

Integridad de datos relacionales

Los datos erróneos o inconsistentes almacenados en bases de datos son causa de errores y fallos, a veces con graves consecuencias económicas para las empresas que los gestionan. A pesar de ello, la comprobación convencional de las restricciones de integridad en bases de datos resulta demasiado costosa. En este artículo presentamos SSQL, una metodología diseñada en el ITI, y actualmente en proceso de desarrollo, para dar soporte a la especificación y comprobación eficiente de restricciones arbitrariamente complejas.

Motivación

En los últimos números de ActualidadTIC se presentaron los sistemas Hydra y COPLA, del ITI. Ambos permiten salvaguardar datos almacenados electrónicamente frente a posibles fallos de software o hardware. Sin embargo, la mayoría de errores y fallos en la información contenida por las bases de datos no se debe a tales fallos, sino a un desacuerdo entre la semántica pretendida de los datos y los valores que de hecho se almacenan. En otras palabras, los datos almacenados guardan a menudo valores incorrectos de sus atributos. En contexto de bases de datos se habla de *violación de la integridad*, es decir, violación de la consistencia semántica de los datos, cuando uno o más valores son obsoletos, engañosos, o incorrectos de cualquier forma.

En la vida cotidiana, los pequeños errores suelen ser inofensivos y a menudo pueden ser corregidos o ajustados sobre la marcha. Pero cuando hablamos de computadoras incluso los errores más nimios pueden tener consecuencias desastrosas. Una coma ausente, por no hablar de errores tipográficos o relaciones incorrectas entre los datos, puede impedir el funcionamiento correcto de un sistema. En los sistemas de bases de datos, los fallos en la consistencia semántica y la integridad pueden tener consecuencias dramáticas sobre los resultados económicos de empresas que usan tales bases de datos para almacenar información sobre productos y clientes, y para administrar su negocio. Por ejemplo, si los datos de contabilidad de una compañía violan las convenciones para las fórmulas de costos, tales errores pueden resultar en una grave pérdida de beneficios, aunque hayan ocurrido inintencionadamente.



”¡¡Te dije que no debíamos confiar en estos datos!!”

Otros ejemplos de datos fallidos que violan reglas de integridad son, por ejemplo, datos erróneos sobre clientes, direcciones o nombres inexactos en las facturas, precios incorrectos, etc. Tales errores a menudo conllevan tediosas interacciones adicionales con los clientes, costosos retrasos en los pagos, y una menor satisfacción de los clientes.

Los fallos en la integridad de los datos pueden tener graves consecuencias económicas.

En general, la integridad de la información en una base de datos puede ser violada con cada modificación de su contenido, resultando en la necesidad de comprobar todas y cada una de las restricciones de integridad. Tal sobrecoste no es tolerable en una base de datos operacional, que necesita sus recursos y potencia computacional sobre todo para contestar a las preguntas y solucionar cuestiones de los usuarios.

Al problema de los costes prohibitivos que impiden un mantenimiento minucioso de la integridad, se añade la gran dificultad de expresar las restricciones de manera precisa, correcta y completa en el marco formal de un sistema de bases de datos. El grupo de trabajo de bases de datos del ITI ha desarrollado una metodología llamada SSQL para expresar y evaluar restricciones de integridad arbitrariamente complejas para cualquier tipo de aplicación de bases de datos, sin incurrir en costes excesivos sobre el rendimiento.

Este artículo revisa los principios fundamentales de SSQL. Mediante este proyecto el Instituto pretende colaborar con los administradores de bases de datos de PYMEs en la resolución de los problemas relacionados con el mantenimiento de la integridad y consistencia de los datos de que tales administradores son responsables.

Objetivos de SSQL

El acrónimo SSQL está compuesto por un prefijo S seguido del sufijo SQL. Este significa *Structured Query Language*, es decir, “lenguaje de preguntas estructuradas”, que es el lenguaje estándar de programación declarativa, muy bien establecido en el campo de las bases de datos (BBDD), cubriendo más del 95% del mercado. El prefijo S significa *sound*, adjetivo inglés con connotación lógica estricta, que los diccionarios traducen como “sano”, “sólido”, “seguro”, “severo”, todo ello en el sentido de datos almacenados sin contradicciones, es decir, datos consistentes, correctos y en correspondencia con el sentido buscado por el programador. En resumen: datos con integridad no-violada. Los datos relacionales almacenados en BBDD, siendo *sound*, no llevan a ningún problema de integridad, ninguna conclusión falsa, ni una mala interpretación de su contenido.

