Implementación asistida de Bases de Datos en Firebird/Interbase

Cherencio, Guillermo Ruben
Universidad Tecnológica Nacional, Facultad Regional Delta

Abstract
El diseño conceptual de una base de datos nos permite determinar las distintas entidades y relaciones existentes en nuestro diseño, el cual podemos traducirlo en un modelo de datos lógico acorde con el modelo relacional y por último, dicho modelo relacional lo podemos implementar utilizando un sistema de gestor de bases de datos relacionales (SGBDR). Un SGBDR ofrece una serie de tecnologías adicionales al modelo relacional, tales como: disparadores (triggers), procedimientos almacenados (stored procedures), tablas derivadas a partir de procedimientos (select procedures); que permiten la resolución de los siguientes casos de implementación típicos que se presentan en este trabajo: atributos derivados o calculados, cardinalidad máxima, tipo - subtipo mutuamente excluyentes, entidad fuerte – entidad débil con discriminante ascendente consecutivo. Estos casos típicos se enmarcan en dos modelos de implementación posibles y extremos: interacción directa sobre las tablas de la base de datos (modelo A) e interacción indirecta sobre las tablas de la base datos (modelo B). Se desarrollan cada una de estas implementaciones posibles en ambos modelos en forma genérica[1], no obstante, se implementan las soluciones propuestas en el lenguaje PL/SQL utilizando el SGBDR Firebird/Interbase® (FB). Todo el trabajo es posible automatizarlo utilizando un software desarrollado a partir de este trabajo para FB 3.0 bajo licencia LGPL. Se abren nuevas líneas de investigación y futuros desarrollos, la posibilidad de agregar nuevos casos que mejoren la productividad y faciliten la implementación de bases de datos relacionales libre de errores.

Palabras Clave
Java Firebird Interbase SQL PL/SQL Base de Datos

Introducción

Los Sistemas de Gestión de Bases de Datos (SGBD)[1] actuales proveen al desarrollador nuevas herramientas y tecnologías para la implementación de base de datos relacionales (BDR). El SGBD Firebird[2] es la versión *open source* del SGBD Interbase de la empresa Borland/Inprise®; es un gestor de base de datos que provee al usuario una amplia funcionalidad, a través de distintos objetos: functions, domains, tables, views, indexes, generators, exceptions, triggers, execute stored procedures, select stored procedures, events, etc.[5]. Estas funcionalidades nos permiten dos tipos de implementaciones extremas[3]:

a) El usuario o la aplicación interactúa directamente con las tablas del sistema: llamaremos a esta opción *"modelo A"*. Esta forma de implementación implica un acceso directo a las tablas, el usuario tiene permisos suficientes para hacer *insert, update, delete, select* sobre las mismas. En este caso, no existe ningún tipo de capa de software que actúe a modo de aislar al usuario o la aplicación de la implementación *real* de las tablas del sistema.

b) El usuario o la aplicación no interactúa directamente con las tablas del sistema: llamaremos a esta opción *"modelo B"*.Esta forma de implementación implica un acceso indirecto a las tablas, el usuario o la aplicación no tiene acceso directo sobre las tablas, sino que éstos interactúan a través de una interfase definida por un conjunto de procedimientos de ejecución (*execute stored procedures*), sobre los cuales el usuario tiene permiso de ejecución. Para hacer *inserts, updates, deletes* sobre las tablas, el usuario utiliza

1 Las soluciones propuestas proponen una serie de restricciones para facilitar su implementación en cualquier SGBDR.

2 Disponible en http://www.firebirdsql.org

3 En medio de estas dos opciones se encuentran implementaciones intermedias que pueden tomar características de uno u otro extremo.

procedimientos de ejecución; para hacer *selects* el usuario utiliza procedimientos de consulta (*select stored procedures)*. El usuario solo tiene permisos de ejecución sobre determinados procedimientos del sistema. El usuario o la aplicación no tiene conocimiento de la existencia de tablas; de esta forma, hay una capa de software intermedia que lo aísla de la implementación y podría considerarse como la API (*application programming interface*) [4] que exhibe hacia el exterior esta BDR y que otorga un mayor nivel de abstracción.

Se proponen una serie de restricciones[4] en cuanto a la programación de disparadores (*triggers*) y procedimientos, con el objeto de obtener resultados aplicables al mayor universo posible de SGBDR's:

-No realizar operaciones de DDL (*data definition language*)[7]
-No realizar operación *select* sobre la tabla objeto del disparador. No se puede utilizar la tabla sobre la cual se disparó el evento, es decir, sobre un trigger de *before update* sobre la tabla X, no puedo hacer una consulta sobre la tabla X, ni mencionarla.
-Asumir el siguiente orden de ejecución de una transacción: 1. Eventos de tipo *before* 2. Aplicación de reglas de integridad (*constraints: foreign key, primary key, unique, check, domains, unique index, etc.*). 3. Eventos de tipo *after.*
-Otorgar permisos *select, update, delete, insert* sobre los objetos que utiliza el disparador o procedimiento.
-Un disparador o procedimiento puede tener permisos para hacer operaciones sobre un objeto X, sin que el usuario de la transacción actual tenga permisos sobre ese objeto X.

