# Tema 1. Bases de datos activas

Diseño de Sistemas de Bases de Datos
Merche Marqués

18 de marzo de 2002

## Índice

# 1. Introducción

En muchas aplicaciones, la base de datos debe evolucionar independientemente de la intervención del usuario como respuesta a un suceso o una determinada situación. En los sistemas de gestión de bases de datos tradicionales (pasivas), la evolución de la base de datos se programa en el código de las aplicaciones, mientras que en los *sistemas de gestión de bases de datos activas* esta evolución es autónoma y se define en el esquema de la base de datos.

El poder especificar reglas con una serie de acciones que se ejecutan automáticamente cuando se producen ciertos eventos, es una de las mejoras de los sistemas de gestión de bases de datos que se consideran de gran importancia desde hace algún tiempo. Mediante estas reglas se puede hacer respetar reglas de integridad, generar datos derivados, controlar la seguridad o implementar reglas de negocio. De hecho, la mayoría de los sistemas relacionales comerciales disponen de *disparadores* (*triggers*). Se ha hecho mucha investigación sobre lo que debería ser un modelo general de bases de datos activas desde que empezaron a aparecer los primeros disparadores. El modelo que se viene utilizando para especificar bases de datos activas es el *modelo evento–condición–acción*.

Mediante los sistemas de bases de datos activas se consigue un nuevo nivel de independencia de datos: la independencia de conocimiento. El conocimiento que provoca una reacción se elimina de los programas de aplicación y se codifica en forma de *reglas activas*. De este modo, al encontrarse las reglas definidas como parte del esquema de la base de datos, se comparten por todos los usuarios, en lugar de estar replicadas en todos los programas de aplicación. Cualquier cambio sobre el comportamiento reactivo se puede llevar a cabo cambiando solamente las reglas activas, sin necesidad de modificar las aplicaciones.

Además, mediante los sistemas de bases de datos activas se hace posible el integrar distintos subsistemas (control de accesos, gestión de vistas, etc.) y se extiende el ámbito de aplicación de la tecnología de bases de datos a otro tipo de aplicaciones.

Uno de los problemas que ha limitado el uso extensivo de reglas activas, a pesar de su potencial para simplificar el desarrollo de bases de datos y de aplicaciones, es el hecho de que no hay técnicas fáciles de usar para diseñar, escribir y verificar reglas. Por ejemplo, es bastante difícil verificar que un conjunto de reglas es consistente, es decir, que no se contradice. También es difícil garantizar la terminación de un conjunto de reglas bajo cualquier circunstancia. Para que las reglas activas alcancen todo su potencial, es necesario desarrollar herramientas para diseñar, depurar y monitorizar reglas activas que puedan ayudar a los usuarios en el diseño y depuración de sus reglas.

# 2. El modelo evento–condición–acción

Un sistema de bases de datos activas es un sistema de gestión de bases de datos (SGBD) que contiene un subsistema que permite la definición y la gestión de reglas de producción

(reglas activas). Las reglas siguen el modelo evento–condición–acción (modelo ECA): cada regla reacciona ante un determinado evento, evalúa una condición y, si ésta es cierta, ejecuta un acción. La ejecución de las reglas tiene lugar bajo el control de un subsistema autónomo, denominado motor de reglas, que se encarga de detectar los eventos que van sucediendo y de planificar las reglas para que se ejecuten.

En el modelo ECA una regla tiene tres componentes:

- El *evento* (o eventos) que dispara la regla. Estos eventos pueden ser operaciones de consulta o actualización que se aplican explícitamente sobre la base de datos. También pueden ser eventos temporales (por ejemplo, que sea una determinada hora del día) u otro tipo de eventos externos (definidos por el usuario).
- La *condición* que determina si la acción de la regla se debe ejecutar. Una vez ocurre el evento disparador, se puede evaluar una condición (es opcional). Si no se especifica condición, la acción se ejecutará cuando suceda el evento. Si se especifica condición, la acción se ejecutará sólo si la condición se evalúa a verdadero.
- La *acción* a realizar puede ser una transacción sobre la base de datos o un programa externo que se ejecutará automáticamente.

