UNIVERSIDAD CARLOS III DE MADRID
DEPARTAMENTO DE INFORMÁTICA

TESIS DOCTORAL

APLICACIÓN DE TÉCNICAS ACTIVAS PARA EL CONTROL DE RESTRICCIONES EN EL DESARROLLO DE BASES DE DATOS

AUTOR: Harith T. Al-Jumaily

DIRECTORAS: Dra. Paloma Martínez Fernández
Dra. Dolores Cuadra Fernández

Leganés, 2006

UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

APLICACIÓN DE TÉCNICAS ACTIVAS PARA EL CONTROL DE RESTRICCIONES EN EL DESARROLLO DE BASES DE DATOS

TESIS DOCTORAL

Harith T. Al-Jumaily

Madrid, 2006

Departamento de Informática

Escuela Politécnica Superior

Universidad Carlos III de Madrid

APLICACIÓN DE TÉCNICAS ACTIVAS PARA EL CONTROL DE RESTRICCIONES EN EL DESARROLLO DE BASES DE DATOS

AUTOR: Harith T. Al-Jumaily

DIRECTORAS: Dra. Paloma Martínez Fernández
Dra. Dolores Cuadra Fernández

D.12

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día ..... de ..... de 2006.

Presidente: D. .....

Vocal: D. .....

Vocal: D. .....

Vocal: D. .....

Secretario: D. .....

Realizado el acto de defensa y lectura de la Tesis el día ..... de .....

..... de 2006 en .....

Calificación: .....

PRESIDENTE VOCALES

EL SECRETARIO

Agradecimientos

Aunque mis palabras son incapaces de expresar mis sentimientos, yo quisiera en estas líneas dar mí gratitud absoluta a todas las personas que han contribuido para que este trabajo salga adelante. Deciros a todos, mi gratitud no es nada para corresponder vuestra generosidad.

En primer lugar quisiera agradecer a mis tutoras, Dra. Paloma Martínez Fernández y Dra. Dolores Cuadra Fernández, por todo el apoyo y la paciencia que me han ofrecido durante la realización de la tesis. Sus esfuerzos y sus orientaciones han sido de gran ayuda para iluminar el camino.

A mis compañeros y amigos, gracias por el apoyo que me habéis dado siempre y cuando os necesito. A Ana, César, Conchi, Elena, Ignacio, Javi, Josele, Lourdes, Manu, y otros tantos más, muchísimas gracias por todo.

A Manolo y Pepita, gracias por la ayuda y la confianza.

A mi querida familia, gracias por todo, os queremos.

A mi querida Ghaida, gracias, sin ti, esto no hubiese salido.

Harith

Resumen

En las metodologías de desarrollo de BD nos encontramos que los modelos conceptuales que aplicamos son cada vez más ricos y capaces de recoger con mayor precisión las especificaciones del dominio. El problema surge cuando queremos mantener esta semántica en la siguiente fase de la metodología, la fase lógica. La mayoría de estas metodologías utilizan el modelo relacional para transformar el esquema conceptual a un esquema lógico. Para ello, se aplican un conjunto de reglas de transformación. Estas reglas son, en general, muy básicas pues solo especifican la transformación de los elementos más simples y sencillos del modelo conceptual, esto implica una pérdida de semántica significativa, que lleva al diseñador a controlar y comprobar las restricciones de integridad del dominio fuera de la BD. Para representar las restricciones de integridad, el estándar SQL3 y la mayoría de los SGBDs comerciales proporcionan dos maneras: *mecanismos declarativos* y *mecanismos procedimentales*. Los mecanismos declarativos aportan una base fundamental y estrategia tradicional para reforzar las reglas básicas de negocio (*Not Null, Default, Unique, Primary Key, Foreign Key , Check, Domain y Assertion*). Sin embargo, estas cláusulas no son capaces de expresar todas las restricciones de integridad incluidas en un modelo conceptual, por eso, se utilizan mecanismos procedimentales como reglas activas/disparadores para poder controlar estas restricciones dentro del SGBD.

En este trabajo de tesis doctoral se propone incorporar reglas activas para preservar todas las restricciones de integridad asociadas a un dominio dentro de la BD y proporcionar a los diseñadores herramientas que faciliten la generación automática de estas reglas, el análisis de su comportamiento teniendo en cuenta las interacciones con el resto de elementos de la BD y su visualización.

El trabajo está enmarcado dentro del proyecto "Plataforma de gestión de procesos software: Modelado, reutilización y medición“. TIN2004/07083

Abstract

In database development methodologies, conceptual models are more abstract and concerned to express more accurately the semantics of the real world. A problem is produced when we want to maintain this semantics in the following phase of methodologies: the logical phase. Most of these methodologies are agree with the application of the relational model to transform the conceptual scheme into a logical scheme. To do that, a set of transformation rules is applied. These rules are basics and simples, they specify the transformation of the simplest conceptual elements and this implies a lost semantics. Therefore, databases developers must employ different mechanisms to enforce database consistency. To verify integrity constraints, the recent SQL standard and almost all commercial DBMSs support two ways: *declaratively mechanisms* and *procedural mechanisms*. Declarative mechanisms provide a fundamental base and traditional strategy to enforce business basic rules by using constraints such as: *Not Null, Default, Unique, Primary Key, Foreign Key, Check, Domain, and Assertion*. Because of these constructors are insufficient to express all integrity constraints, procedural mechanisms active rules/triggers are used in order to control these constraints inside the DBMS.

In this doctoral thesis work, incorporating active rules into database are proposed to preserve all integrity constraints associated. Moreover, we provide some tools to facilitate: automatic rules generation, rules analysis, and rules behaviour visualization considering the interactions with the reminder of elements.

This work is part of the project "Software Process Management Platform: modelling, reuse and measurement". TIN2004/07083

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

1. Motivación y Objetivos

1.1 Marco de Investigación

Este trabajo de tesis doctoral se enmarca dentro de la ingeniería de datos y más concretamente, en el desarrollo de Bases de Datos (BD). La realización de un buen análisis, diseño e implementación de una BD deberá estar apoyada en una metodología de desarrollo, que guíe al diseñador en el proceso a seguir para la obtención de un buen diseño que refleje el mundo real o el Universo del Discurso (UD). Aunque existen distintas metodologías de desarrollo de BD, todas ellas establecen unas fases a seguir bien definidas e incluso formalizadas. A continuación, pasamos a describir cada una de las fases más comunes de una metodología de desarrollo de BD, como se muestra en la figura 1.1:

- Análisis de requisitos. Se recogen las especificaciones que describen el problema del UD. Esta primera fase utiliza el lenguaje natural para recoger la especificación de requisitos y puede refinarse a través de sucesivas entrevistas con los expertos del dominio. Su principal objetivo es acotar o limitar el problema.
- Diseño conceptual. Es una descripción de alto nivel de la estructura de la BD, independientemente del Sistema Gestor Base de Datos (SGBD) que se vaya a utilizar para implementarla. Se describe el UD a través de un lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es

describir el contenido de información de la BD y no las estructuras de almacenamiento que se necesitarán para manejar esta información. Estos modelos poseen la característica de tener un alto nivel de abstracción y nos servirán para validar la modelización del UD con los expertos del dominio. Los modelos conceptuales deben ser intuitivos y ofrecer distintos mecanismos para recoger toda la semántica del problema. A través de esta fase tendremos representado el UD de forma, generalmente gráfica, al ser este el lenguaje que nos proporcionan la mayoría de los modelos de datos conceptuales.