4 No se profundizará en cuanto a las razones técnicas de estas restricciones, que se consideran fuera del alcance de este trabajo.

Para facilitar el análisis de casos y no extender este trabajo, se asume que todas las claves extranjeras (*foreign key*)[6] se han implementado con su comportamiento por defecto[5] y se otorgan todos los permisos sobre los disparadores y procedimientos acorde con el tipo de modelo a implementar[6].

Teniendo en cuenta lo indicado en los puntos anteriores, la programación de disparadores requiere de atributos adicionales para hacer conteos, sumatorias, etc. La ubicación física de estos atributos varía acorde con el tipo de implementación:

-modelo A: se deben crear nuevas tablas,las llamaremos "*tablas auxiliares"*. Toda tabla auxiliar A esta vinculada a la tabla X. El conjunto de atributos de A esta formado por el conjunto de atributos primos[6] de X más el conjunto de atributos requeridos por los disparadores. La tabla A contendrá en todo momento la misma cantidad de tuplas que X y los mismos valores de clave primaria. Ningún usuario tendrá permisos sobre la tabla A; solo los disparadores y procedimientos que la utilicen tendrán permisos sobre la tabla A.

-modelo B: no requiere de tablas auxiliares y el conjunto de atributos requeridos por los disparadores se agregan a las tablas del sistema.

El presente trabajo pretende resolver una serie de casos típicos de implementación de BDR tanto para el "modelo A" como para el "modelo B" y

5 Se refiere al comportamiento "no action"; los otros comportamientos posibles son: cascade update, cascade delete, set default, set null.

6 Por ejemplo, en modelo A, los objetos disparadores deben tener permisos apropiados para insertar, borrar, actualizar, etc. las tablas auxiliares; mientras que el usuario solo tendrá permisos sobre las tablas no auxiliares.

proveer de un software que permita al desarrollador una implementación asistida de una BDR acorde con el modelo que sea requerido.

Elementos del Trabajo y metodología

Todo el trabajo se desarrolló en las siguientes etapas, tomando los lineamientos generales de Blanchette[3]:

1. Requerimientos
2. Análisis de casos.
3. Implementación manual de casos.
4. Implementación automática de casos.
5. Implementación en BDR de ejemplo y Testing.
6. Empaquetado, publicación y distribución.

1. Requerimientos

Los casos a ser implementados deben hacerse teniendo en cuenta las restricciones indicadas en la introducción. La solución manual deberá realizarse en PL/SQL utilizando el SGBDR FB 3.0, tanto para el "modelo A" como para el "modelo B". La metodología a utilizar en cada caso sera la siguiente:

1. Identificar -según el problema a resolver- las tablas involucradas y si es una implementación de "modelo A" o "modelo B".
2. Armar una grilla en donde ubicar todos los posibles tipos de eventos (*insert, update, delete*; considerando además, que pueden ser de tipo *before* o *after*) en las filas y poner en las columnas cada una de las tablas involucradas.
3. Completar la grilla, escribiendo en cada celda en pseudocódigo, la acción a realizar para ese tipo de evento y esa tabla en particular.
4. Realizar la programación PL/SQL.
4.1 Crear todos los objetos requeridos por la solución propuesta.
4.2 Por cada celda de la grilla que contiene pseudocódigo, crear un disparador, con la siguiente nomenclatura: TRG_{BI | AI | BD | AD | BU | AU}<tabla> donde B significa *before*,A significa *after*, I significa *insert*, D significa *delete*, U significa *udpate.*

El software a desarrollar para facilitar la implementación de los casos típicos de implementación deberá ser bajo licencia LGPL, publicado en plataforma de desarrollo colaborativo, multiplataforma, utilización de metodo de acceso standard a BDR. El usuario destinatario de este software es un desarrollador de software orientado a la implementación de BDR y programador PL/SQL; el software deberá poder conectarse a distintas BDR's y sobre la misma, implementar los casos que indique el usuario y éste deberá asistir con el ingreso de datos que requiera cada caso. El proceso de los casos no puede ser totalmente automatizado sino que es asistido por el usuario.

