

Bases de Datos Activas (apuntes de CI-5311)

Soraya Abad Mota

Actualizado Noviembre 2005

1. Motivación

Las bases de datos convencionales se consideran “muerta” o pasivas, en el sentido de que es el usuario o el programa de aplicación quien decide qué hacer a la base de datos y ella no reacciona a las acciones ejecutadas sobre ella.

¿Cómo se puede motivar la necesidad de que una base de datos sea *activa*?

Supongamos una aplicación de inventario para una fábrica de productos que se apoya en una base de datos de los productos, su cantidad en *stock* (existencia) y que varía de acuerdo con la venta de productos. La fábrica tiene la descripción de cada producto, del cual se tiene una cantidad en existencia; a medida que se venden los productos se altera la cantidad en existencia. Lo deseable es que cada producto tenga un rango de la cantidad en existencia, definido por un nivel mínimo y un nivel máximo de cantidad.

Si la cantidad en existencia de un producto se sale de su rango, lo ideal es que se detecte esta situación inmediatamente y se tomen medidas al respecto.

¿Cuáles son las alternativas de solución para este problema?

1. **En cada transacción** que altera el valor de la cantidad en existencia, se coloca la verificación de los límites y se toman las medidas cuando se violen los límites.
2. Se construye una **transacción especial** que, preiódicamente revisa la base de datos, verificando el rango de la cantidad en existencia de cada producto con la cantidad real.

La solución 1 es un desastre en mantenimiento y constituye una mala práctica de ingeniería de software.

Con respecto a la solución 2, ¿cuándo se hace la verificación? ¿con cuál periodicidad? Si se hace muy a menudo, ésto puede incidir negativamente en el rendimiento y puede ser muy ineficiente. Si se hace con poca frecuencia, puede que no se detecten las condiciones a tiempo y no se tomen las acciones oportunamente.

El *paradigma de bases de datos activas* planteado por Morgenstern en 1983, describe la noción de que una base de datos sea *activa*, como una metáfora de su comportamiento, el cual se concentra en:

la dinámica de la interacción con los usuarios
unido a
la "inteligencia" de la base de datos para lidiar
con las consecuencias e implicaciones de esa interacción.

El artículo de Dayal, Hanson y Widom nos da una definición práctica de lo que es una *base de datos activa*, es aquella base de datos capaz de *detectar* situaciones de interés y de *actuar* en consecuencia. Alguien debe especificar las situaciones a detectar y las acciones a llevar a cabo en esas situaciones. El mecanismo que se utiliza para especificar estos aspectos se parece a las reglas de producción utilizadas en el área de inteligencia artificial.

En el área de bases de datos las reglas que se utilizan para especificar estas situaciones y sus acciones se llaman *reglas del tipo evento-condicion-accion* o *reglas que siguen el paradigma de evento-condicion-accion*. El formato genérico de estas reglas es:

```
ON      evento
IF      condicion
THEN   accion
```

El language de reglas de este tipo debe tener componentes para especificar eventos, especificar condiciones y especificar acciones.

Adicionalmente a la especificación de las reglas, es importante conocer cómo es la *semántica de la ejecución de las reglas*. En particular, existen tres aspectos relevantes en la forma en cómo se ejecutan las reglas, que describen el dinamismo de una base de datos, a saber:

1. Granularidad del procesamiento de las reglas.
Es importante saber si una regla se dispara una vez por cada tupla "tocada" o una sola vez por todas las tuplas "tocadas". También hay otras opciones, por ejemplo al final de la transacción que se está ejecutando cuando se despertó la regla.
2. Anidamiento de reglas y terminación.
Otro aspecto importante es si se puede especificar una sola regla por evento o más de una. En el caso en el que se permita más de una regla por evento, en cuál orden se ejecutan esas reglas y cuándo se termina la ejecución, son otros aspectos a considerar.
3. Concurrencia con las transacciones.
Finalmente, es necesario determinar si las reglas se van a ejecutar como parte de la transacción donde se disparen o si se van a ejecutar como transacciones aparte. Esto es fundamental por la propiedad de atomicidad que se debe garantizar para las transacciones de una base de datos.

2. Aplicaciones de las BD activas

Las aplicaciones del paradigma de base de datos activas son muy variadas. Una primera clasificación de las aplicaciones lo establece el uso de las reglas para labores internas del DBMS o para labores externas, las cuales son especificadas por el usuario y permiten realizar labores específicas dependientes del dominio del problema. Algunos ejemplos de las actividades que se pueden realizar en estas aplicaciones se muestran a continuación.

Internas. Soportar las características clásicas del manejo o administración de las bases de datos. Ejemplos de estas aplicaciones son:

1. Control de integridad. (Restricciones implícitas y explícitas.)
2. Mantenimiento de vistas y datos derivados, los cuales pueden existir virtualmente o ser materializados.
3. Administración de copias de los datos (duplicación).
4. Seguridad. Recuperación ante fallas.

Existen otras aplicaciones internas potenciales, pues hasta el momento no han sido explotadas por los DBMS, entre ellas se encuentran: mantenimiento de versiones, administración de la seguridad, “tracking” de eventos, bitácoras.

