Revista Tecnología y Ciencia
ISSN 1666-6933 DOI: https://doi.org/10.33414/rtyc.33.157-174.2018

Implementación Asistida de Bases de Datos Relacionales en *firebird/ interbase*

Guillermo R. Cherencio
Universidad Tecnológica Nacional, Facultad Regional Delta, Buenos Aires, Argentina
gcherencio@unlu.edu.ar

Presentación 18/11/2016
Aprobación 20/07/2017

Resumen

Un universo de discurso puede representarse a través de un diseño conceptual traducible en un modelo de datos lógico relacional y puede implementarse utilizando un sistema de gestor de bases de datos relacionales como Firebird/ Interbase (FB). FB tiene una serie de tecnologías tales como: disparadores, procedimientos almacenados, tablas derivadas a partir de procedimientos; que permiten la resolución de los siguientes casos de implementación genéricos: atributos derivados, 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 (modelo A) e interacción indirecta (modelo B) sobre las tablas de la base de datos. Se desarrollan cada una de estas implementaciones posibles en ambos modelos; se automatiza todo el desarrollo utilizando un software desarrollado como producto de este trabajo, que facilita la correcta implementación de bases de datos relacionales.

Palabras clave: Firebird PL/SQL Modelo Relacional

Aided Implementation of Relational Databases in Firebird/ Interbase

Abstract

An universe of discourse (UoD) can be represented from conceptual design translatable into a logical relational data model and it can be implemented using a relational database management system such as Firebird/Interbase (FB). FB has several technologies e.g.: triggers, stored procedures, select stored

Revista Tecnología y Ciencia
ISSN 1666-6933

procedures; they enable the following generic implementation issues e.g.: derived attributes, max cardinality, type – subtype relationship mutuallyexclusive, regular entity type – weak entity type with ascending consecutive discriminator. These typical issues framed into two possible and extreme implementation model: direct interaction (model A) and indirect interaction (model B) over database tables. I develop each generic implementation issue in both models: A and B; the whole development can be automatized using an ad hoc software developed for this paper, it makes easier a correct implementation of relational databases.

Keywords: Firebird PL/SQL Relational Model

Introducción

Los Sistemas de Gestión de Bases de Datos (SGBD)(Castaño Miguel et al., 2000) actuales proveen al desarrollador nuevas herramientas y tecnologías para la implementación de base de datos relacionales (BDR) (Silberschatz et at., 2002). El SGBD Firebird es la versión open source del SGBD Interbase de la empresa Borland/ Inprise®;[1] 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.(Chan and Yashkir, 2002). Estas funcionalidades nos permiten dos tipos de implementaciones extremas:

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 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 (Jackson, 2006) intermedia que lo aísla de

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

la implementación y podría considerarse como la API (application programming interface) (Blanchette, 2008) que exhibe hacia el exterior esta BDR y que otorga un mayor nivel de abstracción.[2]

Se proponen una serie de restricciones 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) (Hansen and Hansen, 1995)
- 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 disparador 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.

Para facilitar el análisis de casos y no extender este trabajo, se asume que todas las claves extranjeras (foreign key) (Elmasri and Navathe, 2010) se han implementado con su comportamiento por defecto y se otorgan todos los permisos sobre los disparadores y procedimientos acorde con el tipo de modelo a implementar.

Teniendo en cuenta lo indicado en los puntos anteriores, la programación de disparadores requiere de atributos adicionales para hacer conteos, sumatorias, etc.[3] 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 (Elmasri and Navathe, 2010) 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.

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

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

Revista Tecnología y Ciencia
ISSN 1666-6933

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

Desarrollo

Se han determinado los siguientes casos de implementación:

Implementación básica modelo B

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:[4]

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")[5]
- Crear procedimiento de selección SP_S{nombre tabla} que devuelva todas las tuplas de la tabla.

Dada el siguiente tabla de ejemplo:

libro(isbn,titulo)

el código a generar es el siguiente :

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

5 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.

```
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
```

Fig. 1: Implementación Básica Modelo B

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

Fig. 2: Implementación básica modelo B (Cont.)[6]

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.

6 En negritas se resaltan las partes del código PL/SQL específicas para esta tabla en particular.

Revista Tecnología y Ciencia
ISSN 1666-6933

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
```

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

EVENTO		ITEM
I	BEFORE	subotal=cantidad*precio
	AFTER	
U	**BEFORE**	**subotal=cantidad*precio**
	AFTER	
D	**BEFORE**	
	AFTER	

Fig. 3: Disparadores atributos redundantes modelo A y B

el código a generar es el siguiente:

```
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
```

Fig. 4: Código atributos redundantes modelo A y B

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		

Fig. 5: Disparadores tipo – subtipo mutuamente excluyentes

EVENTO		CONTRATISTA	EMPLEADO
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		

Fig. 6: Disparadores tipo – subtipo mutuamente excluyentes (Cont.)

el código a generar es el siguiente:

```
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
```

Fig. 7: Código tipo – subtipo mutuamente excluyentes

Revista Tecnología y Ciencia
ISSN 1666-6933

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.

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

Fig. 8: Disparadores cardinalidad máxima modelo A

el código a generar es el siguiente:

```
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
```

Fig. 9: Código cardinalidad máxima modelo A

```
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
```

Fig. 10: Código cardinalidad máxima modelo A (Cont.)

Revista Tecnología y Ciencia
ISSN 1666-6933

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

Fig. 11: Disparadores cardinalidad máxima modelo B

el código a generar es el siguiente:

```
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
```

Fig. 12: Código cardinalidad máxima modelo B

```
   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
```

Fig. 13: Código cardinalidad máxima modelo B (Cont.)

Entidad fuerte – Entidad débil con discriminante ascendente consecutivo

Dado el siguiente ejemplo:

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

en donde libro es una entidad fuerte (Elmasri and Navathe, 2010), 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.

7. http://www.gnu.org/licenses/lgpl.html
8. http://www.sourceforge.net
9. https://sourceforge.net/projects/fbjreplicator
10. En este momento se esta migrando a FB 3.0 para que ambas interfases gráficas funcionen con la misma versión de FB.

Revista Tecnología y Ciencia
ISSN 1666-6933

Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo A

Implementar un atributo que mantenga el último número 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

Fig. 14: Disparadores Entidad Fuerte – Entidad Débil con discriminante ascendente consecutivo, modelo A

EV.		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

Fig. 15: Disparadores Entidad Fuerte – Entidad Débil con discriminante ascendente consecutivo, modelo A (Cont.)

el código a generar es el siguiente:

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

```
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';
```

Fig. 16: Código Entidad Fuerte –
Entidad Débil con discriminante ascendente consecutivo, modelo A

```
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
```

Fig. 17: Código Entidad Fuerte – Entidad Débil con discriminante ascendente consecutivo, modelo A (Cont.)

Revista Tecnología y Ciencia
ISSN 1666-6933

Entidad fuerte – Entidad débil con discriminante ascendente consecutivo modelo B

El atributo último numero de ejemplar (ue) debe implementarse 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

Fig. 18: Disparadores Entidad Fuerte – Entidad Débil con discriminante ascendente consecutivo, modelo B

el código a generar es el siguiente:

```
CREATE EXCEPTION EX_ANY 'Error!';
CREATE TRIGGER TRG_BILIBRO FOR LIBRO
BEFORE INSERT AS
BEGIN
  NEW.UE=0;
END
```

Fig. 19: Código Entidad Fuerte – Entidad Débil con discriminante ascendente consecutivo, modelo A

```
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
```

Fig. 20: Código Entidad Fuerte – Entidad Débil con discriminante ascendente consecutivo, modelo B (Cont.)

Implementación automatizada de casos

Acorde con los requerimientos se propone el desarrollo de una aplicación cliente Java SE utilizando el framework Swing (Eckel, 2000) 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 propuesta anteriormente, se presta atención al código indicado en negritas, para cada caso a implementar se solicita el ingreso de datos apropiado y se genera el código PL/SQL a pedido del usuario

El software desarrollado se distribuye bajo licencia LGPL , 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 bajo el nombre de proyecto fbjreplicator ; este proyecto fue concebido como una herramienta de replicación para FB 2.5 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.

Revista Tecnología y Ciencia
ISSN 1666-6933

Resultados y Discusión

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.

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.

Conclusiones

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, en una implementación de tipo “modelo B” se podrían implementar reglas de integridad semánticas dentro de los procedimientos, aumentando su complejidad y realizando controles innecesarios y reiterados, en vez de hacerlo con el uso de disparadores junto con una correcta utilización de reglas de integridad no semánticas (Elmasri and Navathe, 2010) 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 ; 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.

Los casos genéricos que se expusieron son aplicables en distintos contextos de implementación, en donde es posible su implementación interactiva utilizando el software propuesto reduciendo tiempo de desarrollo y la probabilidad de error.

Revista Tecnología y Ciencia
ISSN 1666-6933

Referencias

Blanchette, J. (2008). *The Little Manual of API Design*, Trolltech, a Nokia company, June 19. Disponible en <http://www4.in.tum.de/~blanchet/api-design.pdf>.

Castaño Miguel, A. de, Velthuis Piattini, M., Martinez, E. M. (2000). *Diseño de Bases de Datos Relacionales.* Madrid: Ra-Ma, 5-7.

Chan, H., Yashkir, D. (2002). "Conceptual Architecture for InterBase/Firebird". Faculty of Mathematical, Statistical and Computer Sciences, University of Waterloo, Canada, January 29. Disponible en <http://www.ibphoenix.com/resources/documents/design/doc_25>

Eckel, B. (2000). *Thinking in Java.* Upper Saddle River: Prentice Hall: 689-790.

Elmasri, R., Navathe, S. B. (2010). *Sistemas de Bases de Datos, Conceptos Fundamentales.* Wilmington: Addison-Wesley Iberoamericana: 139-155.

Hansen, G. W., Hansen, J. V. (1995). *Diseño y Administración de Bases de Datos.* Upper Saddle River: Prentice Hall: 499-500.

Jackson, Daniel, Software Abstractions, MIT Press, (2006).

Silberschatz, A., Korth, H. F.; Sudarshan S. (2002). *Fundamentos de Bases de Datos.* Madrid: Mc Graw Hill.