2. Análisis de casos.

Se han determinado los siguientes casos de implementación:

2.1 Implementación básica modelo B.

Se considera fuera del alcance de este trabajo la traducción de un modelo de datos lógico relacional (MDLR)[2] a una implementación de tipo modelo A. Se asume que todo el proceso comienza a partir de una BDR que contiene una serie de tablas acorde con un MDLR previamente diseñado. A partir de aquí se considera a la BDR como una implementación inicial de modelo A, no obstante, es posible considerar como un caso típico la derivación de esta BDR en un modelo B, que podría hacerse acorde con estos lineamientos:

Por cada tabla hacer:

-Crear procedimiento de ejecución SP_{nombre tabla} que recibirá como argumentos el tipo de operación a realizar (I → *insert*, D → *delete*, U → *update*) y luego un argumento por cada atributo de la tabla. el procedimiento realizará las operaciones de inserción, actualización y borrado de tuplas. El procedimiento devolverá un código de retorno (0 → operación satisfactoria, cualquier otro valor implica error) y un mensaje de error (en caso de no existir error, devolverá ”OK”)

-Crear procedimiento de selección SP_S{nombre tabla} que devuelva todas las tuplas de la tabla.

2.2 Atributos derivados.

Los atributos derivados o calculados pueden implementarse de dos formas:

a) como calculados, en forma declarativa, utilizando clausulas como “COMPUTE BY” en la creación de la tabla, derivando la tabla original en una vista (*view*) o procedimiento de selección.

b) como atributos físicos redundantes. Debido a que la redundancia es condición para luego provocar una inconsistencia en la BDR, se debe implementar una redundancia “controlada”.

La implementación de atributos derivados de tipo a) tanto en modelo A como en modelo B, es trivial y se realiza tal como se indicó.

2.2.1 Atributos derivados redundantes modelo A y Modelo B

Para ambos modelos puede utilizarse la misma implementación. Dado el siguiente ejemplo:

producto(**idp**,descripcion,stock)
factura(**nro**,fecha,monto)
item(**nro,idp**,cantidad,precio,subtotal)
nro foreign key factura
idp foreign key producto

donde subtotal es un atributo derivado cuyo calculo es cantidad * precio, los disparadores requeridos son:

EVENTO		**ITEM**
INSERT	BEFORE	subotal=cantidad*precio
	AFTER	
UPDATE	BEFORE	subotal=cantidad*precio
	AFTER	
DELETE	BEFORE	
	AFTER	

2.3 Tipo – Subtipo mutuamente excluyentes.

La misma solución es aplicable tanto para el modelo A como para el modelo B, dado el siguiente ejemplo:

persona(**dni**,apellido,nombre)
contratista(**dni**,valor_hora)
dni foreign key persona
empleado(**dni,**salario)
dni foreign key persona

en donde una persona puede ser contratista o empleado, pero no ambos; la implementación para ambos modelos es:

EVENTO		**CONTRATISTA**	**EMPLEADO**
I	B	Si existe tupla en empleado entonces Error!	Si existe tupla en contratista entonces Error!
	A		
U	B	Si dni cambió entonces Si existe nuevo dni en empleado entonces Error!	Si dni cambió entonces Si existe nuevo dni en contratista entonces Error!
	A		
D	B		
	A		

2.4 Cardinalidad máxima

Dado el siguiente ejemplo:

```
producto(idp,descripcion,stock)
factura(nro,fecha,monto)
item(nro,idp,cantidad,precio)
        nro foreign key factura
        idp foreign key producto
```

supongamos que una factura no puede tener mas de 20 items.

2.4.1 Cardinalidad máxima modelo A

Implementar un contador de items (ci) en tabla auxiliar:

```
producto(idp,descripcion,stock)
factura(nro,fecha,monto)
item(nro,idp,cantidad,precio)
        nro foreign key factura
        idp foreign key producto
factura_aux(nro,ci)
```

y los disparadores requeridos son:

EVENTO		FACTURA	ITEM
I	B		Si factura_aux.ci >= 20 entonces Error!
	A	Insertar tupla en factura_aux	factura_aux.ci = factura_aux.ci + 1
U	B		Si nro cambió entonces Si factura_aux.ci >= 20 entonces Error!
	A		Si nro cambió entonces restar 1 en factura_aux.ci para nro viejo y sumar 1 en factura_aux.ci para nro nuevo
D	B		
	A	Borrar tupla en factura_aux	factura_aux.ci = factura_aux.ci - 1

2.4.2 Cardinalidad máxima modelo B

Implementar un contador de items (ci) en tabla no auxiliar:

```
producto(idp,descripcion,stock)
factura(nro,fecha,monto,ci)
item(nro,idp,cantidad,precio)
        nro foreign key factura
        idp foreign key producto
```

y los disparadores requeridos son:

EVENTO		FACTURA	ITEM
I	B	ci=0	Si factura.ci >= 20 entonces Error!
	A		factura.ci = factura.ci + 1
U	B		Si nro cambió entonces Si factura.ci >= 20 para nro nuevo entonces Error!
	A		Si nro cambió entonces restar 1 en factura.ci para nro viejo y sumar 1 en factura.ci para nro nuevo
D	B		
	A		factura.ci = factura.ci - 1

2.5 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo.

