(F.TOTAL)
FROM    VFACTURA F
GROUP BY F.IDCLI;
COMMIT;

SHOW VIEW tot_cli;
```

Sacar el máximo de la consulta anterior

```
CREATE VIEW tot_cli_max (tmax) AS
SELECT  MAX(TVEN)
FROM    tot_cli;
COMMIT;

SHOW VIEW tot_cli_max;
```

Realizo los distintos joins y muestro nombre del cliente de mayor venta:

```
SELECT  cc.nombre
FROM    cliente cc, tot_cli tc, tot_cli_max tcm
WHERE   tcm.tmax = tc.tven and
        cc.idven = tc.idven;
```

Total de ventas por categoría de cliente: podemos pensar esta consulta de la siguiente manera:

1. Join entre factura y cliente
2. Join entre tipocliente y 1
3. Agrupamiento por tipocliente y sumarizar por total de factura

```
SELECT  TC.IDTIPO,
        SUM(F.TOTAL) TOTALTIPO
```

```
FROM    VFACTURA F, CLIENTE C, TIPOCLIENTE TC
WHERE  F.IDCLI = C.IDCLI AND TC.IDTIPO = C.IDTIPO
GROUP BY TC.IDTIPO;
```

Resolvemos los procesos utilizando FireBird:

1) Desarrollar un procedimiento que me permita "mover" a tablas históricas la información relacionada con facturas y sus correspondientes detalles hasta una fecha tope que se indique como parametro de entrada a un procedimiento

```
/*
Las tablas factura_his y detalle_his no poseen las restricciones
que tienen las tablas factura y detalle, debido a que estas funcionan
como tablas historicas, auxiliares, a las cuales, seguramente todos
los usuarios tendran acceso de solo lectura, excepto el o los usuarios
que puedan actualizarla a traves de este proceso.
Ademas, por una cuestion de historicidad, estas tablas pueden ser incon-
sistentes con la tabla de productos, clientes o vendedores.
No podemos incluir dentro del procedimiento un control de transacciones, el
Mismo debera implementarse en la llama al procedimiento, es decir, que la
Ejecucion del procedimiento debera estar dentro de una transaccion. Según
Documentacion consultada, en Firebird el control de transacciones esta delegado
A la aplicación cliente y comando tales como SET TRANSACTION NAME ... estan
Disponibles como SQL embebido; es decir, para ser utilizado desde aplicaciones
Clientes escritas en algun lenguaje anfitrion como podria ser C/C++, Perl, etc.
*/

CREATE TABLE factura_his(idsuc NUMERIC(3,0) NOT NULL,
      nro NUMERIC(8,0) NOT NULL,
      fecha DATE NOT NULL,
      idven NUMERIC(3,0) NOT NULL,
      idcli NUMERIC(5,0) NOT NULL,
      PRIMARY KEY (idsuc,nro));

CREATE TABLE detalle_his(idsuc NUMERIC(3,0) NOT NULL,
      nro NUMERIC(8,0) NOT NULL,
      idart NUMERIC(5,0) NOT NULL,
      cantidad NUMERIC(3,0) NOT NULL,
      precio_fac NUMERIC(5,2) NOT NULL,
      PRIMARY KEY (idsuc,nro,idart));

/*
La llamada a este procedimiento debe estar
realizada dentro de una transaccion
*/

SET TERM !! ;
CREATE PROCEDURE sp_mov_fac (fecha_hasta DATE)
RETURNS (proc_ok NUMERIC(1,0))
AS
      DECLARE VARIABLE can_fac NUMERIC;
      DECLARE VARIABLE mensaje VARCHAR(255);
BEGIN
      SELECT COUNT(*) FROM factura fa
      WHERE fa.fecha < :fecha_hasta
      INTO :can_fac;
```

```
        IF (:can_fac > 0) THEN
          BEGIN
                /* comienzo transaccion */

                        INSERT INTO factura_his
                                SELECT * from factura
                                WHERE fecha < :fecha_hasta;

                        INSERT INTO detalle_his
                                SELECT * from detalle dt
                                WHERE EXISTS (
                                        SELECT * from factura fa
                                        WHERE fa.idsuc = dt.idsuc and fa.nro = dt.nro and
                                            fa.fecha < :fecha_hasta );
                        /*
                          solo borro factura, el trigger trg_bdfactura
                          hara el resto del trabajo
                        */
                        DELETE FROM factura
                                WHERE fecha < :fecha_hasta;

                /* termino transaccion */
                proc_ok = 1;
          END
        ELSE proc_ok = 0;
        EXIT;
    END !!
    SET TERM ; !!