Integridad de datos relacionales

El principal objetivo tecnológico es extender los sistemas gestores de bases de datos relacionales (SGBD) con restricciones de integridad más generales y poderosas que las condiciones de integridad soportadas por BBDD comerciales. Hemos estudiado la viabilidad de las ideas básicas. Estamos desarrollando un prototipo de sistema automático y eficaz para especificar y hacer cumplir tales restricciones a lo largo de la evolución de la BBDD. En general, las restricciones de integridad se definen como la clase de condiciones especificadas en la sintaxis declarativa más general de SQL. Actualmente, ningún SGBD comercial provee el soporte para restricciones de integridad tan generales. La clase de condiciones declarativas de integridad realizada en SSQL es mucho más expresiva y poderosa que la que ofrecen los sistemas en el mercado, permitiendo capturar el sentido del mundo real asociado con los datos almacenados.

Habitualmente, la especificación del esquema de una BD incluye tipos sencillos y rudimentarios de restricciones de integridad. Tales restricciones son condiciones declaradas en un lenguaje formal para capturar el sentido correcto de los datos y para evitar que alguna modificación de estos pudiera violar su corrección y exactitud. La finalidad de la metodología SSQL es ampliar la expresividad y el alcance de la especificación de restricciones de integridad en BBDD relacionales, permitiendo una complejidad arbitraria al expresar cualquier condición, sin comprometer la eficacia de hacerla cumplir automáticamente.

Las restricciones de integridad permiten la declaración y comprobación de condiciones para expresar la consistencia, corrección y exactitud de datos almacenados. Todos los SGBD comerciales ofrecen la posibilidad de especificar condiciones sencillas de integridad. Pero existe muy poco soporte declarativo para las restricciones que se relacionan con el significado general de las aplicaciones y con las partes del mundo codificadas en las BBDD. El objetivo de SSQL es realizar un soporte eficaz de restricciones de integridad que sean de una generalidad y universalidad sin precedente en el mercado, sobrepasando todo lo que se ofrece actualmente por los vendedores más importantes de SGBD. Y asegurando que toda esta funcionalidad adicional se cumpla sin costes exorbitantes.

Beneficiarios de SSQL

Hay tres clases diferentes de beneficiarios inmediatos de SSQL. En primer lugar los diseñadores de BBDD pueden beneficiarse de la mayor facilidad y claridad de la especificación declarativa de restricciones de integridad arbitrariamente complejas. En segundo lugar, los usuarios de BBDD (es decir, gerentes, administradores o colaboradores en proyectos, o bien empleados o clientes con sus aplicaciones) pueden disfrutar las ventajas de un nivel superior de consistencia y corrección de sus datos. En tercer lugar, los vendedores de SGBD o de sistemas que se apoyan en ellos, puesto que esperamos les será fácil incorporar la tecnología SSQL en sus productos, con la finalidad de mejorar su funcionalidad.

En general es un fenómeno bien conocido que una colección de datos almacenados puede perder su integridad a lo largo de su ciclo de vida. Típicamente, la

información que ha sido establecida sólidamente una vez en una BD, tiende a ser sobrescrita, incomprensiblemente corregida o modificada o a quedar obsoleta con el curso del tiempo. Las restricciones de integridad sirven para prevenir tal deterioro de la consistencia y corrección de las informaciones almacenadas.

Normalmente, los desarrolladores de aplicaciones de BBDD están forzados a escribir código adicional para hacer cumplir la integridad de los datos almacenados. Tal código propietario es propenso a errores, tiende a producir efectos secundarios indeseados, y a medida que se acumulan las modificaciones de datos, fácilmente puede comportarse de forma incontrolada. Además, el uso ad-hoc de disparadores procesales (desgraciadamente recomendado por los vendedores de BBDD como medio de apoyar la integridad) tiene esencialmente las mismas desventajas, y puede producir reacciones en cadena incontrolables. En comparación, las restricciones de integridad son declarativas, tienen características demostrables de comportamiento correcto y pueden ser mantenidas muy fácilmente, debido a su independencia de los programas de uso. Así, puede esperarse que la utilización de la tecnología SSQL dé lugar al aumento de la calidad de los datos y de la productividad de los usos de bases de datos.