Dado el siguiente ejemplo:

```
libro(isbn,titulo)
ejemplar(isbn,nro)
        isbn foreign key libro
```

donde libro es una entidad fuerte[6], ejemplar es una entidad débil y nro es el discriminante; los valores de nro van de 1..N para cada isbn de la biblioteca y se pretende que sean correlativos ascendentes. No se permiten cambios en clave primaria y las tuplas de ejemplar solo se borran en forma descendente.

2.5.1 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo A

Implementar un atributo que mantenga el ultimo numero de ejemplar (ue) por cada isbn en tabla auxiliar:

```
libro(isbn,titulo)
ejemplar(isbn,nro)
        isbn foreign key libro
libro_aux(isbn,ue)
```

y los disparadores requeridos son:

EVENTO		LIBRO	EJEMPLAR
I	B		nro=libro_aux.ue + 1
	A	Insertar tupla en libro_aux, ue=0	libro_aux.ue = libro_aux.ue + 1
U	B		No permitir cambios en clave primaria
	A		
D	B		Si nro <> libro_aux.ue entonces Error!
	A	Borrar tupla en libro_aux	libro_aux.ue = libro_aux.ue - 1

2.5.2 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo B

El atributo ultimo numero de ejemplar (ue) implementarlo en tablas no auxiliares:

```
libro(isbn,titulo,ue)
ejemplar(isbn,nro)
        isbn foreign key libro
```

y los disparadores requeridos son:

EVENTO		LIBRO	EJEMPLAR
I	B	ue=0	nro=libro.ue + 1
	A		libro.ue = libro.ue + 1
U	B		No permitir cambios en clave primaria
	A		
D	B		Si nro <> libro.ue entonces Error!
	A		libro.ue = libro.ue - 1

3. Implementación manual de casos.

3.1 Implementación básica modelo B.

Dada el siguiente tabla de ejemplo:

```
libro(isbn,titulo)
```

el código a generar sería el siguiente[7]:

```
CREATE PROCEDURE SP_LIBRO (
OP CHAR(1),
ISBN TYPE OF COLUMN LIBRO.ISBN,
TITULO TYPE OF COLUMN LIBRO.TITULO )
RETURNS (
RET INTEGER,
RETMSG VARCHAR(100) ) AS
BEGIN
  RET=0;RETMSG='OK';
  IF ( OP = 'I' ) THEN
    INSERT INTO LIBRO(ISBN,TITULO)
VALUES(:ISBN,:TITULO);
  IF ( OP = 'U' ) THEN
    UPDATE LIBRO SET TITULO = :TITULO
WHERE ISBN = :ISBN;
  IF ( OP = 'D' ) THEN
    DELETE FROM LIBRO WHERE ISBN =
:ISBN;
  WHEN ANY DO BEGIN
    RET=SQLCODE;
    RETMSG='OP=' || OP || ' ERROR=' || SQLCODE
|| ' TABLE=' || 'LIBRO';
  END
END

CREATE PROCEDURE SP_SLIBRO
RETURNS (
    ISBN TYPE OF COLUMN LIBRO.ISBN,
    TITULO TYPE OF COLUMN
LIBRO.TITULO) AS
BEGIN
```

7 En negritas se resaltan las partes del script específicas para esta tabla en particular.

```
   FOR SELECT ISBN,TITULO FROM LIBRO
   INTO :ISBN,:TITULO DO SUSPEND;
END
```

3.2.1 Atributos derivados redundantes modelo A y modelo B.

```
CREATE TRIGGER TRG_AIITEM FOR ITEM
BEFORE INSERT AS
BEGIN
  NEW.SUBTOTAL = NEW.CANTIDAD *
NEW.PRECIO;
END

CREATE TRIGGER TRG_BUITEM FOR ITEM
BEFORE UPDATE AS
BEGIN
  NEW.SUBTOTAL = NEW.CANTIDAD *
NEW.PRECIO;
END
```

