Rev. Fac. Ing. Univ. Antioquia N.° 55 pp. 165-173. Septiembre, 2010

Detección y corrección de inconsistencias de referencias cíclicas en esquemas lógicos de bases de datos

Detection and correction of inconsistencies of cyclical references in database logical schemas

Carlos García, Abel Rodríguez, Norma Cabrera, Luisa González*

Departamento de Ciencia de la Computación, Universidad Central de Las Villas, Carretera a Camajuaní Km. 5½, Las Antillas, CP 54830, Cuba.

(Recibido el 15 de mayo de 2009. Aceptado el 9 de marzo de 2010)

Resumen

La transformación de esquemas conceptuales a esquemas lógicos llevada a cabo por herramientas de diseño de bases de datos puede traer como resultado esquemas lógicos que presenten algún tipo de inconsistencia y por consiguiente los esquemas físicos tendrán problemas de implementación. En este trabajo se analiza un tipo de inconsistencia (nombrada por los autores como "inconsistencia de referencias cíclicas") que puede presentarse en el esquema lógico de la base de datos y se propone un algoritmo para su detección y corrección. Este algoritmo puede ser implementado en herramientas de diseño de bases de datos.

----- *Palabras clave:* modelo entidad-relación, esquema conceptual, esquema lógico, inconsistencias en bases de datos, restricciones de integridad referencial

Abstract

The transformations of conceptual schemes into logical schemes carried out by database design tools may result in logical schemes that present some kind of inconsistence and therefore the physical schemes will have implementation problems. This work presents an algorithm that allows detecting and correcting these inconsistencies in the logical schema. This algorithm can be implemented in a database design tool.

---- *Keywords:* entity-relationship model, conceptual schema, logical schema, database inconsistencies, referential integrity constraints

* Autor de correspondencia: teléfono: + 53 + 42 + 281 515, correo electrónico: cgarcia@uclv.edu.cu (C. García)

Introducción

El diseño de una base de datos es un proceso, que a pesar de la experiencia del diseñador y del empleo de herramientas CASE (*Computer Aided Software Engineering*), no está exento de errores e insuficiencias, lo que puede comprometer la calidad de los esquemas generados, en particular del esquema físico. Por tanto es deseable detectar estos errores e insuficiencias en las fases iniciales del diseño. En este sentido se han publicado numerosos trabajos cuyo objetivo es detectar inconsistencias en esquemas conceptuales basados en las restricciones de cardinalidad. Algunos de estos métodos sólo deciden si el conjunto de restricciones asociado al esquema conceptual puede ser satisfecho o no [1, 2], otros métodos permiten determinar los conjuntos de restricciones de cardinalidad que hacen que el esquema conceptual sea inconsistente [3-10] y en algunos casos se propone al diseñador un plan de posibles correcciones a realizar en el esquema conceptual [11, 12]. Con relación a la detección de inconsistencias en esquemas lógicos, la producción científica no ha sido tan numerosa y ha estado enfocada principalmente al empleo de la teoría de la normalización [13-17] para prevenir inconsistencias en los datos, de modo que la base de datos pueda normalizarse hasta el grado deseado, garantizándose al menos que no ocurran anomalías de actualización [18].

Como resultado del estudio de los métodos de detección de inconsistencias en esquemas conceptuales, se pudo comprobar con el empleo de casos de estudio, que al transformar esquemas conceptuales consistentes era posible obtener un esquema lógico con esquemas de relación que presentaban un tipo de inconsistencia que los autores de este trabajo han nombrado como "inconsistencia de referencias cíclicas" las que provocan problemas en el momento de la inserción de los datos en la base de datos. La principal contribución de este artículo es presentar un algoritmo para validar esquemas lógicos, que permite detectar y corregir inconsistencias de referencias cíclicas que se presenten a este nivel y que puede ser implementado en una herramienta CASE de diseño de bases de datos.