Un aspecto importante de SSQL es su usabilidad con los productos de todos los vendedores importantes de BBDD conformes al estándar establecido de SQL. Por lo tanto, SSQL resultará interesante y útil tanto para los vendedores de bases de datos como para sus clientes. Además, intentamos desarrollar nuestro sistema para ser utilizado no solo sobre productos comerciales, sino también, en una versión futura, para ser integrado en el código del principal producto *open source* de BD, MySQL. Puesto que MySQL es un producto de código libre, su ampliación con la integridad semántica reforzará su posición primordial en el movimiento de código abierto.

Estado actual

La integridad de datos relacionales puede ser expresada de manera declarativa o procesal. Las soluciones que existen en el campo científico son de tipo declarativo y muy avanzado, pero nunca han salido del campo teórico. Por otra parte, lo que está realizado en los SGBD comerciales con respecto a la integridad es bastante pobre en términos de declaratividad. Por ello las restricciones de integridad avanzadas han de ser especificadas de manera procesal. Esto las hace más propensas a errores y más difíciles de mantener, pues a menudo están completamente oscurecidas dentro del código de la aplicación. Una contribución esencial de SSQL es la de adaptar y trasladar lo que ha sido concebido de forma teórica y científica al mundo práctico y comercial.

Actualmente, la mayoría de los SGBD comerciales solamente ofrecen soporte para los siguientes tipos declarativos de restricciones de integridad:

- Restricciones de dominio (por ejemplo tipos de datos básicos, opcionalmente de rango restringido, como números, cadenas de caracteres, etc., incluyendo opciones que permitan valores nulos).

Integridad de datos relacionales

- Restricciones de unicidad (por ejemplo las claves primarias, que garantizan la inconfundibilidad de los objetos almacenados, o los índices únicos, que facilitan la búsqueda sistemática de datos).

- Restricciones de clave ajena (para asegurar la correlación correcta entre diferentes tablas, p.e., que para cada asignación de una persona a un proyecto en una tabla que almacene los equipos de personal, haga falta que cada una de estas personas esté también registrada en la tabla de empleados).

Las restricciones de dominio y unicidad son muy simples y básicas. Las primeras están ligadas a una sola columna de una tabla, mientras que las segundas pueden aplicarse a combinaciones de columnas, pero dentro de una misma tabla. Las restricciones de clave ajena relacionan columnas que pueden estar en dos tablas distintas. Adicionalmente tales columnas deben satisfacer una restricción de unicidad. La potencia expresiva de las claves ajenas es bastante limitada. Por ejemplo no es posible expresar, con ninguna combinación de las tres construcciones citadas, que en una tabla de registro de ciudadanos solteros y matrimonios en una comunidad, cada persona esté casada como máximo con otra persona y que la edad de cada persona en el momento de casarse sea al menos 18, a menos que haya un consentimiento explícito por parte de los padres legales. En cambio la potencia expresiva de las restricciones de integridad soportadas por SSQL no está limitada en el número de tablas involucradas ni por ningún otro requisito estructural.

En los últimos años, también la expresividad declarativa de SQL se acercó a la completitud de expresar y evaluar restricciones de integridad arbitrariamente generales, en virtud de la introducción de las subconsultas, unas construcciones de calificación (EXISTS) y negación (NOT), y tablas especiales llamadas vistas (cuyas definiciones se apoyan en otras tablas más básicas). Sin embargo, todos los manuales de SGBD comerciales todavía recomiendan la implantación de restricciones de integridad complejas mediante procedimientos, porque el uso de medidas declarativas resulta a menudo demasiado costoso, hasta al punto de su intolerabilidad. Es precisamente este problema de falta de eficiencia lo que SSQL trata de resolver.