- Diseño lógico. En esta fase de la metodología se transforma el esquema conceptual obtenido en la fase anterior a un esquema lógico, cuya principal característica consiste en que se compone de estructuras más cercanas a la implementación. Un esquema lógico es una descripción de la estructura de la BD que puede procesar un tipo de SGBD. Para llegar a esta fase se suelen aplicar un conjunto de reglas establecidas, donde se transforman esquemas conceptuales a lógicos. En la fase anterior no hemos especificado ningún modelo conceptual, sin embargo, en esta fase el modelo por excelencia es el objeto-relacional al ser uno de los modelos de datos más extendidos e implementados por los SGBDs comerciales.

- Diseño físico. Un esquema físico es una descripción de la implementación de una BD en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Por ello, el diseño físico depende del SGBD concreto y el esquema físico se expresa mediante su lenguaje de definición de datos. En esta fase no existe ningún modelo que nos sirva para describir el esquema físico de la BD, al no estar estandarizado, por lo que depende del conocimiento que el diseñador tenga del SGBD.

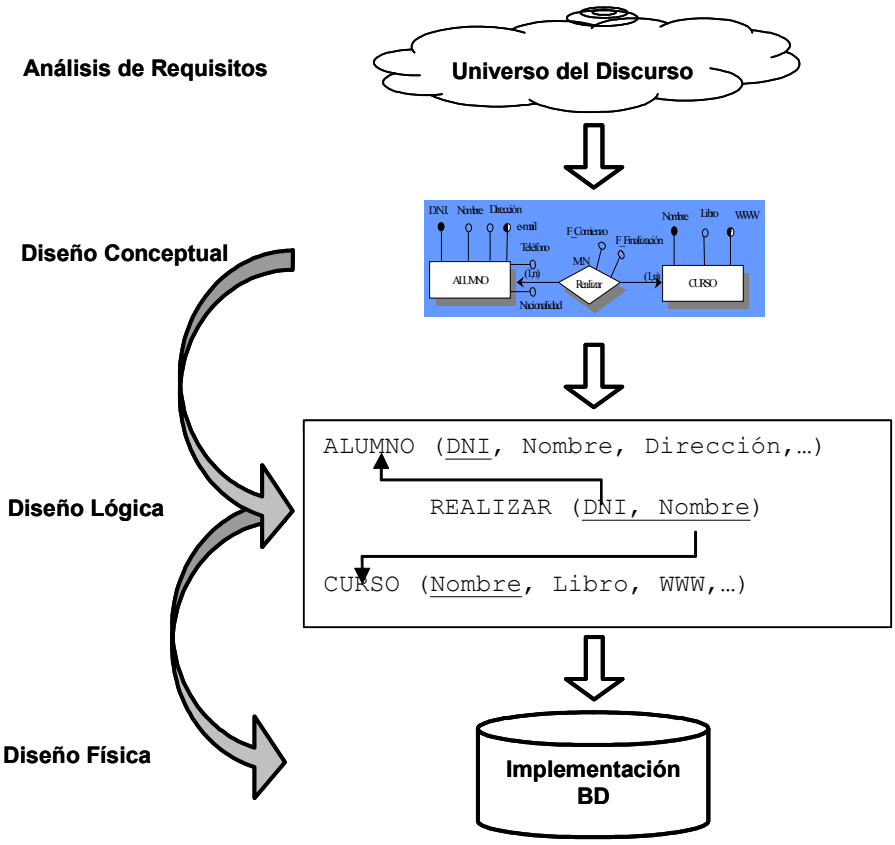


Figura 1.1. Las fases de las metodologías del desarrollo de BD

Aplicando estas fases se pretende obtener una BD consistente y que contemple la semántica del UD. Una BD consistente significa que los datos que se almacenan en ella reflejan la semántica deseada. En general, el problema viene dado porque tanto la elección de los modelos de datos como el paso por las distintas fases de la metodología influyen en la consistencia. Pasar de niveles abstractos a la implementación de la BD genera pérdidas semánticas que son reflejadas posteriormente a nivel de aplicación para asegurar la consistencia.

Para evitar que se pueda dar una pérdida de semántica (inconsistencias en los datos) en la BD, la mayoría de las veces se recurre a incluir algunas restricciones de integridad, como por ejemplo, restricciones de dominios, de claves, de integridad referencial, de integridad semántica, etc., para especificar las restricciones impuestas por el dominio del UD. Concretamente, un SGBD relacional proporciona dos

maneras para reforzar estas restricciones; mecanismos declarativos y especificación de procedimientos. Los mecanismos declarativos, tales como las cláusulas *Primary Key, Unique, Check, Foreign Key*, etc., en el modelo relacional, proporcionan una base fundamental y una estrategia tradicional para reforzar las restricciones de integridad básicas [Elmasri y Navathe, 2004]. Por eso, estos mecanismos no son capaces de especificar otras restricciones de integridad semánticas más complejas. Ejemplos de tales restricciones son: *el sueldo total de los empleados que trabajan en un departamento no debe exceder el presupuesto del departamento al que pertenecen* o *el sueldo de un empleado no debe exceder el salario de su supervisor*.

En este trabajo nos centramos en estudiar las restricciones de integridad presentando previamente una definición de las mismas y centrándonos en aquellas que los mecanismos declarativos no son capaces de verificar. Estas restricciones son de tipo *condición y acción general* y, *condición general* y *acciones específicas.* Las restricciones de tipo condición general y acción general son aquellas donde obligatoriamente se declara la condición por medio de una consulta lógica, permitiendo condiciones complejas arbitrarias y sus acciones, son de rechazo. Mientras, las restricciones de tipo condición general y acción específica son aquellas que obligatoriamente declaran la condición y la acción. Este tipo de restricciones especifican la acción siempre con un procedimiento y se dividen en dos tipos de mecanismos: *procedimientos almacenados y disparadores*. Los procedimientos almacenados pueden ser tan complejos como la semántica del mundo real (para la condición y para la acción). *Los disparadores*, en términos de la acción, pueden ser tan complejos como la semántica del mundo real (dependerá de las características proporcionadas por el lenguaje del SGBD), y la condición tan compleja como una consulta lógica.

1.2 Descripción del Problema

Llegados a este punto podemos especificar que esta propuesta de tesis doctoral está enmarcada en una metodología de desarrollo de BD relacionales y se pretende solucionar el problema de conservar las restricciones de integridad, especificadas en las primeras fases del desarrollo, a lo largo de todas las fases de la metodología. La conservación de las restricciones de integridad de una BD se considera un requisito fundamental para que su contenido refleje lo más posible la realidad del UD. Por eso, uno de los aspectos más importantes a la hora de realizar operaciones de actualización en los datos (inserción, borrado o modificación), es el cumplimiento de los requisitos especificados en la definición de estas restricciones. Durante el diseño conceptual de la BD se intenta realizar un gran esfuerzo para capturar la mayor semántica posible con el objetivo de representar fielmente las especificaciones del dominio. El problema surge cuando queremos mantener esta semántica en la siguiente fase de la metodología, la fase lógica, donde algunas de las metodologías coinciden en la utilización del modelo relacional para transformar el esquema conceptual en un esquema cercano a la implementación. Para ello, se aplican un conjunto de reglas de transformación. Estas reglas son, en general, muy básicas pues solo especifican la transformación de los elementos más simples y sencillos del modelo conceptual. Esto implica una pérdida de semántica significativa, que lleva al diseñador a controlar y comprobar las restricciones de integridad del dominio fuera de la BD.