3.3 Tipo – Subtipo mutuamente excluyentes.

```
CREATE EXCEPTION EX_ANY 'Error!';

CREATE TRIGGER TRG_BICONTRATISTA
FOR CONTRATISTA BEFORE INSERT AS
BEGIN
   IF ( EXISTS ( SELECT * FROM EMPLEADO
      WHERE DNI = NEW.DNI ) ) THEN
      EXCEPTION EX_ANY 'No puede ser
empleado y contratista!';
END

CREATE TRIGGER TRG_BIEMPLEADO FOR
EMPLEADO BEFORE INSERT AS
BEGIN
   IF ( EXISTS ( SELECT * FROM
CONTRATISTA
      WHERE DNI = NEW.DNI ) ) THEN
      EXCEPTION EX_ANY 'No puede ser
empleado y contratista!';
END

CREATE TRIGGER TRG_BUEMPLEADO FOR
EMPLEADO BEFORE UPDATE AS
BEGIN
   IF ( NEW.DNI <> OLD.DNI ) THEN
     IF ( EXISTS ( SELECT * FROM
CONTRATISTA
      WHERE DNI = NEW.DNI ) ) THEN
         EXCEPTION EX_ANY 'No puede ser
empleado y contratista!';
END
```

```
CREATE TRIGGER TRG_BUCONTRATISTA
FOR CONTRATISTA BEFORE UPDATE AS
BEGIN
   IF ( NEW.DNI <> OLD.DNI ) THEN
     IF ( EXISTS ( SELECT * FROM EMPLEADO
      WHERE DNI = NEW.DNI ) ) THEN
         EXCEPTION EX_ANY 'No puede ser
empleado y contratista!';
END
```

3.4.1 Cardinalidad máxima modelo A

```
CREATE EXCEPTION EX_ANY 'Error!';

CREATE TRIGGER TRG_AIFACTURA FOR
FACTURA AFTER INSERT AS
BEGIN
   INSERT INTO FACTURA_AUX(NRO,CI)
VALUES(NEW.NRO,0);
END

CREATE TRIGGER TRG_ADFACTURA FOR
FACTURAAFTER DELETE AS
BEGIN
   DELETE FROM FACTURA_AUX WHERE
NRO = OLD.NRO;
END

CREATE TRIGGER TRG_BIITEM FOR ITEM
BEFORE INSERT AS
BEGIN
   IF ( (SELECT CI FROM FACTURA_AUX
         WHERE NRO = NEW.NRO) >= 20 )
THEN
      EXCEPTION EX_ANY 'No puede haber mas
de 20 items!';
END

CREATE TRIGGER TRG_AIITEM FOR ITEM
AFTER INSERT AS
BEGIN
   UPDATE FACTURA_AUX SET CI = CI + 1
   WHERE NRO = NEW.NRO;
END

CREATE TRIGGER TRG_BUITEM FOR ITEM
BEFORE UPDATE AS
BEGIN
   IF ( NEW.NRO <> OLD.NRO AND
        (SELECT CI FROM FACTURA_AUX
        WHERE NRO = NEW.NRO) >= 20 )
THEN
      EXCEPTION EX_ANY 'No puede haber mas
de 20 items!';
END
```

```
CREATE TRIGGER TRG_AUITEM FOR ITEM
AFTER UPDATE AS
BEGIN
   IF ( NEW.NRO <> OLD.NRO ) THEN BEGIN
      UPDATE FACTURA_AUX SET CI = CI - 1
WHERE NRO = OLD.NRO;
      UPDATE FACTURA_AUX SET CI = CI + 1
WHERE NRO = NEW.NRO;
   END
END

CREATE TRIGGER TRG_ADITEM FOR ITEM
AFTER DELETE AS
BEGIN
   UPDATE FACTURA_AUX SET CI = CI - 1
WHERE NRO = OLD.NRO;
END
```

3.4.2 Cardinalidad máxima modelo B

```
CREATE EXCEPTION EX_ANY 'Error!';

CREATE TRIGGER TRG_BIFACTURA FOR
FACTURA BEFORE INSERT AS
BEGIN
   NEW.CI = 0;
END

CREATE TRIGGER TRG_BIITEM FOR ITEM
BEFORE INSERT AS
BEGIN
   IF ( (SELECT CI FROM FACTURA
         WHERE NRO = NEW.NRO) >= 20 )
THEN EXCEPTION EX_ANY 'No puede haber
mas de 20 items!';
END

CREATE TRIGGER TRG_AIITEM FOR ITEM
AFTER INSERT AS
BEGIN
   UPDATE FACTURA SET CI = CI + 1 WHERE
NRO = NEW.NRO;
END

CREATE TRIGGER TRG_BUITEM FOR ITEM
BEFORE UPDATE AS
BEGIN
   IF ( NEW.NRO <> OLD.NRO AND
        (SELECT CI FROM FACTURA
        WHERE NRO = NEW.NRO) >= 20 )
THEN EXCEPTION EX_ANY 'No puede haber
mas de 20 items!';
END

CREATE TRIGGER TRG_AUITEM FOR ITEM
AFTER UPDATE AS
BEGIN
   IF ( NEW.NRO <> OLD.NRO ) THEN BEGIN
      UPDATE FACTURA SET CI = CI - 1
WHERE NRO = OLD.NRO;
      UPDATE FACTURA SET CI = CI + 1
WHERE NRO = NEW.NRO;
   END
END

CREATE TRIGGER TRG_ADITEM FOR ITEM
AFTER DELETE AS
BEGIN
   UPDATE FACTURA SET CI = CI - 1 WHERE
NRO = OLD.NRO;
END
```