Casi todos los sistemas relacionales incorporan reglas activas simples denominadas disparadores (*triggers*), que están basados en el modelo ECA:

- Los eventos son sentencias SQL de manejo de datos (`INSERT, DELETE, UPDATE`).
- La condición (que es opcional) es un predicado booleano expresado en SQL.
- La acción es un secuencia de sentencias SQL, que pueden estar inmersas en un lenguaje de programación integrado en el producto que se esté utilizando (por ejemplo, PL/SQL en Oracle).

El modelo ECA se comporta de un modo simple e intuitivo: *cuando* ocurre el evento, *si* la condición es verdadera, *entonces* se ejecuta la acción. Se dice que el disparador es *activado* por el evento, es *considerado* durante la verificación de su condición y es *ejecutado* si la condición es cierta. Sin embargo, hay diferencias importantes en el modo en que cada sistema define la activación, consideración y ejecución de disparadores.

Los disparadores relacionales tienen dos niveles de *granularidad*: a nivel de fila y a nivel de sentencia. En el primer caso, la activación tiene lugar para cada tupla involucrada en la operación y se dice que el sistema tiene un comportamiento orientado a tuplas. En el segundo caso, la activación tiene lugar sólo una vez para cada sentencia SQL, refiriéndose a todas las tuplas invocadas por la sentencia, con un comportamiento orientado a conjuntos. Además, los disparadores tienen funcionalidad *inmediata* o *diferida*. La evaluación de los disparadores inmediatos normalmente sucede inmediatamente después del evento que lo activa (opción

*después*), aunque también puede precederlo (opción *antes*) o ser evaluados en lugar de la ejecución del evento (opción *en lugar de*). La evaluación diferida de los disparadores tiene lugar al finalizar la transacción en donde se han activado (tras la sentencia `COMMIT`).

Un disparador puede activar otro disparador. Esto ocurre cuando la acción de un disparador es también el evento de otro disparador. En este caso, se dice que los disparadores se activan en cascada.

## 2.1. Definición y uso de disparadores en Oracle

La sintaxis para la creación de los disparadores en Oracle es la siguiente:

```
CREATE TRIGGER NombreDisparador
        Modo Evento
        ON TablaDestino
        [REFERENCING Referencia]
        [FOR EACH Nivel]
        [WHEN PredicadoSQL]
        BloquePL/SQL
```

- El `Modo` indica si la evaluación se debe realizar antes o después del evento que lo activa, por lo que sus valores pueden ser `BEFORE` o `AFTER`.
- El `Evento` es la operación del lenguaje de manejo de datos que activa el disparador y puede ser `INSERT`, `DELETE` o `UPDATE`. El evento `UPDATE` puede ir seguido de nombres de columnas de la tabla destino: `UPDATE OF col1, col2, ...` Oracle permite especificar como evento cualquier combinación de estas tres operaciones de manejo de datos (`INSERT`, `DELETE` y `UPDATE`) utilizando el operador booleano `OR`.
- La cláusula `REFERENCING` permite la introducción de nombres de variables para hacer referencia a los valores antiguos y nuevos de la fila a la que afecta el evento, con la siguiente sintaxis:

  ```
  OLD AS VarAntigua
  NEW AS VarNueva
  ```

  Si no se utiliza esta cláusula, los nombres por defecto son `OLD` y `NEW`.
- La granularidad de los disparadores se determina mediante la cláusula `FOR EACH`, donde `Nivel` puede ser `ROW` o `STATEMENT`, indicando que es a nivel de fila o a nivel de sentencia. Si se omite, el valor por defecto es `FOR EACH STATEMENT`.