    /*
    Ejemplo de ejecucion de este proceso desde isql:

    SQL> EXECUTE PROCEDURE sp_mov_fac 'today';

    PROC_OK
    =======
        1

    SQL> COMMIT;
    */
```

Lo visto hasta ahora es sólo una mínima y posible implementación de este problema, pero este SGBD nos dá muchísimos más recursos que nos permitirían otro tipo de implementaciones o la utilización de más recursos. También es importante tener en cuenta la aplicación que podamos desarrollar para interactuar con esta base de datos, la cual podrá ser implementada en diferentes arquitecturas, lenguajes y sistemas operativos.

Ejemplo de programa Perl que interactúa con esta base de datos a través del driver DBD-InterBase-040:

Este simple programa sólo hace uso de la View vfactura, creada anteriormente para resolver a través de ella el % de iva, monto de iva y total de factura. En el programa se indican algunos requisitos de software necesarios para que pueda ser utilizado. Esta ampliamente

comentado para que Ud. pueda comprenderlo y hacerle las modificaciones necesarias para utilizarlo en sus futuras implementaciones.

```
#
# Programa demo SQL con FireBird
# Utilizando FireBird / InterBase desde Perl
# Una simple consulta sobre una View
#
# Programacion Aplicada
# Universidad Nacional de Lujan
#
# Lic. Guillermo Cherencio
#
# Se recomienda la lectura del articulo DemoDBI.doc y el seguimiento
# del programa adjunto a dicho documento PruDBI2.pl para entender el
# funcionamiento general de la interfaz DBI y como interactuar con las
# distintas bases de datos del mercado.
#
#
# Modo de ejecucion de este programa:
#
#    perl -w Ferreteria.pl
#
# Atencion!:
# 1) Verificar que el archivo c:\windows\services
#    tenga una entrada del tipo:
#          gds_db 3050/tcp # InterBase Server
#    caso contrario, jamas se conectara a FireBird !
# 2) Por supuesto, debe tener el servidor de FireBird corriendo
# 3) Tiene que tener instalado el packet driver para FireBird
#    puede verificar los paquetes instalados utilizando PPM,
#    desde el prompt DOS tipee:
#       PPM
#    una vez dentro de PPM ejecute el comando:
#       query DBD-InterBase
#    si PPM responde algo similar a esto:
#       DBD-InterBase [0.40] DBD::InterBase is a DBI driver for InterBase, written
#                   using InterBase C API.
#    significa que esta instalado.  Si Ud. quiere saber todos los paquetes
#    que tiene instalado su Perl, tipee solo el comando query
#    Si quiere saber todos los comandos disponibles en PPM tipee help
# 4) Para instalar el package DBD-InterBase-040 haga lo siguiente:
#    Cree una carpeta o directorio temporal cualquiera, supongamos C:\cualquiera
#    En C:\cualquiera copie DBD-InterBase-0.40.zip y luego descomprimalo
#    Dentro de C:\cualquiera, desde el prompt de DOS, tipee:
#       ppm install DBD-InterBase.ppd
# 5) Por supuesto, tiene que tener instalada la interfaz DBI, recuerde que la
#    conexion a una b.d. desde perl es a traves de esta interfaz, de la siguiente
#    manera:
#
# Usuario <---> Programa Perl <---> Interfaz DBI <---> Packet Driver <---> S.G.B.D.
#
#    en nuestro caso:
#
# Usuario <---> Programa Perl <---> DBI <---> DBD-InterBase-040 <---> Servidor FireBird
#
#    en el cd-rom de la materia, podra encontar en \Software\Windows\perl
```

```
#    un perl previamente instalado, el cual ya tiene instalada la interfaz DBI
#    Si no tuvo acceso a esta version, teniendo una conexion abierta a internet,
#    pruebe el siguiente comando desde el prompt DOS:
#
#       ppm install DBI
#
# Para obtener mayor informacion sobre cualquier tema, desde el prompt DOS, tipee:
#
#    perldoc <topico>
#
# Por ejemplo para obtener mayor informacion sobre este packet driver, tipear:
#
#    perldoc DBD::Interbase
#
#
#

#Indico que voy a usar la interfaz DBI
use DBI;
#
# Parametros de conexion con Interbase
#

# Nombre del servidor (no poner su direccion ip)
# consejo: antes de establecer este nombre, desde el prompt DOS, tipee:
#    ping <nombre del servidor>
# si obtiene un mensaje de Reply from ...
# tiene una remota posibilidad de que pueda conectarse
my $servidor = 'localhost';

# Ubicacion de la base de datos dentro del servidor FireBird / Interbase
my $base =  'c:/internet/firebird/bin/ferreteria.gdb';