Inconsistencias en el esquema lógico de una base de datos

Para introducir el problema del tipo de inconsistencia que trata este artículo, se mostrarán varios ejemplos y se hará una caracterización del caso más general. Los diagramas Entidad-Relación que se muestran utilizan la notación de Elmasri [18], en la cual el rectángulo representa los conjuntos de entidades y el rombo a los conjuntos de interrelaciones. La cardinalidad máxima de una interrelación se indica con un "1" o con la letra "M". Para la cardinalidad mínima una línea simple indica participación opcional y una línea doble participación obligatoria de las entidades en la interrelación. En la notación de Elmasri la cardinalidad máxima y la cardinalidad mínima se expresan mediante la notación *look across* y *look here* [19] respectivamente. La transformación del esquema conceptual al esquema lógico se hace siguiendo las reglas de transformación de Elmasri [18], aunque se pudieran haber aplicado otras como las propuestas por Teorey [20], Jajodia [21], De Miguel [22] o Ponniah [23].

El diagrama Entidad-Relación (ER) de la figura 1 modela el hecho de que un empleado tiene que ser supervisado por otro empleado y que un empleado puede supervisar a ninguno, uno o varios empleados.

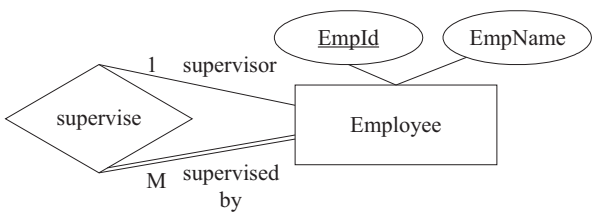

Figura 1 Diagrama ER con una interrelación recursiva

Al transformar el esquema conceptual se obtiene el siguiente esquema de relación, en el cual como convenio se adoptará que las llaves primarias

estarán subrayadas y las llaves extranjeras estarán en itálica:

EMPLOYEE (EMPID, EMPNAME, *SUPERVISOR*)

En el esquema anterior la llave extranjera SUPERVISOR no permite valores nulos debido a la participación obligatoria del conjunto entidad EMPLOYEE en la interrelación supervise. Al generar el código SQL correspondiente al esquema de relación EMPLOYEE se obtiene lo siguiente:

```
CREATE TABLE EMPLOYEE(
    EMPID          INTEGER NOT NULL,
    EMPNAME        CHAR(30),
    SUPERVISOR     INTEGER NOT NULL,
    CONSTRAINT employee_pk1 PRIMARY KEY(EMPID),
    CONSTRAINT employee_fk1 FOREIGN KEY(SUPERVISOR)
        REFERENCES EMPLOYEE (EMPID));
```

Como puede observarse no es posible insertar un primer empleado sin antes haber insertado al empleado supervisor, por lo que se presenta un problema de implementación y no es posible insertar una fila en la tabla EMPLOYEE. Es importante destacar que el problema de implementación expuesto es consecuencia de un esquema de relación inconsistente.

En el diagrama de la figura 2 se modelan los hechos de que los empleados tienen que pertenecer a un departamento y que un departamento tiene que ser dirigido por un empleado.

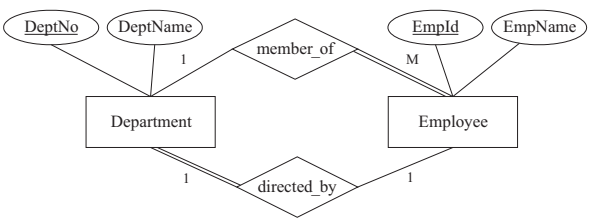

Figura 2 Diagrama ER que forma un ciclo entre dos conjuntos de entidades

Al transformar el esquema conceptual se obtienen los siguientes esquemas de relación:

EMPLOYEE(EMPID, EMPNAME, *DEPTNO*)
DEPARTMENT(DEPTNO, DEPTNAME, *DIRECTOR*)

En este caso, debido a la participación obligatoria de los conjuntos entidad DEPARTMENT y EMPLOYEE en las interrelaciones DIRECTED_BY y MEMBER_OF respectivamente, las llaves extranjeras correspondientes no permiten valores nulos, lo cual se expresa a través del siguiente código SQL:

```
CREATE TABLE EMPLOYEE(
    EMPID          INTEGER NOT NULL,
    EMPNAME        CHAR(30),
    DEPTNO         INTEGER NOT NULL,
    CONSTRAINT employee_pk1 PRIMARY KEY (EMPID),
    CONSTRAINT employee_fk1 FOREIGN KEY (DEPTNO)
        REFERENCES DEPARTMENT (DEPTNO));
CREATE TABLE DEPARTMENT(
    DEPTNO         INTEGER NOT NULL,
    DEPTNAME       CHAR(30),
    DIRECTOR       INTEGER NOT NULL UNIQUE,
    CONSTRAINT department_pk1 PRIMARY KEY (DEPTNO),
    CONSTRAINT department_fk1 FOREIGN KEY (DIRECTOR)
        REFERENCES EMPLOYEE(EMPID));
```