Existen distintas aproximaciones para conservar las restricciones de integridad de la BD, una de estas aproximaciones es la utilización de la tecnología activa (reglas activas). Se ha elegida esta aproximación en esta tesis doctoral teniendo en cuenta la eficacia de la misma para conservar las restricciones de integridad. Al mismo tiempo, la complejidad de esta tecnología y la falta de herramientas para facilitar su desarrollo, causan un rechazo en su uso por parte de los diseñadores de BD,

especialmente, cuando se pueden sustituir aplicaciones tradicionales por estos mecanismos, y a veces se recomienda que las reglas activas no se usen con frecuencia [Simon y Kotz-Dittrich, 1995]. Este problema va aumentando conforme se intentan añadir cambios en la BD provenientes de cambios en las restricciones del dominio. Esta actualización puede exigir crear nuevas reglas o borrar otras antiguas, lo que conlleva la revisión del comportamiento del nuevo conjunto de reglas que pertenecen al sistema de BD.

Además de la complejidad de implementación, existen otros problemas como la no terminación y la activación en cascada de las reglas por la interacción con las acciones de integridad referencial. En el desarrollo de las reglas utilizando las herramientas CASE comerciales, (*Designer2000, ERwin, Rational Rose, Power Designer*..., etc.), se ha detectado que transformar una restricción de integridad a una regla activa es insuficiente ya que cada regla tiene un orden de ejecución dentro de su entorno que determina su comportamiento. El orden en la ejecución y el comportamiento tienen que ser entendidos y verificados por los desarrolladores de BD antes de aplicar estas reglas. Tales herramientas no pueden dar una visión adecuada sobre las conductas de estas reglas y las interacciones entre ellas con las acciones de integridad referencial.

Otro problema que surge de la utilización de mecanismos activos consiste en que la implementación exagerada de estos mecanismos para conservar las restricciones de integridad de la BD lleva consigo una complejidad de realización y un impacto sobre el rendimiento del SGBD. Existe una relación entre la semántica recogida de una BD, la implementación utilizada para controlar esta semántica y el rendimiento del SGBD reduciéndose éste último cada vez que se utiliza una implementación compleja para conservar o recoger mayor semántica del UD.

1.3 Propuesta de Solución

Los objetivos que perseguimos en la elaboración de la tesis están enmarcados dentro del proyecto GPS - Plataforma de Gestión de Procesos Software: Modelado, Reutilización y Medición (TIN-2004-07083) - el cual se dirige a la realización de una herramienta que ayude a los Ingenieros del Software a cumplir con las Prácticas Eficientes para el desarrollo de software, tal y como la han definido organismos tan importantes a nivel internacional como el ISO, SEI [SEI, 2006] o PMI [PMI, 2005]. Esto implica desarrollar lenguajes notacionales para la representación de procesos, un sistema para el almacenamiento y posterior recuperación avanzada de activos y artefactos de procesos, tales como productos de trabajo, planificaciones, informes de seguimiento, requisitos, riesgos, métricas, etc. Dentro del almacenamiento y la gestión del repositorio, en la que se apoyará la herramienta, se encuentra este trabajo de tesis doctoral. El cual aportará los aspectos necesarios para poder llevar a cabo esta tarea de manera eficaz y eficiente.

A continuación pasamos a enumerar los objetivos de este trabajo de tesis doctoral divididos para su presentación en cuatro partes:

- La primera se centra en resolver el problema de la pérdida de semántica cuando transformamos un esquema conceptual a un esquema relacional utilizando la tecnología activa. Para ello, se presentará una metodología que ampliará el conjunto de reglas de transformación a lo largo de las fases de diseño de la BD y con el objetivo principal de solucionar el problema de la pérdida de semántica dentro de la misma.
    - Esta solución parte de la transformación de las reglas de integridad formalizadas según el cálculo relacional de tuplas (*tuple relational calculus*) a disparadores en el estándar SQL3, de acuerdo a las reglas especificadas en la propuesta de esta tesis.
    - La transformación se apoyará en las reglas propuestas en [Ceri et al., 1994]

[Türker y Gertz, 2000] [Decker, 2002] y refinadas para este trabajo. Se aplicará este proceso para convertir las restricciones de integridad reflejadas en el esquema conceptual, y que no tienen una transferencia directa al modelo relacional, a disparadores del estándar SQL3 para controlar la semántica expresada en el esquema conceptual.

- La segunda consta de la presentación de casos de estudio representativos como aplicación de la metodología expuesta para la conservación de la semántica y la aportación de un conjunto de métricas para la mejora del rendimiento [Al-Jumaily et al., (a) 2004]. La importancia semántica se mide a través de un conjunto de parámetros que clasificarán las reglas según una serie de pesos. Las reglas con mayor peso serán activadas con mayor frecuencia que el resto.

- La tercera solucionará aquellos problemas asociados a la utilización de la tecnología activa. Para resolverlos se propone diseñar e implementar una herramienta que tiene como objetivo general automatizar las reglas de transformación aplicadas a lo largo de las distintas fases del desarrollo de una BD, incluyendo las propuestas definidas en este trabajo de tesis doctoral. Los objetivos específicos de esta herramienta son los siguientes:
  a) Implementar la transformación de las restricciones de integridad contempladas en un esquema conceptual.
    - La implementación incluye la transformación de los disparadores del estándar SQL3 a disparadores (SQL Script) directamente ejecutables en SGBDs comerciales, solucionando los problemas asociados con el modelo de ejecución de los mismos.
    - Visualización de los disparadores dentro del esquema con el objetivo de permitir a los desarrolladores acceder al disparador para modificar o añadir código.
  b) Permitir a los desarrolladores de la BD realizar fácilmente las tareas de mantenimiento del esquema conceptual, como por ejemplo, redefinir restricciones e integrar o quitar elementos, es decir, recrear y reformar los

disparadores según las novedades introducidas por los diseñadores.

c) Proporcionar un interfaz interactivo que permite a los desarrolladores acceder al esquema conceptual en cuestión y elegir múltiples opciones para controlar, tanto todo el esquema como parte de él, y también el tipo de operaciones de actualización que se quiere controlar INSERT, DELETE, y UPDATE.

- La cuarta y última, se dirige a solucionar el problema de la invisibilidad del orden de la ejecución y el comportamiento de las reglas activas, proponemos diseñar e implementar otra herramienta, que tiene como objetivo general visualizar y analizar su comportamiento. Los objetivos específicos de esta herramienta son los siguientes:

  a) Utilizar los componentes del Diagrama de Secuencia de UML para modular el proceso de ejecución de las reglas, visualizando la activación en cascada de ellas.

  b) Detectar el problema de la no terminación en la ejecución de los disparadores antes de implantarlos en el sistema.

  c) Mandar mensajes a los desarrolladores sobre el estado de la verificación.