3.5.1 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo A

```
CREATE EXCEPTION EX_ANY 'Error!';

CREATE TRIGGER TRG_AILIBRO FOR LIBRO
AFTER INSERT AS
BEGIN
   INSERT INTO LIBRO_AUX(ISBN,UE)
VALUES(NEW.ISBN,0);
END

CREATE TRIGGER TRG_ADLIBRO FOR
LIBROAFTER DELETE AS
BEGIN
   DELETE FROM LIBRO_AUX WHERE ISBN =
OLD.ISBN;
END

CREATE TRIGGER TRG_BIEJEMPLAR FOR
EJEMPLARBEFORE INSERT AS
   DECLARE VARIABLE VUE TYPE OF
COLUMN LIBRO_AUX.UE;
BEGIN
   SELECT UE FROM LIBRO_AUX WHERE
ISBN = NEW.ISBN INTO :VUE;
   NEW.NRO = VUE+1;
END

CREATE TRIGGER TRG_AIEJEMPLAR FOR
EJEMPLAR AFTER INSERT AS
BEGIN
   UPDATE LIBRO_AUX SET UE = UE + 1
WHERE ISBN = NEW.ISBN;
END

CREATE TRIGGER TRG_BUEJEMPLAR FOR
EJEMPLARBEFORE UPDATE AS
BEGIN
   IF ( NEW.ISBN <> OLD.ISBN OR
```

```
      NEW.NRO  <> OLD.NRO ) THEN
      EXCEPTION EX_ANY 'No puede cambiar
clave primaria';
END

CREATE TRIGGER TRG_BDEJEMPLAR FOR
EJEMPLAR BEFORE DELETE AS
   DECLARE VARIABLE VUE TYPE OF
COLUMN LIBRO_AUX.UE;
BEGIN
   SELECT UE FROM LIBRO_AUX WHERE
ISBN = OLD.ISBN INTO :VUE;
   IF ( VUE <> OLD.NRO ) THEN EXCEPTION
EX_ANY 'Se borra a partir del ultimo ejemplar!';
END

CREATE TRIGGER TRG_ADEJEMPLAR FOR
EJEMPLARAFTER DELETE AS
BEGIN
   UPDATE LIBRO_AUX SET UE = UE - 1
WHERE ISBN = OLD.ISBN;
END
```

3.5.2 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo B

```
CREATE EXCEPTION EX_ANY 'Error!';

CREATE TRIGGER TRG_BILIBRO FOR LIBRO
BEFORE INSERT AS
BEGIN
   NEW.UE=0;
END

CREATE TRIGGER TRG_BIEJEMPLAR FOR
EJEMPLAR BEFORE INSERT AS
   DECLARE VARIABLE VUE TYPE OF
COLUMN LIBRO.UE;
BEGIN
   SELECT UE FROM LIBRO WHERE ISBN =
NEW.ISBN INTO :VUE;
   NEW.NRO = VUE+1;
END

CREATE TRIGGER TRG_AIEJEMPLAR FOR
EJEMPLAR AFTER INSERT AS
BEGIN
   UPDATE LIBRO SET UE = UE + 1 WHERE
ISBN = NEW.ISBN;
END

CREATE TRIGGER TRG_BUEJEMPLAR FOR
EJEMPLAR BEFORE UPDATE AS
BEGIN
   IF ( NEW.ISBN <> OLD.ISBN OR
      NEW.NRO  <> OLD.NRO ) THEN
       EXCEPTION EX_ANY 'No puede cambiar
clave primaria';
END

CREATE TRIGGER TRG_BDEJEMPLAR FOR
EJEMPLAR BEFORE DELETE AS
   DECLARE VARIABLE VUE TYPE OF
COLUMN LIBRO.UE;
BEGIN
   SELECT UE FROM LIBRO WHERE ISBN =
OLD.ISBN  INTO :VUE;
   IF ( VUE <> OLD.NRO ) THEN EXCEPTION
EX_ANY 'Se borra a partir del ultimo ejemplar!';
END

CREATE TRIGGER TRG_ADEJEMPLAR FOR
EJEMPLAR AFTER DELETE AS
BEGIN
   UPDATE LIBRO SET UE = UE - 1 WHERE
ISBN = OLD.ISBN;
END
```

4. Implementación automática de casos.

Acorde con los requerimientos se propone el desarrollo de una aplicación cliente Java SE utilizando el framework Swing[8] como interfaz gráfica, permite la conexión a SGBDR FB 3.0.

La salida que emitirá esta aplicación para cada caso es un script de tipo DDL para la creación de objetos; permitirá copiar, editar y ejecutar el script. Se analiza cada solución manual para determinar el input necesario para implementar cada opción:

4.1 Implementación básica modelo B.

Entrada: seleccionar tablas del sistema
Proceso: por cada tabla seleccionada, generar procedimiento de ejecución y selección.