Con la restricción de integridad NOT NULL sobre el atributo EMPID en la tabla DEPARTMENT, no es posible insertar un departamento a menos que se haya insertado el empleado jefe en la tabla EMPLOYEE; por otro lado con la restricción de integridad NOT NULL sobre el atributo DEPTNO en la tabla EMPLOYEE, no es posible insertar un empleado a menos que se haya insertado el

departamento al que pertenece el mismo. Como puede observarse hay una dependencia mutua que genera un conflicto que no permite la inserción de tuplas en ninguno de los dos esquemas. Este tipo de inconsistencia en los esquemas de relación genera un problema de implementación dado por la presencia de llaves extranjeras con referencias mutuas y que no admiten valores nulos.

En el diagrama de la figura 3 se modelan los siguientes hechos: los empleados tienen que pertenecer a un departamento; un departamento tiene que financiar un proyecto y un proyecto tiene que ser apoyado por un empleado.

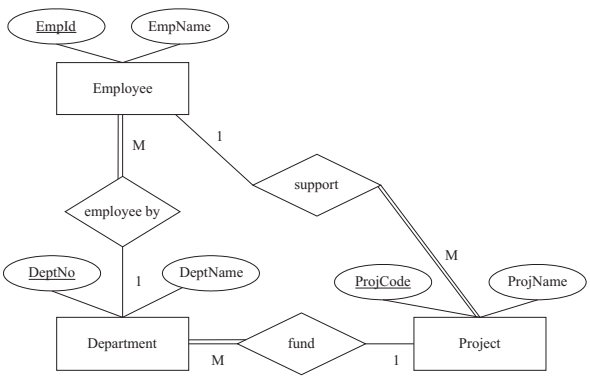

Figura 3 Diagrama ER que forma un ciclo entre tres conjuntos de entidades

Transformando el esquema conceptual se obtienen los siguientes esquemas de relaciones:

EMPLOYEE(EPID, EMPNAME, *DEPTNO*)

DEPARTMENT(DEPTNO, DEPTNAME, *PROJCODE*)

PROJECT(PROJCODE, PROJNAME, *EMPID*)

Nótese que las llaves extranjeras de los esquemas obtenidos no admiten valores nulos debido a la participación obligatoria de los conjuntos de entidades del lado "mucho" de la interrelación.

```
CREATE TABLE EMPLOYEE(
    EMPID          INTEGER NOT NULL,
    EMPNAME        CHAR(30),
    DEPTNO         INTEGER NOT NULL,
    CONSTRAINT employee_pk1 PRIMARY KEY (EMPID),
    CONSTRAINT employee_fk1 FOREIGN KEY (DEPTNO)
        REFERENCES DEPARTMENT(DEPTNO));
CREATE TABLE DEPARTMENT(
    DEPTNO         INTEGER NOT NULL,
    DEPTNAME       CHAR(30),
    PROJCODE       INTEGER NOT NULL,
    CONSTRAINT department_pk1 PRIMARY KEY (DEPTNO),
    CONSTRAINT department_fk1 FOREIGN KEY (PROJCODE)
        REFERENCES PROJECT (PROJCODE));
CREATE TABLE PROJECT(
    PROJCODE       INTEGER NOT NULL,
    PROJNAME       CHAR(30),
    EMPID          INTEGER NOT NULL,
    CONSTRAINT project_pk1 PRIMARY KEY (PROJCODE),
    CONSTRAINT project_fk1 FOREIGN KEY (EMPID)
        REFERENCES EMPLOYEE (EMPID));
```

Como se puede notar en el código SQL anterior, las llaves extranjeras de los esquemas tienen la restricción NOT NULL y al mismo tiempo existe una referencia cíclica entre las tablas, de manera que no es posible insertar una fila en ninguna de las tres tablas, por lo que también se presenta un problema de implementación debido a esquemas de relaciones inconsistentes.

