Optimización Semántica de Consultas: Un Caso Práctico

Leopoldo Bertossi
Universidad Católica de Chile, Escuela de Ingeniería
Departamento de Ciencia de Computación
E-MAIL: BERTOSSI@ING.PUC.CL

Cecilia Bastarrica
University of Connecticut, School of Engineering
Department of Computer Science and Engineering
E-MAIL: CECILIA@ENG2.UCONN.EDU

Ricardo Valdivia
Universidad Católica de Chile, Escuela de Ingeniería
Departamento de Ciencia de Computación
E-MAIL: REVALDIV@ING.PUC.CL

Resumen

Presentamos un caso real de aplicación de optimización semántica de consultas en una base de datos relacional que se realizó con el propósito de conseguir ciertas mejoras en su rendimiento. A partir de ideas expuestas en [3] para la construcción de un optimizador semántico, se muestra una mejora en el proceso de validación de restricciones de integridad por medio de *triggers,* lo que constituye un uso novedoso de este método. Además, se reconstruye la aplicación desde un enfoque deductivo de la optimización semántica de consultas.

1. Introducción

El concepto de optimización semántica de consultas (SQO) fue introducido en forma independiente por King [6] y por Hammer y Zdonik [5]. En contraste con la optimización tradicional (que utiliza métodos de transformación sintáctica, además, de información estadística y el uso de las estructuras de datos subyacentes), la SQO usa conocimiento basado en la semántica de los datos con el propósito de transformar la consulta original en una nueva consulta que produce la misma respuesta, pero que puede ser procesada en forma más eficiente. El conocimiento semántico sobre el cual se basan estas transformaciones está representado por reglas de integridad, que son usadas para asegurar la validez de las transacciones en bases de datos.

Aunque la optimización semántica de consultas tiene una sólida base teórica, no siempre es fácil comprobar empíricamente sus beneficios prácticos, especialmente por no existir aún optimizadores semánticos de uso comercial. En este artículo se describe un caso en el cual se aplicaron los principios de la SQO sobre una base de datos real, consiguiendo mejoras notables en el rendimiento de un procedimento de aplicación masiva.

En la sección 2 se formula el problema práctico y su solución directa. En la sección 3 se presenta una forma optimizada de resolver el problema sobre la base de ideas provenientes del área de optimización semántica de consultas. En la sección 4 se repasa algunos conceptos básicos de bases de datos deductivos y de optimización semántica de consultas en ese contexto, que es aquél en el cual de ha dado una formulación precisa a esa metodología. En la sección 5 se reconstruye la solución optimizada presentada en la seccion 3 desde el punto de vista de optimización semántica de consultas en bases de datos deductivas. Finalmente, en la sección 6, se extrae algunas conclusiones.

2. El Problema

El Sistema de Estadísticas es uno de los componentes de un conjunto de sistemas que han sido desarrollados tanto para el Ministerio de Salud como para los Servicios de Salud Pública de Chile. Su función principal es el registro de los servicios prestados dentro de su ámbito de interés (los Servicios de Salud para el Ministerio y los Establecimientos Hospitalarios para los Servicios de Salud), para permitir posteriormente obtener información estadística acerca de estas prestaciones.

El núcleo del Sistema de Estadística lo conforma una base de datos relacional con aproximadamente 150 tablas, implementada en INFORMIX corriendo en un equipo IBM RS6000 con sistema operativo AIX.

La optimización semántica se aplicó, como ejemplo, a una parte muy pequeña de la base de datos, que contiene dos tablas con los siguientes esquemas:

- $Enfermedad(tipo, codigo)$
- $EgresoHospitalario(ident, codigoBasico, codigoSecundario, codigoExterno)$

Descripción:

1. La tabla *Enfermedad* contiene los códigos de enfermedad de la OMS, donde el *tipo* es un calificativo del origen de la enfermedad: el valor "B" se refiere a una causa básica de enfermedad (por ejemplo, un infarto) y "E" es causa externa (por ejemplo, atropellamiento). Ambos atributos, *tipo* y *codigo*, conforman la clave de la tabla.