1.4 Validación de la Propuesta

La validación de la propuesta vendrá dada por dos principios básicos dentro de la disciplina de las pruebas del Software: la verificación y la validación. El objetivo principal de la validación y la verificación es comprobar que el código generado por nuestra propuesta cumple las especificaciones para las que ha sido definido. Para ello, se presentarán algunas técnicas de pruebas del software para la verificación y la validación (V & V) de los disparadores, con el propósito de comprobar que no existen errores, ni producen inconsistencia en los resultados y satisfacen los requisitos de los usuarios. En este contexto, después de crear un escenario para la experimentación, se realizarán algunas pruebas como pruebas funcionales para las cuales se han elegido dos técnicas que son las clases de equivalencia y de aceptación, pruebas estructurales como el camino básico para comprobar la estructura de los disparadores, y finalmente, se realizará un estudio de rendimiento para comprobar la eficacia del SGBD cuando se ejecuta la propuesta. A continuación se explicarán estas pruebas:

- Pruebas funcionales: para la validación del conjunto de disparadores generados debemos demostrar que estos deben realizar las tareas planteadas por los usuarios, es decir, que realmente realizan la función para la que han sido creados. El objetivo específico de estas pruebas es garantizar que el conjunto de los disparadores generados por la herramienta controlan las restricciones de integridad de la BD.
- Pruebas del camino básico: para la verificación del conjunto de disparadores generados para preservar la semántica se utilizará una técnica denominada control de flujo. Se aplicará aplicaremos esta técnica para cubrir todos los caminos posibles que están asociados al código de una regla activa. Es decir, cuando se genera un evento que activa una regla, y en este momento, el resultado

final depende del camino tomado por la ejecución de la regla.

- Pruebas del rendimiento: para la validación será conveniente medir el rendimiento de un SGBD para comprobar su eficacia. En este caso se utilizará ORACLE para realizar estas pruebas. Se utilizará una BD, con los disparadores asociados para contemplar todo la semántica para realizar actualizaciones masivas de los datos monitorizando los parámetros del rendimiento (tiempo de la respuesta) del SGBD. Para comprobar el rendimiento de la BD se propone una referencia conveniente para la propuesta. En este caso, la referencia viene dado por el rendimiento de los procedimientos almacenados, a partir de la cual se ha podido analizar el rendimiento de los disparadores. Esto puede ser un apoyo para indicar o deducir si el rendimiento de la propuesta está en un marco aceptable.

1.5 Organización de la Memoria

A continuación, explicaremos de una forma breve la estructura de la memoria de esta tesis doctoral con el objetivo de un mejor entendimiento de la misma y para facilitar su uso en la búsqueda de un contenido determinado.

Los dos primeros bloques temáticos son la introducción y el estado de la cuestión donde se enmarca la tesis doctoral. En la introducción se expone de forma resumida el marco donde se engloba este trabajo de tesis doctoral, la descripción del problema a tratar, la propuesta para solucionar el problema, y su validación. El bloque del estado de la cuestión aborda el estudio del problema y las soluciones aportadas en el área por otros investigadores. Se expondrán las características de la tecnología activa, su tratamiento en el estándar SQL3 y su comportamiento en los sistemas de BD relacionales comerciales (por ser el marco de este trabajo de tesis doctoral), para finalizar con una discusión donde se presenta los problemas detectados que no han sido resueltos y que esta propuesta aborda.

El tercer bloque contiene la consecución de los objetivos planteados en esta tesis doctoral. En primer lugar se muestra la metodología propuesta para conservar las restricciones de integridad. Se especifican, en general, las restricciones de integridad y se convierten a disparadores del SQL3. Esta especificación y conversión a disparadores SQL3, toma una importancia relevante en algunos elementos conceptuales que serán objeto de estudio como: la interrelación binaria de tipo N:M y 1:N, así como la generalización, proponiendo soluciones para abordar la pérdida de semántica en estos elementos. Se tratará un caso especial con el fin de aportar de ciertas métricas para mejorar el rendimiento de la BD cuando existen mecanismos activos asociados.

En el cuarto bloque se presenta las carencias de las herramientas CASE comerciales, haciendo especial hincapié sobre las restricciones de cardinalidades y su verificación.

También, se muestra la arquitectura de la propuesta y la implementación de dos herramientas, la generadora y la visualizadora.

En el quinto bloque se demuestra la verificación y la validación de la propuesta, apoyadas en la aplicación de distintas técnicas de pruebas software.

Y por último, se presentan algunas conclusiones, difusión de los resultados obtenidos a partir de la consecución de la tesis y líneas futuras, recalcando los objetivos alcanzados previstos al comienzo del trabajo de tesis doctoral y presentando qué caminos se pueden seguir a partir de este trabajo.

Capítulo 2

2. Estado de la Cuestión

2.1 Abordando la Conservación de Restricciones de Integridad

Las restricciones de integridad garantizan que el contenido de la BD se conforme según las reglas establecidas para representar el Universo del Discurso. La integridad de una BD supone la existencia de dos componentes importantes que son la exactitud (*correctness*) y la completitud (*completeness*) [Motro, 1989], los cuales garantizan que todos los datos son correctos (válidos) y relevantes. Se considera que la tarea de asegurar estas dos propiedades es dificultosa porque todavía no existe ningún modelo de datos que puede capturar la semántica completa de un UD, o la BD puede aceptar datos que son válidos para algunos hechos, lo que podría causar la presencia de información que no tengan ninguna relación con el dominio de la aplicación. La falta de exactitud y completitud en las BDs puede llevar a deducir hechos que no son reales. Por ello, la mayoría de las BDs necesitan apoyarse en distintos mecanismos para vigilar y garantizar el contenido de la misma [Pacheco, 1997].

Se puede clasificar el mantenimiento de las restricciones de integridad de una BD según distintas estrategias. Tales estrategias describen cuándo se detecta la violación y cómo se mantiene la integridad de los datos. En general, estas estrategias son: (a) detectar de forma avanzada las operaciones de actualización que pueden violar las restricciones; rechazando la ejecución de tales operaciones [Ceri y Widom, 1990]. (b) Detectar la inconsistencia de la BD cuando ocurra, y restaurarla (ROLLBACK) a

través de rechazar todas las actualizaciones que han producido la violación [Urban y Delcambre, 1990] [Bertino et al., 1997], (c) y por último, mejora de la estrategia anterior, en vez de deshacer las actualizaciones que generan la inconsistencia, el sistema gestor reacciona de forma autónoma realizando un conjunto de acciones reparadoras capaces de eliminar la violación, llevando la BD a un estado consistente. Esta estrategia es muy costosa en términos de procesamiento computacional y complejidad, pues implica diseñar algoritmos complejos para detectar las inconsistencias y comparar la información [Ceri et al., 1994] [Gertz, 1994].

Por otra parte, para solucionar el problema de las restricciones de integridad recogiendo la mayor parte de la semántica del UD se han realizado distintas propuestas de investigación, como por ejemplo, la de [Lazarevic y Misic, 1991] que propone una extensión del modelo Entidad-Interrelación Extendido (EER) acompañado con restricciones de integridad referencial. En esta propuesta los tipos de entidades e interrelaciones se extienden utilizando acciones asociadas a las operaciones de actualización. Estas acciones especifican si una operación de actualización debe ser restringida, o debería invocar otras acciones para preservar las restricciones de integridad. La transformación del esquema EER al esquema relacional en esta propuesta se realiza a través de la implementación de procedimientos almacenados, en el caso de utilizar una BD relacional, o métodos en el caso de utilizar BDOO. Aunque los procedimientos almacenados se consideran una aproximación tradicional para conservar las restricciones de integridad estos mecanismos no se ejecutan de manera autónoma.