Los ejemplos mostrados anteriormente permiten concluir que a pesar de que un diagrama Entidad-Relación sea estructuralmente válido este pudiera generar un esquema lógico con inconsistencias de referencias cíclicas en dependencia del conjunto de reglas de transformación que se le apliquen a los esquemas conceptuales.

Caracterización de la inconsistencia de referencias cíclicas

Sean R_1, R_2,…, R_n esquemas de relaciones de un esquema lógico L, la inconsistencia de referencias cíclicas se presenta en el esquema lógico de la base de datos si existe una llave extranjera en R_i que no admite valor nulo y que hace referencia a una llave primaria en R_{i+1}, para i=1,…,n-1; y también existe una llave extranjera en R_n que no admite valor nulo y que hace referencia a una llave primaria en R_1, donde R_n y R_1 no son necesariamente diferentes.

De esta caracterización se puede concluir que este tipo de inconsistencia se puede presentar en el esquema lógico cuando existen referencias cíclicas entre esquemas de relación. Estas referencias pueden ser entre un mismo esquema, entre dos esquemas o entre *n* esquemas.

Algunas soluciones desde el punto de vista de implementación y de las facilidades aportadas por algunos Sistemas de Gestión de Bases de Datos Relacionales (SGBDR) pueden ser:

1. CODASYL [24], por ejemplo, proporciona una solución para las referencias cíclicas entre esquemas.
2. Un esquema de relación con una referencia a si mismo puede permitir la inserción de filas si se asume que la primera fila tenga una referencia a si misma.
3. Las referencias cíclicas entre dos o más esquemas pueden ser resueltas a nivel del lenguaje SQL mediante transacciones donde una restricción de integridad referencial puede ser desactivada hasta que se haya completado la transacción. Esto se conoce como el modo SQL diferido (*SQL deferred mode*) que está incluido en el estándar SQL-1992 [25].

La solución 1 no está presente en la mayoría de los SGBDR comerciales. La solución 2 para las referencias cíclicas a un mismo esquema es posible y habrá que tener en cuenta que la inserción de la primera fila debe tener una referencia a si misma. Aunque la solución 3 es una facilidad incluida en el estándar SQL-92, el modo SQL diferido no se encuentra implementado en todos los SGBDR.

A partir de este análisis, los autores de este artículo proponen una solución que es independiente del SGBDR a utilizar en la implementación de la base de datos. En este caso se consideraron dos alternativas: realizar la detección de inconsistencias en el esquema conceptual o realizarla en el esquema lógico.

Al valorar la detección de este tipo de inconsistencia en el esquema conceptual, se pudo comprobar que incluso ante la presencia de esquemas conceptuales estructuralmente válidos, es posible obtener un esquema lógico de la base de datos con inconsistencias de referencias cíclicas y lo que algunas ocasiones también depende del conjunto de reglas de transformación se que apliquen para obtener el esquema lógico. Por esta razón la propuesta está basada en la segunda alternativa, es decir, detectar el tipo de inconsistencia en el esquema lógico. La solución también incluye la corrección de las inconsistencias tratando de mantener la semántica de las interrelaciones expresadas en el esquema conceptual. La corrección consiste modificar y crear esquemas de relación que eliminen la inconsistencia de referencias cíclicas, lo cual será explicado a continuación.

Considere de nuevo el ejemplo de la figura 2 y el esquema lógico obtenido:

EMPLOYEE(<u>EMPID</u>, EMPNAME, *DEPTNO*)
DEPARTMENT(<u>DEPTNO</u>, DEPTNAME, *DIRECTOR*)

La inconsistencia de este esquema es posible comprobarla a través del análisis de los estados S_1 y S_2: S_1= { $r_{EMPLOYEE}$ = [e1, e1name, d1], $r_{DEPARTMENT}$ = [d1, d1name, e1] }; S_2 = { $r_{EMPLOYEE}$ = [e2, e2name, null], $r_{DEPARTMENT}$ = [d2, d2name, e2] }. El estado S_1 no es posible satisfacerlo debido a las referencias mutuas de las llaves extranjeras de los esquemas EMPLOYEE y DEPARTMENT. El estado S_2 tampoco puede ser satisfecho debido a que la llave extranjera del esquema EMPLOYEE no admite valores nulos.