Externas. Estas aplicaciones contienen conocimiento de la aplicación, expresado en la forma de reglas, a las cuales comúnmente se les llama *reglas del negocio*.

Con respecto al *control de integridad* las restricciones que se pueden establecer con las reglas activas son:

- Restricciones estáticas: se evalúan sobre un estado de la base de datos, un ejemplo de las cuales son las restricciones de dominio.
- Restricciones dinámicas: se evalúan sobre la transición de un estado a otro, por ejemplo: el sueldo de un empleado solo puede aumentar.

Independientemente de si las restricciones son estáticas o dinámicas, dependiendo de quién las especifica, se pueden dividir en:

- restricciones “built-in”: son fijas y especificadas con cláusulas del DDL, por ejemplo: referential integrity (foreign keys, REFERENCES) y claves primarias (PRIMARY KEY), o
- genéricas: especificadas por el usuario, por ejemplo con la definición de CONSTRAINTS; algunos ejemplos de éstos son: NOT NULL, UNIQUE y CHECK.

3. El estándar de SQL y las bases de datos activas

El primer estándar de SQL donde se consideró la noción de ser activa de una base de datos fue en SQL:99 (es decir, en SQL 3); para el estándar que salió en 1992 no se incluyó este aspecto porque ya era muy extenso el estándar y quienes lo desarrollaron pensaron, equivocadamente, que los implementadores de DBMS no iban a implementar ese aspecto en los manejadores por un tiempo. Sin embargo, los desarrolladores si implementaron este aspecto, pero como no había estándar, cada uno realizó su implementación independientemente, obteniendo así una gran diversidad de implementaciones de triggers, específicamente. La descripción de triggers y assertions si se encuentra en SQL:99, por lo tanto, ahora se quiere tratar de uniformizar todas las implementaciones existentes.

En síntesis, los aspectos de base de datos activas en el estándar SQL:99 se concentran en los siguientes aspectos: constraints, triggers y assertions. A continuación se presenta una breve descripción de cada uno de estos aspectos.

Constraints. Son especificaciones del DDL de SQL que se aplican a columnas o tuplas de una tabla. Algunos tipos especiales de *constraints* son: UNIQUE, NOT NULL, REFERENCES, CHECK; éste último permite especificar una amplia gama de reglas, como por ejemplo, rangos de valores y listas de valores, entre otros. Es conveniente que cada *constraint* tenga un nombre, pues cuando el *constraint* es violado el sistema indica su nombre y se puede saber exactamente qué falló.

La verificación de las restricciones y el hacer que se cumplan se puede hacer de manera *inmediata* (constraint check time IMMEDIATE) durante la ejecución de la transacción o de forma *diferida* (constraint check time DEFERRED), es decir, a tiempo de compromiso de la transacción (commit).

Los constraints pueden ser violados solo por una o más tuplas de una tabla y no por la tabla en sí misma, es decir, que una tabla vacía cumple con todos los constraints, una tabla vacía no viola restricción alguna.

Assertions. Es un tipo de restricción especial que se puede especificar en SQL sin que deba estar asociada a una tabla en particular, como es el caso de los *constraints*. Generalmente se utilizan para describir restricciones que afectan a más de una tabla.

Como los constraints sólo se pueden establecer sobre tuplas de una tabla, los assertions son útiles cuando es necesario especificar una condición general de la base de datos que no se puede asociar a una tabla específica de la base de datos. Por esta característica de poder especificar un *assertion* para toda la base de datos y no para una tabla en particular, también se les llama *standalone constraints*. Esto tiene grandes ventajas a nivel conceptual, pero las hace difíciles de implementar.

Triggers Los *triggers* en SQL:99 son parecidos a los *constraints*, pero proveen mayor flexibilidad pues el usuario puede especificar un conjunto de acciones complejas a ejecutarse cuando el *trigger* se *dispare*. Los *constraints* actúan cuando se viola lo que ellos indican, pero el usuario no puede especificar acción alguna con esa violación, sino que el DBMS simplemente envía un mensaje de error e impide que se realice la operación que produjo la violación.

La descripción de un *trigger* contiene ocho (8) componentes, a saber:

Nombre, debe ser único en el esquema donde se define.

Tabla sujeto, es el nombre de la tabla cuyos datos, al ser alterados, van a activar al *trigger*.

Tiempo de la acción, le dice al DBMS cuándo ejecutar el cuerpo del trigger, lo cual puede ser antes (BEFORE) o después (AFTER) que ocurra el evento del *trigger*.

Evento, indica cuál es el comando de actualización sobre la tabla, que va a activar el *trigger*, este evento puede ser uno o más de los siguientes: INSERT, DELETE y UPDATE.

Lista de atributos, sólo se pueden especificar si el evento es un UPDATE.

Alias, tanto los valores de la tupla antes y después de la actualización, como las tablas pueden tener nombres que fungan de sinónimos de los nombres originales.

Cuerpo del trigger, comandos de SQL que se ejecutan cuando el *trigger* se dispara.

Sello de tiempo, indica cuándo fue creado el trigger.

Los *triggers* se definen con un comando de CREATE TRIGGER y para borrarlos se utiliza el comando DROP TRIGGER.