La propuesta de [Balaban y Shoval, 2002] realiza una extensión del modelo ER introduciendo métodos de estructura para asegurar la consistencia de las restricciones de integridad cuando se realizan operaciones de actualización. Por ejemplo, en esta propuesta cuando se inserta una ocurrencia en un tipo de entidad, se estudian todas las relaciones binarias asociadas con esta entidad para asegurar que las restricciones de cardinalidad se verifican. Así pues, se pide al usuario todas las interrelaciones asociadas con la ocurrencia insertada para cumplir la integridad. Para ello, cabe la posibilidad de que en algunos casos haya ocurrencias nulas insertadas en la BD. En

esta propuesta se ha tratado el problema de conservar las restricciones de cardinalidad generando automáticamente métodos estructurales sobre un esquema EER. Estos métodos estructurales tienen la responsabilidad de reforzar las restricciones. La propuesta depende de la interacción con los usuarios, es decir, preguntar a los usuarios para elegir el tipo de acción a realizar para reforzar una restricción.

En la propuesta de [Olivé, 2003] se ha concretado un método para definir las restricciones de integridad en el modelo conceptual orientado a objeto como operaciones especiales. La especificación formal de estas operaciones se deriva de la especificación de las restricciones de integridad correspondiente. El objetivo principal de estas operaciones es controlar restricciones de integridad estáticas y dinámicas. Estas restricciones se especifican en el cálculo relacional y se adaptan a diagrama de clases de UML. Esta propuesta se ha realizado en el modelo conceptual, sin definir las reglas para transformar estas operaciones a un modelo lógico. Tampoco se ha explicado qué mecanismo se va a utilizar para transformar estas operaciones a la hora de la implementación en una BD. Aunque sí que indica como trabajo futuro, la transformación de estos métodos a constructores apropiados en una plataforma independiente o específica incluyendo mecanismos de BD como checks, aserciones, disparadores, etc.

En la propuesta de [Pastor, 2004] se presenta un algoritmo que acepta un esquema EER y genera un conjunto de transacciones seguras asociadas al mismo. La propuesta incluye un análisis de las restricciones definidas en el esquema con respecto a cada operación posible, si se ve afectada o no la restricción por estas operaciones y, en el caso de que sí se vea afectada, se aplican criterios de mantenimiento de integridad incorporando si es posibles nuevas operaciones junto a la operación estudiada en una misma transacción, o sino, se puede restaurar el estado inicial de la BD. En esta propuesta se ha utilizado un lenguaje transaccional que permite especificar las transacciones que modelan la dinámica del sistema de información. El problema encontrado es que no ha sido implementada o probada en un SGBD, ni tampoco tiene un soporte en una herramienta CASE comercial para

aplicar el algoritmo de generación de estas transacciones seguras, por lo que resulta muy complicado su uso.

La propuesta de [Cuadra, 2003] tiene como aportación principal, dentro de la fase conceptual de la metodología de desarrollo de BD, la definición formal de las restricciones de cardinalidad de entidad y de interrelación de Merise [Tardieu et al., 1983] y de Chen [Chen, 1976] fijando, por tanto, de todas las aproximaciones estudiadas la más ventajosa para el modelado conceptual. También, se estudió la fase de modelado lógico, en la que el modelo por excelencia es el relacional, ya que la mayoría de las reglas de transformación para pasar de un esquema conceptual a un esquema relacional son básicas y no consideran todas las restricciones definidas en el modelo conceptual. Para controlar esta pérdida se propone trabajar más dentro de la fase de diseño relacional generando automáticamente mecanismos de control para la conservación de las restricciones de cardinalidad.

En resumen, podemos indicar, después del estudio realizado, que sería necesario realizar un análisis sistemático de las restricciones de integridad en las BDs para dar una solución completa que abarque todas las fases de la metodología de desarrollo de BD relacional. Puesto que las otras propuestas dan una solución de una parte o una fase del desarrollo sin estudiar los problemas asociados con la solución.

Tampoco existen aproximaciones que den un soporte automático para los mecanismos que proponen dentro una herramienta CASE. En el caso de encontrar alguno, estos estudios se realizan sobre prototipos de investigación. Ninguna de las propuestas incorporan los mecanismos de control de semántica a una herramienta CASE comercial. Las cuales carecen de reglas de transformación complejas por lo que se limitan a la transformación de elementos básicos.

Además, como hemos visto en este estudio no existen aproximaciones que implementen sus propuestas en un SGBD. La mayoría de las propuestas se realizan en la fase conceptual, y al final estas propuestas no se verifican ni se validan para comprobar sus comportamientos.

2.2 Conceptos Sobre Tecnología Activa Aplicada a BD

Este apartado pretende dar una visión sobre como se aplica la tecnología activa a la BD. Para ello el apartado se ha divido en: una introducción sobre los sistemas activos en BD para analizar las carencias y las ventajas tanto de los sistemas activos como pasivos. Posteriormente, se exponen los componentes básicos de los sistemas activos. La semántica de reglas activas incluyendo las características del modelo de conocimiento y del modelo de ejecución. También, se presenta en esta sección la arquitectura genérica del sistema activo explicando sus principales procesos. Por último, se describe el análisis de ejecución de las reglas activas explicando las distintas estrategias utilizadas para detectar y evitar los problemas de la ejecución. En la sección 2.2.4 se resumen algunos trabajos relacionados en el campo de los sistemas activos, como la metodología IDEA para el desarrollo de las BDs activas. En la misma sección se estudia el sistema activo del estándar SQL3 con profundidad y las características de algunos sistemas gestores comerciales que incluyen mecanismos activos o reglas activas considerando las diferencias y las comparaciones entre el estándar SQL3 y estos sistemas. Por último, en la sección 2.2.5 se discute algunos resultados obtenidos sobre lo expuesto en el estado de la cuestión, algunas comparativas de los distintos enfoques, y el análisis de los resultados según el estudio realizado.

2.2.1 Introducción

Un SGBD es activo si es capaz de detectar eventos y reaccionar ante los mismos. La capacidad reactiva se dispara y se gestiona de manera automática por parte del SGBD. Para que el SGBD pueda hacerse cargo de este comportamiento activo, es preciso que el desarrollador de la BD haya indicado las características del comportamiento activo. Es decir, qué situación debe ser vigilada, cuál es la reacción que debe tener lugar cuando se produzca esta situación y cómo debe tratarse el conjunto de reglas y su ejecución [Elizondo, 1998].

Muchos sistemas de BD soportan mecanismos activos. En los sistemas relacionales se llaman TRIGGERS (Disparadores). Sin embargo, desgraciadamente estos mecanismos se han desarrollado independientemente por los vendedores, y por ello no son reutilizables en distintos SGBDs, aunque estos sistemas utilicen el mismo modelo de datos [Cochrane et al., 1996] [Kulkarni, 1998]. En los últimos años, se ha dirigido un considerable esfuerzo a mejorar la comprensión de estos sistemas activos, se han definido diversas funcionalidades y se han sugerido muchos usos en torno a ellos. Estas funcionalidades son, por ejemplo, la verificación de las restricciones de integridad, la monitorización del proceso de introducir los datos, las auditorías de Las BDs, etc.

En general, un SGBD activo permite manejar el comportamiento reactivo de una manera centralizada mejor que distribuida, replicada, e incluida en los programas de la aplicación. Mediante los sistemas de BD activos se consigue un nuevo nivel de independencia de conocimiento [Paton, 1998] que tiene las ventajas siguientes: (1) Evolución sencilla: se puede actualizar la semántica activa cambiando las reglas en lugar de actualizar el conjunto de las aplicaciones que llevan el comportamiento reactivo; y (2) Garantizar una política de reforzamiento: cualquier aplicación que accede a la BD debe obedecer las restricciones descritas por las reglas responsables de estas políticas. Aunque los sistemas activos tienen muchas capacidades, el desarrollo de las aplicaciones de reglas activas es una tarea difícil debido al comportamiento impredecible de estas reglas. No es sencillo monitorizar la ejecución de las reglas y el desarrollador necesita más esfuerzo para llevar acabo la tarea de realizar un análisis de las reglas activas, para garantizar la ejecución y tener un estado final consistente en la BD.