- La condición (`PredicadoSQL`) sólo puede estar presente en los disparadores a nivel de fila y consiste en un predicado simple sobre la fila actual: sólo si el predicado es verdadero, se ejecuta el bloque de la acción. Este predicado no puede contener subconsultas ni llamadas a funciones definidas por el usuario, sólo se puede hacer referencia a los parámetros del evento. Los disparadores con granularidad a nivel de sentencia pueden, sin embargo, sustituir la condición por estructuras de control en la parte de la acción.
- La acción, en ambas granularidades, se escribe en PL/SQL, un lenguaje proporcionado por Oracle que extiende el lenguaje SQL añadiéndole las construcciones típicas de un lenguaje de programación. La parte de la acción no puede contener sentencias de definición de datos ni de transacciones.

Las referencias a los estados anterior (`OLD`) y posterior (`NEW`) de la fila que modifica el evento sólo se pueden utilizar en los disparadores a nivel de fila, tanto en la condición del `WHEN` como en el bloque de PL/SQL correspondiente a la acción. Cuando se hace una inserción sólo se define el estado posterior y cuando se hace un borrado, sólo se define el estado anterior. Las variables `OLD` y `NEW` están disponibles de modo implícito para indicar los valores antiguo y nuevo de una tupla. Con `NEW` se hace referencia a la tupla insertada o actualizada, mientras que `OLD` se utiliza para referirse a la tupla borrada o a la tupla antes de ser actualizada. Si la regla es `BEFORE` y `FOR EACH ROW`, en el bloque de PL/SQL se puede hacer una asignación sobre los atributos de `NEW`.

Los disparadores de Oracle son inmediatos y permiten las opciones `BEFORE` o `AFTER`, tanto si son a nivel de fila como a nivel de sentencia. Combinando ambas funciones y ambas granularidades se obtienen cuatro combinaciones para cada evento:

```
BEFORE ROW
BEFORE STATEMENT
AFTER ROW
AFTER STATEMENT
```

La ejecución de las sentencias `INSERT`, `DELETE` y `UPDATE` de SQL se entremezclan con la ejecución de los disparadores que ellas mismas activan siguiendo este algoritmo:

1. Se consideran los disparadores de nivel `BEFORE STATEMENT` y se ejecutan.
2. Para cada fila de la tabla a la que afecta la sentencia:
   *a*) Se consideran los disparadores de nivel `BEFORE ROW`[1] y se ejecutan.

[1] En este tipo de disparadores se puede hacer una asignación sobre `NEW` en el bloque de la acción.

   *b*) La sentencia se aplica a la fila y a continuación se realizan las comprobaciones de la integridad que se hayan especificado (`CHECK`).
   *c*) Se consideran los disparadores de nivel `AFTER ROW` y se ejecutan.

3. Se llevan a cabo las comprobaciopnes de la integridad especificadas para la tabla.
4. Se consideran los disparadores de nivel `AFTER STATEMENT` y se ejecutan.

Si ocurre algún error durante la evaluación de un disparador, se deshacen todas las modificaciones llevadas a cabo como consecuencia de la sentencia SQL que ha activado el disparador. Oracle garantiza un *rollback* de la sentencia y de todas las acciones llevadas a cabo por los disparadores que ésta ha activado.

Versiones anteriores de Oracle imponían el límite de poder definir sólo un disparador de cada tipo (`BEFORE/AFTER`, `ROW/STATEMENT`); las versiones actuales ya no tienen esta limitación, pero no tienen ningún mecanismo que permita priorizar los disparadores del mismo tipo que se activan por el mismo evento. Su consejo, cuando hay varios disparadores del mismo tipo para el mismo evento, es combinarlos todos en uno solo, de modo que se pueda establecer el orden en que se han de ejecutar las operaciones de las acciones de los distintos disparadores.

Además de `BEFORE` y `AFTER`, Oracle permite utilizar `INSTEAD OF` pero solamente en el caso en que la tabla destino sea una vista (`BEFORE` y `AFTER` no se pueden especificar sobre vistas). En este caso, se ejecuta el código del disparador en lugar de la sentencia sobre la vista. Por defecto, se activa para cada fila y no puede contener la cláusula `WHEN`.

Cuando se crea un disparador, éste está habilitado. Los disparadores pueden ser deshabilitados (`DISABLE`) y volver a ser habilitados (`ENABLE`). Mientras un disparador está deshabilitado no se activa.