Las restricciones de integridad que van más allá del nivel introductorio de SQL, normalmente implican *joins* (productos de combinaciones) enormes de varias tablas, consultas de tablas enteras, subconsultas y negaciones anidadas, y similares. Así, los costes de su evaluación llegan a ser fatalmente caros. Las aplicaciones típicas de SGBD (p.e., sistemas de la reserva de vuelos) y también los procesos de tiempo crítico (p.e., el *data warehousing* para extraer, borrar, transformar, homogeneizar y cargar datos empresariales), no pueden soportar caras comprobaciones de integridad. Por eso, los vendedores de SGBD desaconsejan la utilización de condiciones declarativas de consulta. En lugar de eso, se debe recurrir a las construcciones más opacas basadas en procedimientos.

Hay tres maneras de expresar restricciones de integridad mediante procedimientos: como *triggers* SQL disparados ante actualizaciones predefinidas, como procedimientos almacenados asociados a determinadas transacciones, o incluyéndolas directamente en el código de la aplicación.

Cada una de estas tres opciones compromete gravemente el ideal de declaratividad. La utilización de procedimientos implica los peligros ya mencionados. Además, cada implantación mediante procedimientos de las restricciones de integridad tiene el inconveniente adicional de frustrar su posible simplificación, que aceleraría su evaluación enormemente. Por el contrario, SSQL aprovecha esta posibilidad de simplificar la comprobación.

Ejemplos de SSQL

Fijemos una BD conteniendo entradas de trabajadores o gerentes en la jerarquía de empleados de una gran empresa. Una condición típica de integridad semántica que no puede ser expresada y mantenida eficientemente con las medidas de integridad proporcionadas por los SGBD comerciales sería la de que ningún obrero sea gerente. Otra condición aún más complicada podría exigir para cualquier departamento la existencia de un individuo que fuera el superior de cada empleado del propio departamento, excepto este mismo individuo.

Dado que cada actualización puede resultar en una violación de la integridad de los datos, se sigue la necesidad de comprobar las restricciones de integridad casi continuamente en BBDD con elevada frecuencia de transacciones. Tal sobrecoste en general no es tolerable. Por ejemplo, la comprobación de una restricción de integridad que postule que la altura de la jerarquía de empleados no sobrepase tres niveles, necesitaría, en una BD con entradas para cada empleado y cada uno de sus superiores directos, la travesía de un espacio de búsqueda enorme (1.000.000.000.000 de pares de datos relacionales cuando el número de empleados sea 10.000). En una BD muy dinámica con miles de datos la comprobación de restricciones complejas de integridad sería pues prácticamente imposible.

Consideremos ahora como ejemplo concreto una BD para la administración del personal en un departamento de una empresa. Supongamos una tabla con el nombre **superior**, capturando qué empleado es el superior directo de otro empleado y asumamos por simplicidad que la tabla **superior** tiene solamente dos columnas, cada una con una gama de los valores que permiten la identificación única de empleados. Una entrada **superior(A,B)** significaría que el individuo **A** es un superior directo de **B**. Supongamos que en una ronda de reestructuración ha sido impuesta una regla de negocio según la cual la jerarquía directiva deba ser aplanada, de forma que solo tres niveles jerárquicos de gerencia están permitidos. Esto significa que no debe existir ninguna cadena de entradas en la tabla **superior** de la forma:

superior(X1,X2)–superior(X2,X3)–superior(X3,X4),

donde X1, X2, X3, X4 simbolizan individuos de manera formal. O, de forma más exacta:

No existen X1, X2, X3, X4 tal que **superior(X1,X2)** y **superior(X2,X3)** y **superior(X3,X4)**.

La traducción de especificaciones de restricción de integridad en lenguaje natural, aún más conveniente para el usuario, en una forma equivalente de SQL está fuera del alcance de SSQL. Sin embargo, la continuación del proyecto SSQL incluirá tal objetivo, teniendo en mente

Integridad de datos relacionales

que un grupo muy fuerte del ITI se ocupa de lenguajes naturales.