Para eliminar esta inconsistencia se propone crear un nuevo esquema de relación en lugar de una de las restricciones de integridad referencial, por ejemplo se seleccionará la llave extranjera que representa a la interrelación MEMBER_OF y en su lugar se creará un nuevo esquema con el mismo nombre, de manera que el esquema lógico resultante sería:

EMPLOYEE(EMPID, EMPNAME)

DEPARTMENT(DEPTNO, DEPTNAME, *DIRECTOR*)

MEMBER_OF(EMPID, DEPTNO)

Considere ahora el siguiente estado: $S_1 = \{r_{EMPLOYEE} = [e1, e1name], r_{DEPARTMENT} = [d1, d1name, e1], r_{MEMBER\_OF} = [e1, d1]\}$. Como puede observarse este estado es consistente y se logra insertando primero una tupla en el esquema EMPLOYEE y luego insertando una tupla en el esquema DEPARTMENT. Una vez que se tenga al menos una tupla en los esquemas EMPLOYEE y DEPARTMENT entonces es posible insertar una tupla en el esquema MEMBER_OF.

La solución planteada evita el tipo de inconsistencia objeto de estudio en este artículo, a la vez que permite reducir el número de nuevos esquemas de relación a generar y también reducir el número de tuplas con valores nulos.

Detección de inconsistencias de referencias cíclicas en esquemas lógicos

En esta sección se describe el algoritmo de detección y corrección de inconsistencias de referencias cíclicas en esquemas lógicos. El algoritmo utiliza un multigrafo dirigido para representar el conjunto de esquemas de relaciones generados a partir del esquema conceptual. El algoritmo selecciona una solución que respeta el criterio de tener la menor cantidad de esquemas posibles, al generar esquemas independientes con el objetivo de eliminar las inconsistencias, y de esta manera representar más adecuadamente la semántica del problema modelado.

Representación de un esquema lógico

Como representación del esquema lógico se propone como estructura de datos un multigrafo dirigido. Según [26] un grafo dirigido G = (V, A, f) consta de un conjunto no vacío V denominado conjunto de vértices del grafo, un conjunto A de aristas del grafo y una correspondencia f del conjunto de aristas A en un conjunto de pares ordenados de V. La función f se define como sigue: $f : A \rightarrow VxV$; $e \varepsilon A$; $f(e) = (u,v)$ donde $u,v \varepsilon V$.

La representación de un esquema lógico de una base de datos mediante un multigrafo dirigido es directa si se asume que los vértices son los esquemas de relaciones y las aristas dirigidas indican la relación que existe entre la llave extranjera de un esquema de relación con su correspondiente llave primaria en el otro esquema de relación.

Sea G_{EL}=(V,A, f) un multigrafo dirigido para la representación del esquema lógico de una base de datos, donde:

- V es el conjunto de vértices que representan los esquemas de relación generados a partir del esquema conceptual.
- A es el conjunto de aristas del grafo que se construyen a partir de la función *f*.
- f(e)=(u,v) la arista e representa que el esquema de relación asociado al vértice u tiene una llave extranjera cuya llave primaria corresponde al esquema de relación asociado al vértice v, donde u y v son no necesariamente diferentes.
- La arista *e* tiene pesos representados por *FK_constraint* y *Involvement*, cuyos posibles valores son:

$$FK\_constraint = \begin{cases} IsNotNull \text{ si la llave extranjera no admite el valor nulo} \\ IsNull \text{ en el caso contrario} \end{cases}$$

Involvement: Contiene un valor entero mayor o igual a cero que indica la cantidad de veces que la arista *e* participa en algún ciclo simple de longitud *n*.

Algoritmo de detección de inconsistencias

La representación del esquema lógico mediante un grafo tiene como ventaja que permite el uso de estructuras de datos y algoritmos conocidos en la literatura.

Algoritmo:

PASO 1: Creación del grafo G_{EL} que representa al esquema lógico.