Las aplicaciones tradicionales de las reglas activas forman parte de la BD, y son gestionadas por el SGBD de una forma transparente a los usuarios. Algunos ejemplos de estas aplicaciones son:

- Mantener las reglas de negocio que expresan las estrategias de una organización para llevar a cabo sus funciones primarias.

- Utilizar reglas activas para mantener datos derivados, como por ejemplo, el importe total de una factura o la nota media del expediente de un estudiante.
- Mantener la consistencia de tablas replicadas, especificando reglas que actualicen las réplicas cuando las tablas originales son actualizadas.
- Permitir la notificación de que está ocurriendo algún evento de interés. Por ejemplo, se puede utilizar un sistema de BD activo para monitorizar la temperatura de un horno industrial. La aplicación puede insertar periódicamente en la BD las lecturas de los sensores de la temperatura y se pueden crear reglas que se activen cuando se alcancen niveles peligrosos, disparando una alarma.
- Mantener la seguridad y realizar auditorias sobre el acceso a los datos.

2.2.2 BD Pasiva Versus BD Activa

En general, se pueden distinguir distintas arquitecturas de BD según donde se desarrolle el control de restricciones, cada una de ellas con sus ventajas y desventajas como se resume en la tabla 2.1. La arquitectura tradicional (Cliente-Servidor) garantiza la integridad de los datos a través de incorporar procedimientos de control dentro de las aplicaciones que acceden a la BD. En este caso, la semántica se repite en todas las aplicaciones que actualizan los datos y se produce un problema de descentralización semántica que lleva consigo redundancia, distribución y dificultad en el mantenimiento de la BD. Una aproximación para garantizar las restricciones en la arquitectura tradicional consiste en incorporar aplicaciones especiales que realicen un sondeo (*polling*) periódico para comprobar el estado de la BD. Los inconvenientes de esta solución son que el sondeo puede no realizarse en el momento adecuado y la realización del mismo afecta a la eficiencia de la BD, especialmente cuando se realizan muchas consultas sobre los datos.

La arquitectura a tres capas es una solución a la limitación presentada por la arquitectura tradicional, pero además de la complejidad asociada a esta

aproximación, las restricciones de integridad son gestionadas fuera de la BD, lo que provoca que sean más difíciles de entender y de implementar.

Por el contrario, la utilización de la tecnología activa ha sido considerada ampliamente en la literatura desde que esta tecnología ha sido introducida en Los SGBDs. En estos sistemas se puede definir y transformar una restricción de integridad a una regla activa [Widom y Finkelstein, 1990] [Gertz y Lipeck, 1993]. Esta tecnología permite a los desarrolladores ejecutar códigos de programas de una manera autónoma, para verificar las restricciones lo que ayuda a mejorar el funcionamiento de la BD, centralizando la semántica en la misma, además de facilitar el mantenimiento y la reutilización de las restricciones.

Arquitectura	Ventajas	Desventajas
En los clientes (BD pasiva)	Mejor rendimiento, tradicional y muy conocida, etc.	Redundancia, distribución, y difícil mantenimiento, etc.
En una capa intermedia (Tres capas)	El cliente se comunica con la capa media independientemente del SGBD utilizado. Mejora el rendimiento si existe gran cantidad de usuarios, etc.	Complejidad para desarrollar. Carencia en la portabilidad de aplicaciones, etc.
Dentro de la BD (BD activa)	Reutilización, Nivel de automatización, Semántica centralizada, Mantenimiento más sencillo, etc.	Hay que garantizar la terminación y la confluencia. Complejidad de implementación, etc.

Tabla 2.1. Comparación entre distintas arquitecturas de BD.

Los SGBDs soportan entre otros aspectos, concurrencia, control del acceso, recuperación de errores, y distribución de datos. Sin embargo, los sistemas de información necesitan otros componentes para (1) manejar la interfaz con los usuarios finales, (2) realizar operaciones complejas sobre los datos, y (3) controlar la ejecución de las operaciones [Berghe, 2000]. Por ello, normalmente, estas tareas se realizan a través de los programas de aplicaciones como reglas activas o disparadores

(*active rules, triggers*) que son los mecanismos fundamentales que nos proporciona esta tecnología para conservar la consistencia de la BD.

Así pues, una BD activa puede aceptar ciertas actualizaciones deseadas e impedir otras indeseadas que violen las restricciones de integridad de sus datos. Una vez que la BD activa detecta la aparición de un evento determinado, dispara autónomamente unas reglas preestablecidas que examinan primero el estado de la BD por unas condiciones predefinidas en estas reglas. Si estas condiciones son ciertas, las reglas realizan ciertas operaciones definidas por los desarrolladores para proteger la consistencia de los datos. A estas reglas se les denominan reglas ECA (Evento, Condición y Acción).

La fuerza del sistema activo es su capacidad de llevar una parte de las funcionalidades de las aplicaciones. En la figura 2.1 se muestra como una acción a_i debe realizarse dentro de varios programas en la BD pasiva, mientras la misma acción puede mantenerse en una sola regla activa compartida entre las aplicaciones de la BD activa. Los beneficios del sistema activo son los siguientes:

- Esta técnica simplifica la implementación de las aplicaciones y permite la reutilización de acciones implementadas en las reglas activas.
- Permite un nivel más alto de automatización para controlar dominios donde se pueden activar muchas operaciones sin las intervenciones directas del usuario.
- Proporciona una independencia que permite reducir el coste del desarrollo y el esfuerzo del mantenimiento, al actualizar la semántica cambiando unas reglas en lugar de actualizar el conjunto de aplicaciones.
- Garantiza una política de seguridad que no permite acceder a la BD sin obedecer las autorizaciones descritas por las reglas responsables de estas políticas.

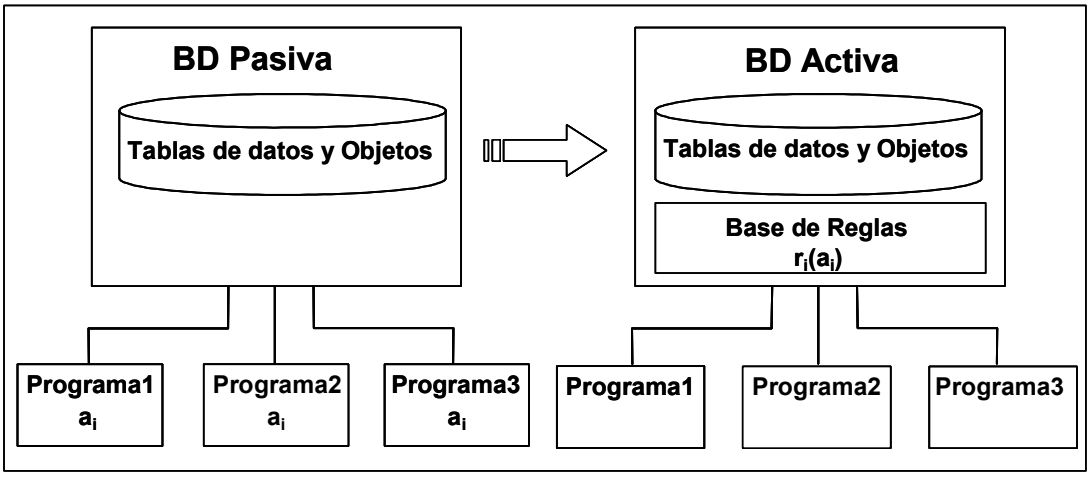


Figura 2.1. La diferencia entre una BD pasiva y una BD activa.