4.2 Atributos derivados redundantes
4.2.1 Atributos derivados redundantes modelo A y modelo B.

Entrada: seleccionar tablas del sistema, para cada tabla seleccionada, indicar los

atributos nuevos a implementar como derivados redundantes, por cada atributo indicar expresión basada en otros atributos y definición DDL del atributo.
Proceso: por cada tabla seleccionada, modificar la tabla, generar disparadores before insert y before update.

4.3 Tipo – Subtipo mutuamente excluyentes.

Entrada: mensaje de error, seleccionar tabla tipo, por cada tabla tipo, seleccionar tablas subtipo.
Proceso: por cada tabla subtitpo, generar disparadores de before insert y before update.

4.4 Cardinalidad máxima
4.4.1 Cardinalidad máxima modelo A

Entrada: seleccionar tablas padre, por cada tabla padre, indicar tabla hija, cantidad máxima de items, mensaje de error.
Proceso: por cada tabla padre seleccionada, crear tabla auxiliar, crear disparadores after insert, after delete; por cada tabla hija, crear disparadores before insert, after insert, before update, after update, after delete.

4.4.2 Cardinalidad máxima modelo B

Entrada: seleccionar tablas padre, por cada tabla padre, indicar tabla hija, cantidad máxima de items, mensaje de error.
Proceso: por cada tabla padre, modificar tabla, crear disparador before insert, por cada tabla hija, crear disparadores before insert, after insert, before upate, after update, after delete.

4.5 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo
4.5.1 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo A

Entrada: seleccionar tabla fuerte, por cada tabla fuerte, indicar tabla débil, seleccionar discriminante, mensaje de error en borrado de tuplas, mensaje de error en cambio de clave primaria.
Proceso: por cada tabla fuerte, crear tabla auxiliar, crear disparadores after insert, after delete; por cada tabla débil, crear disparadores before insert, after insert, before update, before delete, after delete.

4.5.2 Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo B

Entrada: seleccionar tabla fuerte, por cada tabla fuerte, indicar tabla débil, seleccionar discriminante, mensaje de error en borrado de tuplas, mensaje de error en cambio de clave primaria.
Proceso: por cada tabla fuerte, modificar tabla fuerte, crear disparador before insert; por cada tabla débil, crear disparadores before insert, after insert, before update, before delete, after delete.

5. Implementación en BDR de ejemplo y Testing.

El software desarrollado se distribuye con una base de datos FB 3.0 test.fdb que incluye todas las tablas utilizadas en este trabajo, para ejecutar todos los casos de implementación que aquí se describen y probar su funcionamiento.

6. Empaquetado, publicación y distribución.

El software desarrollado se distribuye bajo licencia LGPL[8], se incluyen los programas fuentes, ejemplos, documentación, etc., que los usuarios pueden descargar, compilar y ejecutar. La distribución y publicación de este trabajo se realiza a través del portal sourceforge[9] bajo

8 http://www.gnu.org/licenses/lgpl.html
9 http://www.sourceforge.net

el nombre de proyecto *fbjreplicator*[10]; este proyecto fue concebido como una herramienta de replicación para FB 2.5[11] y el software resultante de este trabajo se incorpora como una interfaz gráfica adicional del mismo, de esta forma, el usuario puede partir de una base de datos centralizada, optar por una implementación modelo A o B, aplicar los casos de implementación que desee y luego replicarla.

Resultados

Este trabajo se ha aplicado en trabajos prácticos integradores de los alumnos de la asignatura 11078 Base de Datos II y 11077 Base de Datos I de la carrera Licenciatura en Sistemas de Información de la Universidad Nacional de Luján a finales de la cursada 2015, también se han recibido experiencias y recomendaciones por correo electrónico de usuarios y ex alumnos que han utilizado el software con fines profesionales, destacando su facilidad de uso y mejora en la productividad.

Discusión

El presente trabajo ha sido realizado sobre una plataforma Linux, no obstante, puede utilizarse en otras plataformas, ya que ha sido desarrollado 100% en Java y no posee dependencias directas con el sistema operativo utilizado en su desarrollo.

Todo el desarrollo se encuentra en su primera versión y esta orientado a FB, siendo dependiente del driver JDBC Jaybird, no obstante, se ha considerado en su diseño la aislación suficiente del código para que luego puedan incorporarse otros drivers JDBC y permitir que el software pueda interactuar con otros SGBDR's.

La integración del software resultante de este trabajo con *fbjreplicator* facilita el uso de la herramienta dentro de un proyecto mayor de implementación de BDR replicada.