2. *ident* es la clave de la tabla *EgresoHospitalario*, incluye la identificaciones del establecimiento hospitalario y de la persona, y la fecha de egreso.

3. Los atributos *codigoBasico*, *codigoSecundario* y *codigoExterno* corresponden a códigos de enfermedad, tal como *codigo* en la tabla *Enfermedad*. Los dos primeros corresponden a causas básicas y el último es una causa externa. Sólo *codigoBasico* es necesariamente no nulo, pues debe haber alguna causa para la hospitalización.

Se tiene una tabla muy grande de *EgresoHospitalario*, con aproximadamente 1.500.000 registros anuales. Cada Establecimiento informa mensualmente sus egresos hospitalarios.

Por otro lado, la tabla *Enfermedad*, que es prácticamente estática, contiene alrededor de 68.000 registros. En consecuencia, buscar por un campo que no es clave es un proceso lento. Los esquemas anteriores son versiones simplificadas de la base de datos real, en la cual las tablas tienen muchos más campos.

Una de las principales transacciones consiste en el ingreso de datos aportados por los Establecimientos Hospitalarios en *EgresoHospitalario*. Los datos a insertar en la tabla *EgresoHospitalario* vienen almacenados en archivos ASCII a razón de una línea por registro. La información se reordena y se inserta en la tabla en forma de tupla de acuerdo con la definición hecha en la base de datos.

Sin embargo, dado que los datos provienen de fuentes externas, deben someterse a controles de integridad sintáctica y semántica. Estos controles son de diferentes tipos:

1. definición de los tipos de cada uno de los atributos de la tabla,

2. definición de restricciones de integridad referenciales, y

3. definición de *triggers* de inserción que invocan procedimientos que verifican las condiciones deseadas.

Las dos primeras formas de control se definen conjuntamente con la base de datos, de modo que es posible usar las optimizaciones definidas por el sistema administrador de base de datos en el chequeo de estas condiciones. Sin embargo, la tercera opción, o sea, los procedimientos de control, deben ser programados. Se debe tener extremo cuidado en las consideraciones de eficiencia en la ejecución, sobre todo cuando el número de registros a ser ingresados es grande, como en nuestro caso. Cuando los *triggers* incluyen la consulta a otras tablas, el procesamiento se suele hacer muy lento.

En general, los sistemas administradores de bases de datos no permiten la definición de claves foráneas sobre atributos potencialmente nulos; y aunque lo permitiesen, no se considera una buena práctica. Por ello, los atributos potencialmente nulos que hacen referencia a otras tablas, deben ser chequedados mediante consultas invocadas por un *trigger* de inserción.

En una primera instancia, la tabla *EgresoHospitalario* se definió conteniendo solamente los datos informados desde los establecimientos, entre éstos, los códigos de enfermedad básico, secundario y externo, pero en ninguno de los tres casos se indicaba el *tipo* de la enfermedad. Esto producía como consecuencia que, aun en el caso del *codigoBasico*, que está definido como no nulo, no pudiera definirse en *EgresoHospitalario* una clave foránea a la tabla *Enfermedad* (esto tendría la ventaja de mejorar la búsqueda en la tabla *Enfermedad*). Por lo tanto, el *trigger* de inserción debía comprobar mediante tres consultas a la tabla *Enfermedad* la validez del código de enfermedad informado (se utilizaron tres consultas independientes para determinar la causa exacta del probable error), teniendo aún como agravante el hecho de estar consultando solamente por el *código*, es decir, solamente por parte de la clave. Por otra parte, y como atenuante de la lentitud, las consultas por los códigos de enfermedad secundaria o externa solamente se ejecutaban en el caso de no ser valores nulos.