El uso del sistema activo puede mejorar el proceso de modular las aplicaciones ya que muchos aspectos del comportamiento de las aplicaciones pueden expresarse como políticas "causa y efecto". Después de definir estas políticas dentro de un UD (por ejemplo, una organización determinada), la transformación de estas políticas a reglas ECA será una tarea sencilla [Ceri et al., 2000]. De esta manera, el proceso de diseñar aplicaciones activas puede facilitarse hasta cierto punto. Aparte de esto, hay que tener en cuenta las consecuencias negativas que pueden aparecer por utilizar estos sistemas.

El primer problema en estos sistemas es cómo se puede garantizar que las aplicaciones activas correspondan completamente a las expectativas y necesidades del usuario. En las aplicaciones pasivas la responsabilidad de reforzar la integridad de los datos se realiza a través de ejecutar de manera planteada los procedimientos almacenados o por las interfaces de aplicación. Mientras, en las aplicaciones activas, esta responsabilidad se realiza de forma totalmente automática e independiente, a través de reglas que monitorizan el proceso de ejecutar acciones apropiadas. Desgraciadamente, estas reglas pueden actuar de una manera muy complicada y sus interacciones son a menudo escasamente visibles, sobre todo cuando el número de reglas especificadas es grande.

El segundo problema con el uso del sistema activo es el diseño de la aplicación, donde todavía existe poca experiencia con respecto a la representación de la semántica dentro de las reglas activas [Vaduva, 1998].

2.2.3 Componentes Básicos de un Sistema Activo

En esta sección se presentan los componentes básicos que comparten la mayoría de los sistemas activos de BD.

2.2.3.1 Semántica de las reglas activas

Un sistema activo es una extensión de un sistema tradicional de BD, el cual incluye un conjunto de reglas. Incorporando estas reglas se refuerza significativamente la funcionalidad de estos sistemas [Ceri y Widom, 1990] [Widom y Ceri, 1996] [Paton, 1998]. Estas reglas ofrecen alternativas flexibles para implementar, dentro de las BDs funcionalidades importantes, como por ejemplo, la verificación y la comprobación de las restricciones de integridad [Kim y Chakravarthy, 1995]. Independientemente del lenguaje utilizado para implementar los sistemas activos existen dos tipos de reglas activas, Evento-Acción (EA) o Evento-Condición-Acción (ECA), como se muestra en la figura 2.2.

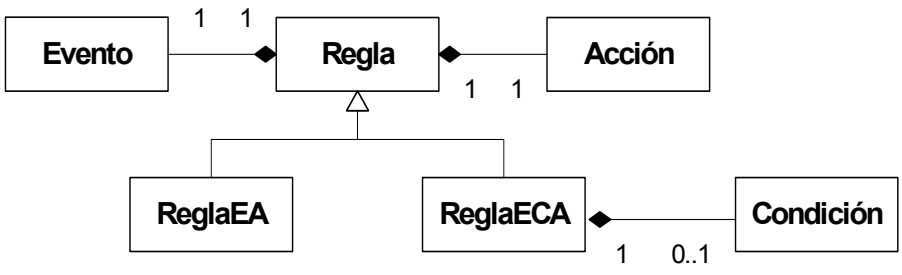


Figura 2.2. Las reglas activas EA y ECA

La ejecución de una regla activa se realiza a través de detectar el evento, evaluar la condición en el caso de ECA, y ejecutar la acción. Las fuentes de los eventos pueden

provenir de: una sentencia de manipulación de datos DML (*Data Manipulation Language*) o una operación de recuperación de datos, una invocación de un método en las BDs orientados a objetos, un signo del reloj del sistema, o una combinación de éstos. La actuación de un sistema activo se realiza siempre y cuando existan reglas activas predefinidas para controlar las actualizaciones sobre los datos. Una vez haya ocurrido un evento, la condición de la regla se evalúa. Cuando se evalúa la condición a cierto, la acción de la regla se ejecuta.

El presente apartado pretende dar una visión actual de los sistemas activos, y de las distintas funcionalidades aportadas por cada uno de ellos. Para ello se presentarán primero, las características estructurales de las reglas activas y, segundo, las diversas estrategias de ejecución que pueden adoptar.

1. Modelo de conocimiento

El modelo de conocimiento de un sistema activo de BD indica los componentes que definen las reglas activas en ese sistema.

Todos los sistemas activos comparten el mismo modelo de conocimiento, que consiste en tres componentes principales [Paton y Díaz, 1999]: un evento, una condición, y una acción (ECA). La definición y las características de cada uno de estas componentes se muestran a continuación.

A. Evento

Un evento es una operación que cuando ocurre, el sistema activa una o varias reglas en un tiempo determinado. Las características asociadas a un evento se muestran a continuación:

- **Fuentes de las que puede provenir un evento;**
    - Sentencias de actualizaciones de datos (INSERT, DELETE, UPDATE), o sentencias de recuperación de datos (SELECT). En la figura 2.3, la regla $R_1(E_1, C_1, A_1)$ controla la inserción de nuevas tuplas en la tabla T. Esta regla se activa siempre y cuando el evento E_1 se genera. Si la condición C_1 es cierta, la acción A_1 se ejecutará.

```
DEFINE RULE: < R1 >
ON           < E1 ON T >
IF           < C1 > THEN DO < A1 >
```

Figura 2.3. La estructura general de una regla activa

- Sentencias que controlan el proceso de transacción en Las BDs, como BEGIN, ROLLBACK, COMMIT.
- Eventos abstractos o definidos por los usuarios (*abstract o user-defined*). Estos tipos de evento se generan explícitamente dentro de un programa de aplicación o acciones de reglas.
- Los eventos del reloj del sistema que se generan en un momento determinado. Este evento puede ser absoluto o periódico (*absolute o periodic*). Absoluto, como "1 de Enero de 2000 a las 12:00" y periódico, como "10 min del tiempo de cerrar la bolsa de valores".