En la acción que se especifica mediante el bloque de PL/SQL se pueden utilizar condiciones especiales para ejecutar secciones específicas dependiendo del tipo de evento que ha activado el disparador:

- `INSERTING` es verdadero si el disparador ha sido activado por una sentencia `INSERT`.
- `DELETING` es verdadero si el disparador ha sido activado por una sentencia `DELETE`.
- `UPDATING` es verdadero si el disparador ha sido activado por una sentencia `UPDATE`.
- `UPDATING(col)` es verdadero si el disparador ha sido activado por una sentencia `UPDATE` que actualiza la columna `col`.

El siguiente ejemplo muestra el uso de los disparadores de Oracle con un problema de gestión de almacén. El disparador `TrigPedido` se utilizará para generar una nueva orden de pedido automáticamente, insertando una fila en la tabla `Pedidos`, cuando la cantidad en almacén de una determinada pieza, `CantDisp`, caiga por debajo del límite mínimo (`CantLim`):

```
CREATE TRIGGER TrigPedido
AFTER UPDATE OF CantDisp ON Almacen
FOR EACH ROW
WHEN (NEW.CantDisp < NEW.CantLim)
    declare
        X number;
    begin
        SELECT COUNT(*) INTO X
        FROM Pedidos
        WHERE CodPieza = :NEW.CodPieza
        if X = 0
        then
            INSERT INTO Pedidos
            VALUES (:NEW.CodPieza, :NEW.CantPedido, SYSDATE);
        end if;
    end;
```

## 2.2. Procesamiento de reglas activas

Hay dos algoritmos alternativos para el procesamiento de las reglas activadas por una sentencia: al algoritmo iterativo y al algoritmo recursivo. Ambos se detallan a continuación.

**Algoritmo Iterativo**

**mientras** existan reglas activadas:
1. seleccionar una regla activada R
2. comprobar la condición de R
3. si la condición es cierta, ejecutar la acción de R

**fin mientras**

**Algoritmo Recursivo**

**mientras** existan reglas activadas:
1. seleccionar una regla activada R
2. comprobar la condición de R
3. si la condición es cierta
   3.1. ejecutar la acción de R
   3.2. ejecutar este algoritmo para las reglas activadas por la acción de R

**fin mientras**

En el modelo de ejecución de reglas de Oracle, el tipo de procesamiento es recursivo. El orden en que se van seleccionando las reglas de entre el conjunto de reglas activadas es indeterminado para disparadores del mismo tipo. La terminación del algoritmo de ejecución de reglas se asegura estableciendo un límite máximo al número de reglas disparadas durante la ejecución del algoritmo (normalmente es 32).

Cuando se trabaja con Oracle, en el procesamiento de una regla se puede producir un error a causa de una tabla mutante (*mutating*). Una tabla mutante es un tabla que está siendo modificada por una sentencia SQL (`INSERT, DELETE, UPDATE`) o por el efecto de un `DELETE CASCADE` asociado a la sentencia SQL.

Las restricciones que plantean las tablas mutantes son las siguientes:

- El bloque PL/SQL de una regla `FOR EACH ROW` no puede consultar ni actualizar una tabla mutante para su evento.
- El bloque PL/SQL de una regla `FOR EACH STATEMENT` activada como efecto de un `DELETE CASCADE` no puede consultar ni actualizar una tabla mutante para su evento.

Existe una excepción: las reglas `FOR EACH ROW` activadas por un `INSERT` que inserta una única fila. En estos casos no se da el error de tabla mutante.

Además, Oracle también define lo que considera una tabla restringida (*restricted*). Es una tabla que puede ser consultada debido a la ejecución de una sentencia SQL, por ejemplo por un `INSERT INTO ... SELECT ...` o por la comprobación de la integridad referencial. Las restricciones que plantean las tablas restringidas son las siguientes:

- El bloque PL/SQL de una regla `FOR EACH ROW` no puede modificar columnas definidas como `PRIMARY KEY`, `UNIQUE` o `FOREIGN KEY`.
- El bloque PL/SQL de una regla `FOR EACH STATEMENT` activada como efecto de un `DELETE CASCADE` no puede modificar columnas definidas como `PRIMARY KEY`, `UNIQUE` o `FOREIGN KEY`.