El siguiente código SQL asociado al *trigger* de inserción, describe la situación planteada. La primera consulta SELECT valida la existencia del *codigoBasico* en la tabla *Enfermedad*, la segunda, la existencia del *codigoSecundario*, y la tercera, la existencia del *codigoExterno*.

```
LET existe = 0;

SELECT count(*) INTO existe
FROM   Enfermedad
WHERE codigo = codigoBasico;
IF existe = 0 THEN
        RAISE EXCEPTION -32762;


LET existe = 0;

IF codigoSecundario != "" AND codigoSecundario != "0000" THEN
        SELECT count(*) INTO existe
        FROM   Enfermedad
        WHERE codigo = codigoSecundario
        IF existe = 0 THEN
                RAISE EXCEPTION -32761;
        END IF;
```

```
END IF;


LET existe = 0;

IF codigoExterno != "" AND codigoExterno != "0000" THEN
        SELECT count(*) INTO existe
        FROM    Enfermedad
        WHERE codigo = codigoExterno;
        IF existe = 0 THEN
                RAISE EXCEPTION -32760;
        END IF;
END IF;
```

Los resultados que se obtuvieron con los procedimientos definidos anteriormente fueron de alrededor de 4 horas para insertar 1.500 registros de egresos hospitalarios, lo cual era absolutamente inaceptable desde el punto de vista práctico.

3. La Optimización

Tomando en consideración que se contaba con la siguiente información:

1. *codigoBasico* y *codigoSecundario* corresponden a enfermedades de tipo básico (*tipo* "B"),
2. *codigoExterno* es de *tipo* "E",
3. *codigoBasico* es siempre no nulo,
4. *tipo* y *codigo* forman una clave de la tabla *Enfermedad*,

se introdujo en la definición de la tabla *EgresoHospitalario* un nuevo atributo: *tipo_enf*, del mismo tipo que *tipo* de la tabla *Enfermedad*, de modo que siempre tuviese el valor "B" (dado que el *codigoBasico* es siempre no nulo y que se sabe que corresponde a una causa basica de enfermedad). Se puede así definir una clave foránea hacia la tabla *Enfermedad*, con los campos *tipo_enf* y *codigoBasico*. De esta forma, no se necesita tener la validación como parte del *trigger* de inserción, sino que se aprovecha las optimizaciones que puede hacer el administrador de la base de datos para controlar las claves foráneas.

No fue posible definir los otros dos atributos de códigos de enfermedad de la relación *EgresoHospitalario* como claves foráneas por ser potencialmente nulos. Pero de todas maneras, la optimización semántica fue útil para mejorar las consultas usadas en el proceso de validación de los códigos secundario y externo.

En la primera solución lo único que se estaba chequeando era que el código informado por el establecimiento hospitalario, en caso de no ser nulo, fuese válido en la tabla *Enfermedad* (es decir, que estuviera en la tabla *Enfermedad*), pero no estábamos usando datos conocidos (las restricciones de integridad): en el caso de *codigoSecundario* el tipo era siempre "B" y para *codigoExterno* era "E". Al agregar estas condiciones a las consultas de verificación de cada uno de estos códigos, la búsqueda se realiza por la clave completa de la tabla *Enfermedad* y, por lo tanto, el administrador de la base de datos tiene un índice ya definido, lo cual hace que la consulta sea mucho más eficiente.

El siguiente código, optimiza las consultas de acuerdo a la explicación anterior. Así, la primera consulta SELECT que valida la existencia del *codigoSecundario*, se realiza a través de la clave de la tabla *Enfermedad*, conformada por los atributos *codigo* y *tipo*; lo mismo ocurre con la segunda consulta que valida la existencia del *codigoExterno*.

```
LET existe = 0;

IF codigoSecundario != "" AND codigoSecundario != "0000" THEN
        SELECT count(*) INTO existe
        FROM    Enfermedad
        WHERE codigo = codigoSecundario
        AND tipo = "B";
        IF existe = 0 THEN
                RAISE EXCEPTION -32761;
        END IF;
END IF;

LET existe = 0;

IF codigoExterno != "" AND codigoExterno != "0000" THEN
        SELECT count(*) INTO existe
        FROM    Enfermedad
        WHERE codigo = codigoExterno
        AND   tipo = "E";
        IF existe = 0 THEN
                RAISE EXCEPTION -32760;
        END IF;
END IF;
```

Este código, a diferencia del presentado en la sección anterior, permite que el optimizador de consultas pueda construir una estrategia de búsqueda más eficiente a través de la clave principal de la tabla *Enfermedad*.