# Usuario a utilizar para esta conexion
my $usuario = 'sysdba';

# Password del usuario a utilizar en esta conexion
my $password = 'masterkey';

# Flag para el manejo de errores
my $sqlerror = 0;

# Variables adicionales para volcar los datos provenientes de FireBird / Interbase
my $campo = '', @res = ();

#
# Me conecto a Interbase, base de datos ferreteria, usuario sysdba, password masterkey
#
my $dbh = DBI->connect("dbi:InterBase:database=$base;host=$servidor;ib_dialect=3",
                  "$usuario",
                  "$password",
                  { PrintError => 0 }) or $sqlerror = 1;

if ( not $sqlerror ) { #se conecto

   $sth = $dbh->prepare("SELECT * FROM vfactura") or $sqlerror = 1;

   if ( not $sqlerror ) { #se preparo el sql
```

```
    $sth->execute or $sqlerror = 1;

    if ( not $sqlerror ) { #se ejecuto el sql

      print "Recupero las Filas de vfactura:\n";

      while (@res = $sth->fetchrow_array()) { #se recuperan filas

        #muestro cada campo de cada fila
        foreach $campo ( @res ) { print "$campo\t"; }
        print "\n";

      }
      $sth->finish();  #libero los recursos

    } else {
      print "Error en execute:\n$DBI::errstr\n";
    }
  } else {
    print "Error en prepare:\n$DBI::errstr\n";
  }
  $dbh->disconnect();  #me desconecto

} else {
  print "Error en conexion:\n$DBI::errstr\n";
}
exit;
```

Ejemplo de pagina PHP que interactúa con esta base de datos a través de las funciones PHP para InterBase:

Esta página realiza la misma consulta que en ejemplo anterior con Perl. Las funciones Php para el manejo de InterBase se encuentran en la DLL php_interbase.dll; la misma se debe encontrar accesible para el windows que Ud. esté utilizando, debe modificar su Php.ini para "descomentar" la linea que inhabilita dicha extensión. Luego de estas consideraciones, inicie su web server Apache con extensiones php, coloque esta pagina en el directorio que corresponda de acuerdo con la configuración que tiene su web server, estando en ejecución su servidor FireBird, habiendo modificado su archivo services (ver comentario del programa Perl anterior), puede probar la siguiente pagina php:

```
<html>
<title>Consulta de Facturas</title>
<body>
<table border=1><th colspan=9>Facturas</th>
  <tr>
    <td><b>Nro</b></td>
```

```
        <td><b>Sucursal</b></td>
        <td><b>Fecha</b></td>
        <td><b>Vendedor</b></td>
        <td><b>Cliente</b></td>
        <td><b>Neto</b></td>
        <td><b>% IVA</b></td>
        <td><b>IVA</b></td>
        <td><b>Total</b></td>
</tr>
<?php
#
# Pagina PHP demo SQL con FireBird
# Utilizando FireBird / InterBase desde PHP
# Una simple consulta sobre una View
#
# Programacion Aplicada
# Universidad Nacional de Lujan
#
# Lic. Guillermo Cherencio
#
#

#
# Parametros de conexion con Interbase
#

# Nombre del servidor (no poner su direccion ip)
# consejo: antes de establecer este nombre, desde el prompt DOS, tipee:
#    ping <nombre del servidor>
# si obtiene un mensaje de Reply from ...
# tiene una remota posibilidad de que pueda conectarse
$servidor = 'localhost';

# Ubicacion de la base de datos dentro del servidor FireBird / Interbase
$base =  'c:/internet/firebird/bin/ferreteria.gdb';

# Usuario a utilizar para esta conexion
$usuario = 'sysdba';

# Password del usuario a utilizar en esta conexion
$password = 'masterkey';

# Flag para el manejo de errores
$sqlerror = 0;

#Buffer en donde armo la linea de tabla html a mostrar
$linea = '';
$dbh = ibase_connect("$servidor:$base", $usuario, $password) or $sqlerror = 1;
if (!$sqlerror) {
  #formato de fechas dd/mm/yy
  ibase_timefmt("%d/%m/%Y", IBASE_DATE);
  $sth = ibase_query($dbh, 'SELECT * FROM vfactura') or $sqlerror = 1;
  if (!$sqlerror) {
    #nro,idsuc,fecha,idven,idcli,neto,porc,iva,total)
    while ($row = ibase_fetch_row($sth)) {
      $linea="<tr>\n";
       foreach($row as $campo) {
                $linea.="\t<td>$campo</td>\n";
```

```
        }
        $linea.="</tr>\n";
        #envio la linea de  la tabla html
        echo $linea;
      }
      ibase_free_result($sth);
    }
  }
  #En caso de error, muestro el mensaje
  if ($sqlerror) {
    $linea="<tr><td colspan=9>";
    $linea.=ibase_errmsg();
    $linea.="</td></tr>\n";
    echo $linea;
  }
  ibase_close($dbh);
  ?>
</table>
</body>
</html>
```

Glosario:
SGBD / DBMS: Sistema Gestor de Base de Datos / Data Base Managment System, software de propósito general utilizado para la manipulación, creación y administración de base de datos.
A.I.P.: Atributo Indicador Principal, atributo o conjunto de atributos de una entidad que me permiten identificar una entidad del conjunto entidad representado por un rectángulo en el D.E.R.
D.E.R.: Diagrama de Entidad Relación