En los SGBD del mercado, una regla como la propuesta ha de ser llevada a cabo por un procedimiento uso-dependiente, que comprobaría si hay alguna cadena que viole el requisito especificado en la regla. De ser así, publicaría un mensaje de advertencia. Tales procedimientos son muy propensos a errores. Por el contrario, SSQL permite una representación declarativa de la regla de negocio en forma de restricción de integridad, como la siguiente condición:

```
NOT EXISTS (SELECT * FROM superior s1,
            superior s2, superior s3 WHERE superior1.#2 =
            superior2.#1 AND superior2.#2 = superior3.#1)
```

Según la sintaxis estándar SQL-2, los alias superior1, superior2, superior3 sirven para distinguir los tres diferentes usos de la tabla **superior**, y los dos atributos de la tabla están referidos por .#1 y .#2, respectivamente. La evaluación de condiciones como la anterior puede ser extremadamente costosa. Si hay 10.000 empleados y 30.000 superiores directos (para cada empleado puede fácilmente haber tres superiores diferentes, p.e., un técnico, un administrativo y un comercial), nuestra condición SQL proporciona un espacio de búsqueda de 27 billones de tripletes de pares de empleados, cada uno de los cuales podría en principio ser un candidato a violar la integridad. Este enorme espacio tendría que ser recorrido para cada cambio en la BD. Con SSQL, el espacio se reduce en 4 potencias de 10 a 2.700 millones. Otra reducción del mismo orden se logra con un índice apropiado impuesto sobre la tabla **superior**.

Loa seis pasos de SSQL

Vamos a ilustrar la funcionalidad innovadora de SSQL, que logra una reducción dramática de costes, mediante un ejemplo concreto y una sucesión de seis pasos de la metodología.

Consideremos una BD con tablas con atributos de nombre (**name**) para trabajadores (**worker**) y gerentes (**manager**). Supongamos la restricción de que ningún trabajador sea gerente. Eso se expresa en SQL como

```
NOT EXISTS (SELECT * FROM worker, manager
            WHERE worker.name = manager.name)
```

Un programador de SQL podría decir que dicha restricción será mucho más fácilmente comprobable con un disparador relacionado a las inserciones en la tabla **worker**. Sin embargo, es fácil pasar por alto que la restricción de integridad arriba indicada ya exige otro disparador, para las inserciones en **manager**. Y, aparte de que los disparadores codificados a mano sean susceptibles de fallos, pueden traer también efectos imprevisibles de interacciones mutuas.

Supongamos que **INSERT worker(Fred, sales)** sea una actualización a llevar a cabo. La metodología implica la ejecución de los siguientes pasos:

1. Generar diferencia entre estado previo y posterior

La actualización explícita **INSERT worker(Fred, sales)** puede tener actualizaciones derivadas implícitas sobre

vistas de la BD cuyas definiciones involucren a la tabla **worker**. El conjunto de actualizaciones implícitas y explícitas constituye la diferencia entre el estado previo de la base de datos y el nuevo. Cada acción en esta diferencia puede violar la integridad.

2. Evitar actualizaciones vanas

Si Fred era ya un trabajador (p.e., en otro departamento) antes de la inserción, no es necesario comprobar de nuevo que no es un gerente.

3. Identificar las restricciones relevantes

A menos que se haya aplicado 2, la restricción de que ningún trabajador pueda ser gerente es claramente relevante por las inserciones en la tabla **worker** (pero nunca por un borrado), y debe ser comprobada.

4. Especializar las restricciones relevantes

Para la inserción dada, la condición SQL

```
EXISTS (SELECT * FROM worker, manager WHERE
        worker.name = manager.name)
```

podría especializarse a una forma mucho menos cara:

```
EXISTS (SELECT * FROM worker, manager
        WHERE worker.name = 'Fred' AND worker.name =
        manager.name)
```

5. Optimizar las restricciones especializadas

Claramente, la condición especializada en 4 puede ser optimizada por la siguiente sentencia:

```
EXISTS (SELECT * FROM manager WHERE name =
        'Fred')
```

6. Evaluar restricciones

Finalmente, la evaluación de la consulta resultante sobre si Fred es un gerente será sencilla. La búsqueda de una sola fila en una sola tabla es, por supuesto, mucho menos costosa que tener que evaluar por completo todas las restricciones de integridad.

Conclusión

El ejemplo comentado es muy sencillo. Sin embargo se pueden obtener las mismas simplificaciones para restricciones arbitrariamente complejas.

Actualmente en el ITI se está desarrollando una metodología sistemática que traduce automáticamente los resultados de esta secuencia de pasos en *triggers* seguros que resultan aún más eficientes.

Autor: Hendrik Decker
Más información: sidi@iti.upv.es