Estos cambios en la definición de las consultas y en el esquema de la base de datos, permitieron mejorar el tiempo de procedimiento de validación de inserción, reduciendo el tiempo de ejecución de las 4 horas originales a menos de 5 minutos.

4. Algunos Aspectos de Bases de Datos Deductivas

Antes de introducirnos en la reconstrucción lógica de la solución planteada en la sección anterior, es necesario revisar algunos conceptos sobre lógica, bases de datos (en particular, de bases de datos deductivas) y optimización semántica de consultas para su mejor comprensión.

La descripción de los componentes de la base de datos (hechos, reglas deductivas, restricciones de integridad y consultas) se hará siguiendo un notación en el estilo de programación lógica utilizado en [2, 3].

Un literal es una formula atómica o la negación de una formula atómica de la lógica de predicados de primer orden [1]. Una cláusula es una disyunción de literales de la forma $S_1 \vee \ldots \vee S_m \vee \neg R_1 \vee \ldots \vee \neg R_n$, donde cada S_i y R_j representa una fórmula atómica. Las cláusulas se pueden escribir en forma equivalente usando una implicación, como $S_1 \vee \ldots \vee S_m \leftarrow R_1 \wedge \ldots \wedge R_n$, para la cláusula anterior. En particular, un literal positivo se escribe $S \leftarrow$ y un literal negativo $\leftarrow R$.

En programación lógica, un programa consiste de cláusulas. En esta notación restringiremos las cláusulas a ser de Horn, las cuales son apropiadas para representar un amplio espectro de aplicaciones en bases de datos. Una cláusula es de Horn si tiene a lo más un literal positivo. En la representación dada anteriormente si $m \leq 1$, además, los S_i conforman la *cabeza* de la cláusula y los R_j el *cuerpo*.

En una base de datos deductiva los hechos se representan como literales cerrados. Por ejemplo, el hecho que una tupla $(a_1, \ldots, a_n)$ es miembro de la relación R se expresa por el literal atómico $R(a_1, \ldots, a_n) \leftarrow$. Hechos de este tipo componen la base de datos extensional, EDB, y una relación R es llamada una relación extensional. La base de datos intencional, IDB, contiene las reglas deductivas. Las relaciones definidas por reglas deductivas son llamadas relaciones intencionales y son usualmente cláusulas de Horn. Otro componente de una base de datos es el conjunto de restricciones de integridad, IC, que contiene reglas que expresan restricciones que debe satisfacer la base de datos en toda su evolución. Limitaremos, también, las restricciones de integridad a ser cláusulas de Horn.

La estrategia de optimización semántica presentado en este artículo se basa en el esquema propuesto por Chakravarty et al. [3], según el cual la evaluación de consultas en una base de datos deductiva (que no posee reglas de inferencia de naturaleza recursiva) puede particionarse en una fase de compilación semántica seguida por una fase de transformación de consultas.

La compilación semántica se puede describir informalmente como el proceso de retener fragmentos relevantes de las restricciones de integridad. Esto puede verse como un paso en el cual toda la información útil (para el propósito del procesamiento de consultas) ha sido extraída de las restricciones de integridad. Este proceso es realizado sobre las componentes relativamente estables de la base de datos (como EDB, IDB e IC), de modo que la compilación semántica se ejecuta sólo una vez sobre una base de datos deductiva.