Existe también una excepción: las reglas `FOR EACH ROW` activadas por una sentencia `INSERT` que sólo inserta una tupla.

## 2.3. Características de las reglas activas

Además de las características que poseen los disparadores que incorporan los sistemas relacionales, algunos sistemas más avanzados y algunos prototipos de bases de datos activas ofrecen algunas características que incrementan la expresividad de las reglas activas:

- Respecto a los eventos, éstos pueden ser temporales o definidos por el usuario. Los eventos temporales permiten utilizar expresiones dependientes del tiempo, como por ejemplo: cada viernes por la tarde, a las 17:30 del 29/06/2002. Los eventos definidos por el usuario son eventos a los que el usuario da un nombre y que son activados por los programas de usuario. Por ejemplo, se podría definir el evento de usuario 'nivel-alto-azufre' y que una aplicación lo activara; esto activaría la regla que reacciona al evento.
- La activación de los disparadores puede que no dependa de un solo evento sino que dependa de un conjunto de eventos relacionados en una expresión booleana que puede ser una simple disyunción o una combinación más compleja que refleje la precedencia entre eventos y la conjunción de eventos.
- La consideración y/o ejecución de reglas se puede retrasar. En este caso, la consideración y/o la ejecución tienen lugar durante transacciones distintas, que pueden ser completamente independienes o pueden estar coordinadas con la transacción en la que se ha verificado el evento.
- Los conflictos entre reglas que se activan por el mismo evento se pueden resolver mediante prioridades explícitas, definidas directamente por el usuario cuando se crea la regla. Se pueden expresar como una ordenación parcial (utilizando relaciones de precendencia entre reglas), o como una ordenación total (utilizando prioridades numéricas). Las prioridades explícitas sustituyen a los mecanismos de prioridades implícitos que poseen los sistemas.
- Las reglas se pueden organizar en conjuntos y cada conjunto se puede habilitar y deshabilitar independientemente.

## 3. Propiedades de las reglas activas

No es difícil diseñar reglas activas de modo individual, una vez se han identificado claramente el evento, la condición y la acción. Sin embargo, entender el comportamiento colectivo de las reglas activas es más complejo ya que su interacción suele ser sutil. Por este motivo, el problema principal en el diseño de las bases de datos activas está en entender el comportamiento de conjuntos complejos de reglas. Las propiedades principales de estas reglas son terminación, confluencia e idéntico comportamiento observable.

- Un conjunto de reglas garantiza la *terminación* cuando, para cada transacción que puede activar la ejecución de reglas, esta ejecución produce un estado final en un número finito de pasos.
- Un conjunto de reglas garantiza la *confluencia* cuando, para cada transacción que puede activar la ejecución de reglas, la ejecución termina produciendo un estado final único que no depende del orden de ejecución de las reglas.

- Un conjunto de reglas garantiza un *comportamiento observable idéntico* cuando, para cada transacción que puede activar la ejecución de reglas, esta ejecución es confluyente y todas las acciones visibles llevas a cabo por la regla son idénticas y producidas en el mismo orden.

Estas propiedades no tienen la misma importancia. Concretamente, la terminación es una propiedad esencial; se debe evitar la situación en que las transacciones, activadas por el usuario, causan ejecuciones infinitas por la activación recursiva de reglas. Por otra parte, la confluencia y el idéntico comportamiento observable no son esenciales.

El proceso del *análisis de reglas* permite la verificación de si las propiedades deseadas se cumplen en un conjunto de reglas. Una herramienta esencial para verificar la terminación es el *grafo de activación*, que representa interacciones entre reglas. El grafo se crea incluyendo un nodo para cada regla y un arco de la regla $R_1$ a la regla $R_2$ cuando la acción de $R_1$ contiene un sentencia del lenguaje de manejo de datos que es también uno de los eventos de $R_2$. Una condición necesaria para la no terminación es la presencia de ciclos en el grafo de activación: sólo en este caso podemos tener una secuencia infinita de ejecución de reglas.