PASO 2: Detección de inconsistencias en el grafo G_{EL}.

PASO 3: Corrección de las inconsistencias en el grafo G_{EL}.

A continuación se describe cada paso del algoritmo propuesto.

PASO 1. El grafo G_{EL} es creado cuando se realiza el proceso de transformación del esquema conceptual al esquema lógico. Para cada arista se establece el valor del peso *FK_constraint* indicando si la llave extranjera admite o no el valor nulo.

PASO 2. Se buscan todos los ciclos de longitud n, actualizando el peso *Involvement* de las aristas. Para cada uno de los ciclos encontrados, si todas las aristas que forman parte del ciclo tienen el valor del peso *FK_constraint* indicando que no admiten el valor nulo, entonces se marca ese ciclo como inconsistente.

PASO 3. El proceso de corrección distingue entre los lazos o bucles y los ciclos simples de longitud mayor o igual que dos que han sido marcados como inconsistentes.

> PASO 3.1. Para cada arista *e* de un vértice *u* que sea un lazo se hace lo siguiente: eliminar del vértice *u* los atributos que forman la llave extranjera y que hacen referencia al propio esquema a través de la arista *e*; crear un nuevo vértice *w* con aristas *f(e')=(w,u) y f(e")= (w,u)*; insertar en el esquema *w* una llave extranjera por cada arista *e' y e"* que haga referencia a la llave primaria del esquema *u* y definir una de las dos llaves extranjeras como llave primaria del esquema *w*; actualizar los pesos de las aristas *e'* y *e"*; eliminar la arista *e*.
>
> PASO 3.2. Para todos los ciclos simples de longitud mayor o igual que dos hacer lo siguiente: buscar una arista *e* con valor máximo en el peso *Involvement* y eliminar del vértice *u* la llave extranjera que hace referencia al vértice *v* a través de la arista *e*; crear un nuevo vértice *w* con aristas *f(e')=(w,u) y f(e")=(w,v)*; insertar en el esquema *w* dos llaves extranjeras, una llave extranjera que hará referencia a través de la arista *e'* a la llave primaria del vértice *u* y la otra hará referencia a través de la arista *e"* a la llave primaria del vértice *v*; definir como llave primaria a la llave extranjera que hace referencia al vértice *u*; actualizar los pesos de las aristas *e'* y *e"*; eliminar la arista *e* y disminuir en uno el valor del peso *Involvement* para aquellas aristas que formaban parte del ciclo con la arista *e*. Repetir este paso mientras existan aristas con peso *Involvement* mayor o igual que cero.

Como resultado de la aplicación del algoritmo descrito se obtiene un nuevo esquema lógico que no presenta inconsistencias de referencias cíclicas, el cual puede ser entonces traducido al esquema físico de la base de datos donde se a llevar cabo la implementación de la misma.

Un ejemplo de aplicación del algoritmo

A partir del diagrama Entidad-Relación que aparece en la figura 4, se mostrará un esquema lógico con inconsistencias y a continuación el esquema lógico que se obtiene cuando se aplica el algoritmo descrito anteriormente.

Al transformar el diagrama Entidad-Relación de la figura 4 se obtiene el siguiente esquema lógico:

PROFESSOR(PROFID, PROFNAME, *DEPTNO*)

DEPARTMENT(DEPTNO, DEPTNAME, *MANAGES_PROFID*, *SUPPORTS_PROFID*)

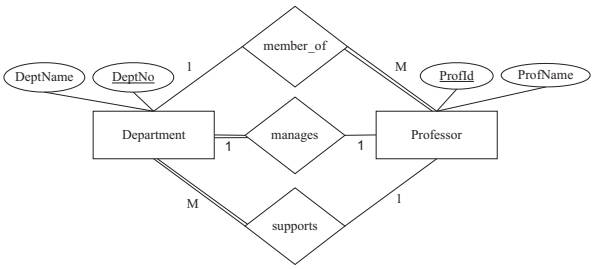

Figura 4 Diagrama ER del ejemplo

En la figura 5 se muestra la representación de este esquema en el grafo G_{EL}.