Durante esta fase, los fragmentos de las restricciones de integridad, denominados residuos, son computados y asociados con reglas deductivas. El resultado es un conjunto de axiomas de restricción semántica, que pueden ser filtrados y almacenados para su uso posterior.

Cuando se expresa una consulta, un transformador semántico usa los residuos almacenados para generar posiblemememente varias consultas semánticamente equivalentes. Informalmente, una consulta es semánticamente equivalente a la consulta original si ambas producen las mismas respuestas sobre la misma base de datos. Asi, eventualmente, una o varias de las nuevas consultas podrán ser procesadas más rápidamente que la consulta original.

Para producir los residuos en la fase compilación semántica, es necesario manejar la noción de *subsunción*. Subsumir significa incluir o considerar algo como un caso particular sujeto a una regla o principio general. En este caso, se intenta subsumir los cuerpos negados de las definiciones de las relaciones intencionales en las restricciones de integridad. Es poco frecuente que se dé este tipo de subsunción, ya que involucra una contradicción y, por lo tanto, puede deducirse que no pueden haber tuplas en la relación. En cambio, es más común que sólo una parte de la restricción de integridad subsuma a la relación. En este caso, la fracción discordante (llamada residuo) representa información adicional que puede llegar a asociarse a la relación.

Por ejemplo, si tenemos:

$$\leftarrow Enfermedad(tipo, codigo), tipo = A,$$

que es una restricción de integridad que dice que "no existen enfermedades de tipo A" y una relación intencional:

$$EnfermedadA(tipo, codigo) \leftarrow Enfermedad(tipo, codigo), tipo = A,$$

entonces, es evidente que no hay tuplas que puedan asociarse a la relación, ya que existe una contradicción entre ambas cláusulas. En este caso, la negación del cuerpo de la definición de la relación intencional es subsumida por la restricción de integridad. Ahora, si definimos una nueva relación:

$$EnfermedadT(tipo, codigo) \leftarrow Enfermedad(tipo, codigo), codigo > 1000,$$

entonces, si bien la restricción de integridad no subsume a

$$\leftarrow Enfermedad(tipo, codigo), codigo > 1000,$$

sí lo hace una parte de ella, a saber, $\leftarrow Enfermedad(tipo, codigo)$. En este caso la fracción discordante es:

$$\{\leftarrow tipo = A\}.$$

Este residuo puede asociarse a la relación $EnfermedadT$ para enfatizar el hecho que no son admisibles las enfermedades de tipo A.

Formalmente, una cláusula C subsume a una cláusula D si hay una sustitución σ tal, que $C\sigma$ es una subcláusula de D (es decir, si cada literal de $C\sigma$ está en D).

Una *sustitución* es un conjunto finito de asociaciones entre variables y términos: $\{t_1/v_1, \ldots$
$, t_n/v_n\}$, donde cada v_i es una variable, cada t_i es un término y se cumple que: (1) cada variable esta asociada a lo más con un término, y (2) ninguna variable asociada con un término aparece en alguna de las expresiones. Al proceso de determinar cuándo dos expresiones son idénticas por una apropiada sustitución de sus variables se le denomina *unificación.*

En nuestro caso, el concepto importante es el de subsunción parcial: Una restricción de integridad IC subsume parcialmente a un axioma A, si IC no subsume al cuerpo de A, pero una subcláusula de ella sí lo hace.

Ahora, necesitamos un proceso mecánico que utilice la noción de subsunción en la búsqueda de residuos. Para esto es posible aplicar una variación del algoritmo descrito en [4]. Antes de ello revisemos los conceptos de resolución y resolución lineal.

En términos simples, resolución dice que si una cláusula contiene el literal L y otra cláusula contiene el literal $\neg L$, podemos inferir la cláusula consistente de todos los literales de ambas cláusulas sin el par complementario. Resolución lineal establece una estrategia para aplicar resolución sobre un conjunto de cláusulas, ésta dice que en cada aplicación de resolución debe intervenir el último resolvente producido y un elemento del conjunto original de cláusulas.