Los sistemas que tienen muchas reglas activas suelen ser cíclicos. Sin embargo, sólo unos pocos ciclos son los que provocan situaciones críticas. De hecho, el que un grafo sea cíclico es condición necesaria pero no suficiente para la no terminación.

## 4. Aplicaciones de las bases de datos activas

Las aplicaciones clásicas de las reglas activas son internas a la base de datos: el gestor de reglas activas trabaja como un subsistema del SGBD implementando algunas de sus funciones. En este caso, los disparadores son generados por el sistema y no son visibles por parte de los usuarios. La característica típica de las aplicaciones internas es la posibilidad de dar una especificación declarativa de las funciones, a partir de la que derivar las reglas activas. Ejemplos de ello son el mantenimiento de la integridad referencial (`FOREIGN KEY`) y el mantenimiento de restricciones de integridad (`CHECK`).

Por ejemplo, cuando una clave ajena es compuesta, la regla de integridad referencial dice lo siguiente: "si en una relación hay alguna clave ajena, sus valores deben coincidir con valores de la clave primaria a la que hace referencia, o bien, deben ser completamente nulos." En Oracle es el propio sistema quien se encarga de hacer respetar la primera parte de esta regla, una vez que el usuario ha definido las claves ajenas:

```
FOREIGN KEY (ColX,ColY) REFERENCES Tabla
```

Sin embargo, la segunda parte de la regla se debe hacer respetar mediante una restricción:

```
CHECK( (ColX IS NULL AND ColY IS NULL) OR
       (ColX IS NOT NULL AND ColY IS NOT NULL) )
```

En este sistema, la condición de las restricciones expresadas mediante la cláusula `CHECK` debe cumplir lo siguiente:

- debe ser una expresión booleana que se pueda evaluar usando los valores de la fila que se inserta o que se actualiza;
- no puede contener subconsultas;
- no puede incluir las funciones `SYSDATE, UID, USER, USERENV`;

lo que hace que en muchas ocasiones no se puedan establecer restricciones de integridad mediante esta cláusula y sea necesario el uso de disparadores. En el siguiente ejemplo se muestra un disparador que se encarga de mantener los salarios de los empleados dentro del rango establecido para el puesto que ocupan.

```
CREATE TRIGGER TrigCompruebaSalario
BEFORE INSERT OR UPDATE OF Salario, Puesto ON Emp
FOR EACH ROW
    declare
        minsal number;
        maxsal number;
        salario_fuera_rango exception;
    begin
        SELECT Minsal, Maxsal INTO minsal, maxsal
        FROM Salarios
        WHERE Puesto = :NEW.Puesto;
        if (:NEW.Salario < minsal or :NEW.Salario > maxsal)
        then raise salario_fuera_rango;
        end if;
    exception
        when salario_fuera_rango
            then raise_application_error(-20300,'Salario '||
            to_char(:NEW.Salario)||' fuera de rango para el puesto '
            ||:NEW.Puesto||' del empleado '||:NEW.Nombre);
    end;
```

También se pueden utilizar reglar activas para mantener datos derivados, como puede ser el importe total de una factura o la nota media del expediente de un estudiante. Una aplicación similar es la de utilizar reglas activas para mantener la consistencia de las vistas materializadas cuando cambian los datos de las relaciones base sobre las que están definidas. Esta aplicación tiene más relevancia cuando se piensa en la tecnología de los grandes almacenes de datos (*data warehousing*). Y otra aplicación también relacionada es el mantenimiento de la consistencia de tablas replicadas, especificando reglas que modifiquen las réplicas cuando las tablas originales son modificadas.

Una aplicación importante es el permitir la notificación de que está ocurriendo algún suceso de interés. Por ejemplo, se puede utilizar un sistema de bases de datos activas para monitorizar la temperatura de un horno industrial. La aplicación puede insertar periódicamente en la base de datos las lecturas de los sensores de tempertatura y se pueden crear reglas que se activen cuando se alcancen niveles peligrosos, disparando una alarma.