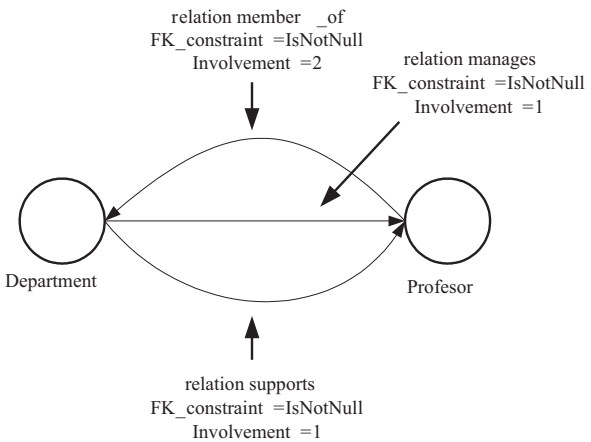

Figura 5 Grafo G_{EL} representando esquemas lógicos con inconsistencias

Como puede observarse, los ciclos de longitud 2 entre los vértices DEPARTMENT y PROFESSOR que presentan inconsistencias de referencias cíclicas son:

PROFESSOR → MEMBER_OF → DEPARTMENT → MANAGES → PROFESSOR

PROFESSOR → MEMBER_OF → DEPARTMENT → SUPPORTS → PROFESSOR

Note que la arista MEMBER_OF participa en dos ciclos por lo que el valor del peso *Involvement* es 2, y tiene el valor mayor en comparación con las otras aristas del grafo, por lo que según el algoritmo explicado esta arista se elimina y en su lugar crea un nuevo vértice llamado MEMBER_OF: donde una arista se dirigirá hacia el vértice PROFESSOR y la otra arista hacia el vértice DEPARTMENT. La figura 6 muestra una solución luego de la aplicación del algoritmo descrito.

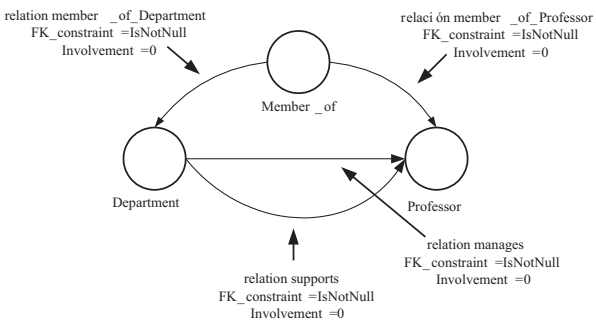

Figura 6 Grafo G_{EL} representando esquemas lógicos sin inconsistencias de referencias cíclicas

Observe que este grafo no tiene ciclos de longitud 2 que estén en conflicto. A partir de este grafo se puede obtener el siguiente esquema lógico de la base de datos sin inconsistencias.

PROFESOR(PROFID, PROFNAME)

DEPARTMENT(DEPTNO, DEPTNAME, *MANAGES_PROFID*, *SUPPORTS_PROFID*)

MEMBER_OF(PROFID, *DEPTNO*)

Conclusiones

En este artículo se ha caracterizado un tipo de inconsistencia que se puede presentar en el esquema lógico de una base de datos. Como principal contribución se ha presentado un algoritmo para la detección y corrección de inconsistencias de referencias cíclicas en el esquema lógico. La solución propuesta es independiente del Sistema de Gestión de Bases de Datos Relacionales a utilizar en la implementación de la base de datos. Como trabajo futuro está la implementación del algoritmo en la herramienta ERECASE que ha sido desarrollada por los autores de este artículo.