El algoritmo puede, entonces, resumirse en los siguientes pasos:

- Instanciar la relación a una cláusula cerrada (sin variables) utilizando para ello nuevas constantes no presentes en la restricción de integridad ni en la relación. Denotemos con θ esta sustitución.

- Negar la cláusula cerrada, obteniendo un conjunto de literales.

- Construir un árbol de resolución lineal utilizando inicialmente la restricción de integridad y uno de los elementos del conjunto de literales obtenido en el punto anterior. Usar en cada paso un nuevo elemento del conjunto de literales.

- La fracción de la IC estará instanciada como una cláusula cerrada. Debe entonces aplicarse θ^{-1} (denominada sustitución inversa), que consiste en recuperar las variables que fueron sustituidas originalmente.

- Finalmente, la cláusula obtenida corresponde a un residuo.

En la siguiente sección, aplicaremos estas ideas al problema planteado en este artículo, esto permitirá ilustrar el uso del algoritmo de subsunción y su importancia en el contexto de optimización semántica de consultas.

5. Reconstrucción Lógica de la Aplicación

En esta sección queremos poner la solución optimizada dada al problema en el contexto de la SQO, tal como esta se da en bases de datos deductivas. Esto se puede ver como una reconstrucción *lógica* de los *mecanismos* empleados en la solución optimizada.

Retomando el problema planteado, lo que tenemos es un conjunto de consultas cuyo resultado valida un conjunto de restricciones de integridad referencial. La idea subyacente está en transformar este conjunto de consultas en otro conjunto semánticamente equivalente.

Analizando la descripción dada en la sección 3, se puede ver que los dos primeros puntos de la descripción corresponden a información semántica, y los dos últimos, a información física que puede ser aprovechada por el optimizador del sistema administrador de base de datos una vez transformadas las consultas.

La información semántica, puede ser expresada como un conjunto de restriciones de integridad especificadas en programación en lógica, viendo nuestra base de datos como una base de datos deductiva [8]:

$$RI1 : Enfermedad(B, cb) \leftarrow EgresoHospitalario(i, cb, cs, ce)$$

$$RI2 : Enfermedad(B, cs) \leftarrow EgresoHospitalario(i, cb, cs, ce)$$

$$RI3 : Enfermedad(E, ce) \leftarrow EgresoHospitalario(i, cb, cs, ce).$$

Utilicemos ahora el método de dos fases descrito anteriormente. En la fase de compilación semántica debemos aplicar el álgoritmo de subsunción. En este contexto, consideraremos la restricción de integridad $RI1$, utilizando la relación *EgresoHospitalario*. Para ello, es necesario expresar la relación en términos de un axioma trivial de la forma $R \leftarrow R$ (recuérdese que la noción de subsunción parcial esta dada en términos de relaciones intencionales), en nuestro caso:

$$\mathbf{A}: \quad EgresoHospitalario(i, cb, cs, ce) \leftarrow EgresoHospitalario(i, cb, cs, ce).$$

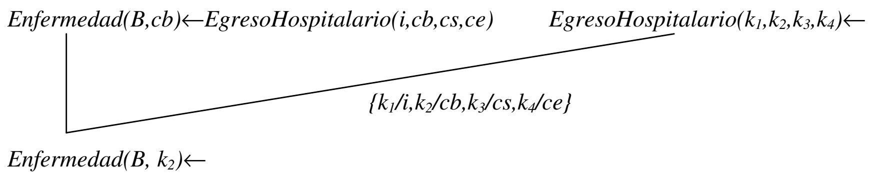
El siguiente paso consiste en instanciar el cuerpo del axioma transformándolo en una cláusula *ground*. Usaremos $k_1, \ldots, k_4$ para las constantes y llamaremos Ab al cuerpo del axioma y θ a la sustitución. Así tenemos:

$$\mathbf{Ab}\theta: \quad \leftarrow EgresoHospitalario(k_1, k_2, k_3, k_4).$$

Debemos ahora negar $Ab\theta$, con lo que obtenemos:

$$\neg\mathbf{Ab}\theta: \quad EgresoHospitalario(k_1, k_2, k_3, k_4) \leftarrow .$$