También se pueden utilizar reglas activas para mantener la seguridad y para realizar auditorías sobre el acceso a los datos.

Otra aplicación de las bases de datos activas es el mantenimiento de otras reglas, clasificadas como externas, que expresan conocimiento específico de la aplicación y que están más allá de los esquemas predefinidos y rígidos. Estas reglas son las denominadas reglas de negocio ya que expresan las estrategias de una organización para llevar a cabo sus funciones primarias. En el caso de las reglas de negocio no hay técnicas de derivación de reglas basadas en las especificaciones. Es por ello que cada problema se debe afrontar por separado.

## 4.1. Ejemplo: gestión de restricciones de integridad

Hay dos formas de definir las restricciones de integridad: declarativa y operacional. La declarativa se realiza mediante SQL cuando se crean las tablas de la base de datos (`CREATE TABLE`). En este caso la comprobación de la integridad es responsabilidad del SGBD: analiza las operaciones relevantes y tiene su propia estrategia de comprobación. Para definir restricciones de forma operacional se utilizan las reglas activas. En este caso la comprobación de la integridad es responsabilidad del diseñador que define las reglas: analiza las operaciones relevantes (serán los eventos) y decide la estrategia de comprobación (serán la condición y las acciones).

La gestión de las restricciones integridad mediante el uso de reglas activas requiere que primero se expresen las restricciones en forma de predicado SQL. El predicado corresponderá a la parte de la condición de una o más reglas activas asociadas a la restricción; hay que notar, sin embargo, que el predicado debe aparecer negado en la regla, de modo que su consideración lleva al valor verdadero cuando se viola la restricción. Después de esto, el diseñador se debe concentrar en los eventos que pueden originar la violación de la restricción. Estos eventos serán los que se incluirán en las reglas activas. Por último, el diseñador tendrá que decidir qué acción llevar a cabo cuando se viola la restricción. Por ejemplo, la

acción podría ser la de forzar un *rollback* parcial de la sentencia que ha causado la violación, o bien realizar alguna acción compensatoria que corrija la violación de la restricción.

A continuación se muestra a modo de ejemplo el problema de la integridad referencial (aunque la mayoría de los sistemas actuales gestionan automáticamente esta restricción). De la tabla `Empleado` a la tabla `Departamento` hay una clave ajena que indica el departamento al que pertenece el empleado. La especificación de esta restricción de integridad se puede realizar en la sentencia de creación de la tabla `Empleado` mediante la cláusula:

```
EXISTS (SELECT * FROM Departamento
         WHERE DeptNum = Empleado.DeptNum)
```

Esta aserción indica una propiedad que debe ser verdadera para todos los empleados, pero en una regla activa lo que interesa es capturar las situaciones que violan la restricción, por lo que se debe utilizar la negación de la aserción para construir la condición que se debe incluir en la regla activa:

```
NOT EXISTS (SELECT * FROM Departamento
             WHERE DeptNum = Empleado.DeptNum)
```

Por último se deberán construir cuatro reglas activas: dos de ellas reaccionarán a la inserción en `Empleado` y a la modificación de su atributo `DeptNum`, cancelando el efecto de las operaciones si violan la restricción (no hay que olvidar que según la definición de la integridad referencial, las operaciones que la violan se deben rechazar). Las otras dos reglas reaccionarán al borrado en `Departamento` y a la modificación de su clave primaria.

### Bibliografía

En el capítulo 23 del texto de Elmasri y Navathe (1999) se presentan los sistemas de bases de datos activas, con ejemplos basados en el sistema de gestión de bases de datos Oracle. Algunas de las aplicaciones de las bases de datos activas se estudian en otros capítulos, como las vistas materializadas o su uso en los grandes almacenes de datos (*data warehousing*). Atzeni et al. (1999) tratan las bases de datos activas en el capítulo 12, haciendo un estudio de los disparadores en Oracle y DB2.