Se han obtenido comentarios favorables en cuanto a la velocidad de generación de scripts y si bien sólo funciona con FB 3.0, los últimos cambios en PL/SQL permiten una mayor compatibilidad con otros SGBDR's; permitiendo a los usuarios exportar los scripts, modificarlos y ejecutarlos en otros SGBDR's.

Se ha considerado en el diseño de clases de este software la posibilidad de cambiar la generación de scripts para otros lenguajes o sintaxis distintas a PL/SQL.

Este software puede considerárselo como un elemento mas dentro de un proceso de diseño e implementación de BDR's, la licencia LGPL permite que el mismo sea modificado para adaptarlo a las herramientas de diseño que utiliza el usuario, ampliar su funcionalidad (por ejemplo, incluir la posibilidad de traducción de un MDLR a una implementación de una BDR física en un SGBDR determinado) de forma tal, de automatizar e integrar todo el proceso de diseño e implementación de BDR's.

Conclusión

Las bases de datos no relacionales han ganado mucho terreno motivado por el desarrollo de nuevas tecnologías, no obstante, las BDR's ocupan aún un rol fundamental y continúan siendo una opción apropiada para muchos sistemas de gestión. Las BDR's no han sido ajenas a una evolución en cuanto a los nuevos objetos y lenguajes soportados.

Es posible inferir la posibilidad de error por mal uso de esos nuevos objetos, por ejemplo, es habitual observar en mis alumnos cuando están en una implementación de tipo “modelo B” implementar reglas de integridad

10 https://sourceforge.net/projects/fbjreplicator

11 En este momento se esta migrando a FB 3.0 para que ambas interfases gráficas funcionen con la misma versión de FB.

semánticas dentro de los procedimientos, aumentando su complejidad y realizando controles innecesarios, en vez de hacerlo con el uso de disparadores junto con una correcta utilización de reglas de integridad no semánticas[6] que posee el SGBDR. Otra fuente importante de error es el uso incorrecto de los tipos de eventos de los disparadores, hay ciertas acciones que deben hacerse en disparadores de tipo *before* y otras que deben hacerse en disparadores de tipo *after*[12]; esto puede impactar negativamente en el *rollback* de transacciones, a pesar de dar una ilusión de "correcto funcionamiento". Esta herramienta pretende evitar ese tipo de errores.

Este trabajo abre nuevas lineas de investigación en cuanto a nuevos casos típicos de implementación que puedan agregarse, en cuanto a ampliaciones de su funcionalidad, soportar otros SGBDR's, etc. permitiendo ampliar la base de usuarios destinatario de este trabajo y aportando a una mejora en la productividad y eficacia en la implementación de BDR's..

Agradecimientos

Al Lic. Carlos Gabriel Antonio Rodriguez de la Universidad Nacional de Luján con quien trabajo desde hace mas de 23 años en bases de datos relacionales y quien ha motivado la presentación del presente trabajo.

Referencias

[1] Castaño Miguel, Adoración de; Velthuis Piattini, Mario; Martinez, Esperanza Marcos, Diseño de Bases de Datos Relacionales, Editorial Ra-Ma, Madrid, 2000, ISBN 84-7897-385-0, Pag 5.

[2] Silberschatz, Abraham; Korth, Henry F.; Sudarshan S., Fundamentos de Bases de Datos, 4ta.Ed., Ed. Mc Graw Hill, Madrid, 2002, ISBN: 0-07-228363-7, Pag. 17.

[3] Blanchette, Jasmin, The Little Manual of API Design, Trolltech, a Nokia company, June 19, 2008, disponible en http://www4.in.tum.de/~blanchet/api-design.pdf

[4] Jackson, Daniel, Software Abstractions, MIT Press, 2006.

[5] Chan, Hubert; Yashkir, Dmytro, Conceptual Architecture for InterBase/Firebird, Faculty of Mathematical, Statistical and Computer Sciences, University of Waterloo, Canada, January 29, 2002, disponible en http://www.ibphoenix.com/resources/documents/design/doc_25

[6] Elmasri, Ramez; Navathe, Shamkant B., Sistemas de Bases de Datos, Conceptos Fundamentales, 2da Ed, Addison-Wesley Iberoamericana, ISBN 0-201-65370-2, Pag. 139-155.

[7] Hansen, Gary W.; Hansen, James V., Diseño y Administración de Bases de Datos, 2da Ed, Prentice Hall, Pag. 499.

[8] Eckel, Bruce; Thinking in Java, 2da Ed, Prentice Hall, ISBN 0-13-027363-5, Pag. 689-790.

12 La discusión de este tópico se considera fuera del alcance de este trabajo.

Datos de Contacto:

Cherencio, Guillermo Ruben. Universidad Tecnológica Nacional, Facultad Regional Delta. San Martin 1171, CP 2804, Campana, Provincia de Buenos Aires, República Argentina. E-mail: grchere@yahoo.com.