En terminos generales, $\neg\mathbf{Ab}\theta$ es un conjunto de literales que, mediante resolución, se proceden a cancelar con los literales que componen la cláusula $RI1$. Aplicando el mecanismo de resolución lineal, que en este caso particular resulta simple, obtenemos:



Finalmente, aplicando θ^{-1}, se obtiene

$$Enfermedad(B, cb) \leftarrow .$$

Esta fórmula es un *residuo*. En términos generales, los residuos representan información que puede ser asociada a la relación de la que se originan, e indican, además, que existe una subsunción parcial de la relación por parte de la restricción de integridad.

La asociación entre residuos y relaciones produce los denominados *Axiomas de Restricción Semántica* (en lo sucesivo ARS). Por ejemplo, para nuestro caso tenemos:

$$\textbf{ARS1}: \quad \mathit{EgresoHospitalario}(i, cb, cs, ce) \leftarrow \mathit{EgresoHospitalario}(i, cb, cs, ce)$$

$$\{\mathit{Enfermedad}(B, cb) \leftarrow; \;\; \mathit{Enfermedad}(B, cs) \leftarrow; \;\; \mathit{Enfermedad}(E, ce) \leftarrow\}$$

La segunda fase denominada *Transformación Semántica* utiliza los ARS para transformar una consulta en una o más consultas equivalentes. En nuestro caso particular, las consultas existen como parte de un procedimiento (trigger) de validación de inserción, por lo que debemos interpretarlas en este contexto. Por ejemplo, si deseamos ingresar $\mathit{EgresoHospitalario}(I, CB, \_, \_)$ (por simplicidad se han obviado los valores de código secundario y externo), deberíamos consultar por la existencia de algún tipo t (una variable) en la tabla *Enfermedad*. Formalmente:

$$\textbf{Q1}: \quad \leftarrow \mathit{Enfermedad}(t, CB), \mathit{EgresoHospitalario}(I, CB, \_, \_),$$

y deberíamos esperar que esta fórmula sea verdadera.

A partir del axioma de restricción semántica ARS1 (en particular, utilizando el primer residuo), es posible incorporar una nueva restricción a la consulta, sin alterar su significado semántico.

Para mayor claridad expresemos **Q1** como una disyunción de literales:

$$\textbf{Q1}: \quad \neg \mathit{Enfermedad}(t, CB) \vee \neg \mathit{EgresoHospitalario}(I, CB, \_, \_)$$

y también a **ARS1**, a la cual se le ha asociado explícitamente el primer residuo:

$$\textbf{ARS1}: \quad \mathit{EgresoHospitalario}(i, cb, cs, ce) \vee \neg \mathit{EgresoHospitalario}(i, cb, cs, ce) \vee \neg \mathit{Enfermedad}(B, cb)$$

Si unificamos considerando la sustitución $\{B/t, I/i, CB/cb\}$ y luego aplicamos resolución sobre ambas cláusulas, obtenemos la consulta modificada:

$$\textbf{Q1}': \quad \leftarrow \mathit{Enfermedad}(B, CB), \mathit{EgresoHospitalario}(I, CB, \_, \_).$$

Esta nueva consulta, que preserva el significado semántico de la primera, es claramente más ventajosa, ya que el optimizador tradicional del sistema administrador de base de datos puede hacer uso de la estructura de datos creada para la relación *Enfermedad* (típicamente un B-tree) con clave formada por los atributos *tipo* y *codigo*.

6. Conclusiones

Uno de los aspectos importantes de considerar en este trabajo está en la aplicación de conceptos propios del ambiente de bases de datos deductivas al ambiente de bases de datos relacionales. Demostrando con esto que existen conceptos importantes que pueden ser traspasados y aplicados con éxito.