Referencias

1. D. Calvanese, M. Lenzerini. "On the Interaction Between ISA and Cardinality Constraints," *Tenth International Conference on Data Engineering*. Houston (TX). 1994. pp. 204-213.
2. M. Lenzerini, P. Nobili. "On the satisfiability of dependency constraints in entity–relationship schemata." *Information Systems.*Vol. 15. 1990. pp. 453-461.
3. J. Dullea, I. Y. Song. "An Analysis of Structural Validity of Ternary Relationships in Entity Relationship Modeling." *7th International Conference on Information and Knowledge Management (CIKM '98)*. Washington. 1998. pp. 331-339.
4. J. Dullea, I. Y. Song. "An Analysis of the Structural Validity of Unary and Binary Relationships in Entity Relationship Modeling." *4th International Conference on Computer Science and Informatics.* Research Triangle Park. North Carolina. Vol. 3. 1998. pp. 329-334.
5. J. Dullea, I. Y. Song. "A Taxonomy of Recursive Relationships and Their Structural Validity in ER Modeling." *ER 1999*. Paris. Vol. LNCS 1728. 1999. pp. 384-398.
6. J. Dullea, I. Y. Song, I. Lamprou. "An Analysis of Structural Validity in Entity-Eelationship Modeling". *Data & Knowledge Engineering.* Vol. 47. 2003. pp. 167-205.
7. S. Hartmann. "On the Consistency of Int-cardinality Constraints." *ER 1998*. Singapore. Vol. LNCS 1507. 1998. pp. 150-163.
8. S. Hartmann. "On Interactions of Cardinality Constraints, Keys and Functional Dependencies." *FoIKS 2000*. Vol. LNCS 1762. Burg (Germany). 2000. pp. 136-155.
9. S. Hartmann. "On the implication problem for cardinality constraints and functional dependencies." *Annals of Mathematics and Artificial Intelligence.* Vol. 33. 2001. pp. 253-307.
10. B. Thalheim. "Fundamentals of cardinality constraints." *ER 1992*. Karlsruhe (Germany). Vol. LNCS 645. 1992. pp. 7-23.
11. S. Hartmann. "Coping with Inconsistent Constraint Specifications." *ER 2001*. Yokohama (Japón). Vol. LNCS 2224. 2001. pp. 241-255.
12. S. Hartmann. "Soft Constraints and Heuristic Constraint Correction in Entity-Relationship Modelling." *International workshop on semantics in databases*. Dagstuhl Castle (Germany). Vol. LNCS 2582. 2003.pp. 82-99.
13. E. F. Codd. "Further Normalization of the Data Base Relational Model." *Database Systems*. Ed. Prentice-Hall. Englewood Cliffs (NJ). 1972. pp. 200-216.
14. E. F. Codd. "Recent Investigations in Relational Database Systems," *IFIP Congress*. Amsterdam. 1974. pp. 1017-1021.
15. D. Maier. *The Theory of Relational Databases*. Ed. Computer Science Press. Rockville (MD). 1983. pp. 269-283.
16. J. D. Ullman. *Principles of Database and Knowledge-Base Systems* . Ed. Computer Science Press. New York. Vol. 1. 1988. pp. 121-123.
17. E. F. Codd. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM.* Vol. 13. 1970. pp. 377-387.
18. R. Elmasri, S. B. Navathe. *Fundamentals of Database Systems*.5th ed. Ed. Addison-Wesley. Boston (MA). 2007. pp. 49-361.
19. I.Y. Song, M. Evans, E. K. Park. "A comparative analysis of entity-relationship diagrams." *Journal of Computer and Software Engineering.*Vol. 3. 1995. pp. 427-459.
20. T. Teorey, D. Yang, J. Fry. "A logical design methodology for relational databases using the extended E-R model." *ACM Computing Surveys.* Vol. 18. 1986. pp. 197-222.
21. S. Jajodia, P. Ng, F. Springsteel. "The problem of equivalence for entity-relationship diagrams." *IEEE Transactions on Software Engineer.* Vol. SE-9. 1983. pp. 617-630.
22. A. De Miguel, M. Piattini, E. Marcos. *Diseño de Bases de Datos Relacionales*. RA-MA. Madrid. 1999. pp. 313-366.
23. P. Ponniah. *Database Design and Development*. Ed. Wiley Interscience. New York. 2003. pp. 274-298.
24. G. T. Fadok. *Effective design of CODASYL database*. Ed. Collier Macmillan. Basingstoke (UK). 1985. pp. 90-140.
25. ANSI/ISO/IEC. "ISO/IEC 9075:1992. Database Language SQL." *ANSI/ISO/IEC International Standard*. Maynard (MA). 1992. pp19-422.
26. W. K. Grassmann J. P. Tremblay. *Matemática Discreta y Lógica. Una perspectiva desde la Ciencia de la Computación*. Ed Prentice Hall. Madrid. 1997. pp. 392-403.