Se probó que la ausencia de un Optimizador Semántico de Consultas no es motivo para no utilizar exitosamente los métodos propuestos para la SQO, produciendo mejoras notables con respecto a los tiempos de de respuesta originales. Mas aún, un adecuado proceso de deducción basado en la lógica puede demostrar la correctitud de un proceso de optimización.

El ejemplo presentado amplía el espectro de uso de la SQO a consultas usadas con fines diferentes de los tradicionales. En particular, tenemos que, en este caso, el proceso de validación asociado a una regla de integridad es mejorado utilizando la información semántica presente en otras reglas de integridad. Por ejemplo, la consulta involucrada en la restricción de integridad que valida que: *"el valor del atributo codigoBasico a ingresar en la tabla EgresoHospitalario se encuentre entre los valores de codigo en la tabla Enfermedad"* es optimizada utilizando la restricción de integridad que dice que: *"el codigoBasico corresponde siempre a enfermedades de tipo B en la tabla Enfermedad"*.

La solución presentada en la sección 3 para la validación del código básico expresa implícitamente el uso de la consulta **Q1**$'$. Ella implementa la consulta como una búsqueda por clave entre las tablas *Enfermedad* y *EgresoHospitalario*, donde el valor del atributo *tipo* permanece constante (con valor "B") y el valor del atributo *codigo* esta dado en la tupla a insertar. Nótese que la conjunción de las tablas *Enfermedad* y *EgresoHospitalario* en el cuerpo de **Q1**$'$, se puede interpretar en los términos de la solución optimizada original, a saber, como la incorporación de un atributo extra (*tipo*) en la tabla *EgresoHospitalario*.

Agradecimientos: Se agradece a Jack Minker por estimulantes conversaciones y por toda la información que nos entregó sobre el tema de optimización semántica de consultas durante su visita a la Universidad Católica de Chile. Además, se agradece a Gerardo León por sus aclaraciones sobre el proyecto de estadísticas hospitalarias. Esta investigación fue finaciada parcialmente por proyectos Fondecyt (#1930554) y DIPUC; y por el Centro de Servicio Exterior en Computación (CESEC) de la Universidad Católica de Chile.

Referencias

[1] Bertossi, L. *Lógica para Ciencia de Computación.* Ediciones Universidad Católica de Chile, 1996.

[2] Chakravarthy, U.; Fishman, D.H. y Minker, J. *Semantic Query Optimization in Expert Systems and Database Systems.* Expert Database Systems, 1986, 651-674.

[3] Chakravarthy, U.; Grant, J. y Minker, J. *Logic-Based Approach to Semantic Query Optimization.* ACM Transaction on Database Systems, Vol. 15, No. 2, Junio 1990, 162-207.

[4] Chang, C. L., y Lee, R. C. T. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[5] Hammer, M. y Zdonik, S. *Knowledge-based query processing.* Proc. Sixth Intl. Conf. On Very Large Databases, Montreal, Octubre 1980, 137-147.

[6] King, J.J. *Quist: A system for semantic query optimization in relational databases.* Proc. 7th. Intl. Conf. On Very Large Databases, Cannes, Francia, Septiembre 1981, 510-517.

[7] Levy, A. y Sagiv, Y. *Semantic Query Optimization in Datalog Programs.* Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Databases Systems, PODS-95, San Jose, California, 1995.

[8] Lloyd, J. *Foundations of Logic Programming.* Springer, 2nd edition, 1987.

[9] Siegel, M. *Automatic Rule Derivation For Semantic Query Optimization.* Expert Database Systems, Proc. 2nd. Intl. Conf. On Expert Database Systems, 1988.

[10] Zhang, X. y Ozsoyoglu Z.M. *Reasoning with implication and referential constraints in semantic query optimization.* Proceeding of the Post-ILPS'94 Workshop on Constraints and Databases, Ithaca, N.Y., U.S.A., 1994.