- **Tipos de eventos:**
  - Eventos primitivos: un evento es primitivo cuando consta solo de una ocurrencia de un evento de los tipos anteriores. Todos los SGBDs soportan reglas que se activan implícitamente o explícitamente por los eventos primitivos. Los eventos que activan reglas de manera implícita, más comunes en las BDs son, por ejemplo, las sentencias LMD y de gestión de transacciones. Los eventos que activan reglas de manera explícita que se proponen en algunos sistemas son, por ejemplo, operaciones del reloj del sistema, y la invocación de una función (desde un programa o directamente por un usuario).
  - Eventos compuestos: un evento compuesto es una combinación de varios eventos primitivos y/o compuestos. No todos los sistemas de BD activos soportan eventos compuestos. Los mecanismos activos que se proponen para las BDs relacionales, como por ejemplo SQL3 [Kulkarni, 1998], ARIEL

[Hanson, 1996], STARBURST [Widom, et al., 1991], POSTGRES [Stonebraker y Kemnitz, 1991] soportan una fuente limitada de eventos y también la detección de eventos compuestos es limitada [Philippe et al., 1999]. Además, existen otros sistemas con una declaración clara de un conjunto de operadores para expresar los eventos compuestos, como por ejemplo, CHIMERA [Guerrini y Montrsi, 1997] y SAMOS [Gatziu y Dittrich, 1998]. Los operadores más comunes para construir evento compuestos según [Bailey y Ramamohanarao, 1995] son:

- CONJUNCTION: (E1, E2), ocurre cuando se presentan los eventos E1 y E2, en cualquier orden.
- DISJUNCTION: (E1 | E2), ocurre cuando toma lugar cualquiera de los eventos E1 ó E2.
- SEQUENCE: SEQ (E1, E2), ocurre cuando sucede primero E1 y posteriormente E2.
- NEGATION: NOT (E) en INT, ocurre cuando el evento E no sucedió durante el intervalo INT. En el ejemplo de la figura 2.4, se muestra un evento compuesto que usa el operador (OR) para señalar si uno o ambos eventos del operador tienen lugar para activar la regla.

```
DEFINE RULE: <nombre_regla>
ON <ON INSERT INTO T1 OR DELETE ON T2>
IF <Condición> THEN DO <Acción>;
```

Figura 2.4. Evento Compuesto

- **El papel de un evento (*event role*):**

Otra característica asociada a un evento es el papel que juega un evento para disparar una regla. Si el papel es obligatorio, entonces cada vez que se genera un evento se dispara una regla. Las reglas activas en este caso son de tipo ECA. Sino existe el

papel el disparo de las reglas no está asociado con una ocurrencia de un evento. Una regla activa en este caso es de tipo CA que tiene funcionalidad significativamente diferente de las reglas de ECA. Si el papel es opcional, entonces se puede encontrar con ambos tipos de reglas.

B. Condición

La condición es una expresión lógica descrita en SQL o un procedimiento de tipo lógico escrito en un lenguaje de programación. La condición especifica lo que tiene que ser verificado una vez activada la regla. Las características asociadas con la estructura de una condición son las siguientes:

- El Papel de una condición (*condition role*): define si se debe incluir o no una condición en la regla. En las reglas ECA, la condición es, generalmente opcional. Sino se especifica ninguna condición, la regla es de tipo EA.

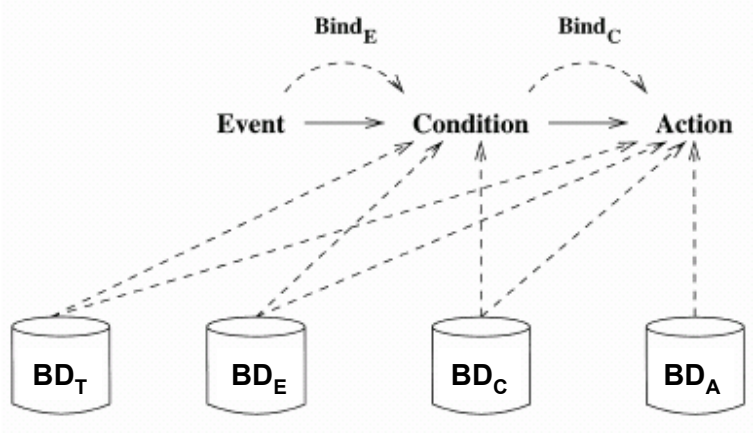


Figura 2.5. El contexto durante el cual se procesa una regla.

- El Contexto de la condición es la información que está disponible para evaluar la condición. Los distintos componentes de la regla no son evaluados y ejecutados de forma aislada en la BD. Una condición puede ver y comprobar por lo menos cuatro estados de la BD diferentes: el estado de la BD al comienzo de la transacción actual (BD_T), cuando tiene lugar la ocurrencia del evento (BD_E), cuando se evalúa la condición (BD_C), y cuando se ejecuta la acción (BD_A). Los

sistemas activos deben facilitar a las condiciones de las reglas, el acceso a acceder a cero o más estados BD_T, BD_E, y BD_C, y debe proporcionar además, acceso para unir los eventos con la condición ($Bind_E$). La disponibilidad de información de los diferentes componentes de una regla se muestran en la figura 2.5. La regla que se muestra en la figura 2.6 es un ejemplo de la utilidad del contexto de la condición. Esta regla utiliza la información del evento BD_E para responder con una acción apropiada a cualquier modificación del precio de venta de una acción. En general, las condiciones y acciones acceden a parámetros del evento, usando para referirse al antiguo valor antes de la modificación y al nuevo valor después de la modificación.

```
ON UPDATE TO PRECIO OF ACCION
IF NEW.PRECIO < > OLD.PRECIO
DO UPDATE ACCIONISTA
SET VALOR = VALOR * (NEW.PRECIO/OLD.PRECIO)
WHERE ID_ACCIONISTA IN
(SELECT ID_ACCIONISTA FROM POSEE
     WHERE ID_ACCION = UPDATE.ID_ACCION)
```

Figura 2.6. Un ejemplo de la utilidad del contexto de la condición y la acción.

C. Acción

La acción es la parte donde se implementa la reacción requerida ante un evento, la acción se ejecuta cuando la regla se activa y su condición es verdadera. Las posibilidades de implementar acciones con muchas capacidades dependen del lenguaje de programación utilizado. Las características asociadas con una acción son:

- Las acciones pueden contener sentencias DML, como INSERT, DELETE, UPDATE, operación de recuperación de datos como SELECT, operaciones de control de transacción, COMMIT O ROLLBACK, invocación de funciones de fuera de la BD, o realizar acciones alternativas utilizando DO-INSTEAD.

- El Contexto de la acción es similar al de la condición, e indica la información que está disponible para la acción, como se muestra en la figura 2.5. A veces es posible que la información pase de la condición de una regla a su acción como BD_E o $Bind_C$. La regla de la figura 2.6 es un ejemplo de la utilidad del contexto de la acción. Esta regla se usa para revisar los datos almacenados en los valores de todos los accionistas que son afectadas por el cambio de las acciones que poseen.

2. Modelo de ejecución

En general, el modelo de ejecución de los sistemas activos es bastante complicado porque su actividad depende del comportamiento activo de sus reglas. El modelo de ejecución de las reglas especifica cómo se trata un conjunto de reglas y cómo se activa en el momento de la ejecución. El modelo de ejecución de un SGBD determinado se relaciona estrechamente con las características específicas de este sistema. Es decir, el modelo de ejecución de las reglas activas está fuertemente relacionado con el modelo de datos y de las transacciones del sistema gestor. Por eso, los modelos de ejecución de los SGBDs que existen en el mercado varían según la naturaleza de cada sistema.

Se puede decir, en general, que los modelos de ejecución comparten un algoritmo genérico [Widom y Finkelstein, 1990] [Baralis et al., 1998] que describe las fases de ejecución de las reglas activas, como se muestra en la figura 2.7.

A continuación, pasamos a describirlo:

- Señalar el evento (*signalling*): Significa la aparición de una ocurrencia de evento causada por una fuente de evento. Todas estas ocurrencias se reúnen en el historial de eventos, que es una lista de ocurrencias ordenadas según el tiempo de la ocurrencia (*timestamp*). La historia de los eventos contiene el primer evento que se produce al principio de la transacción y todos los eventos señalados como consecuencia de la señalización del primero.
- Desencadenamiento (*triggering*): En esta fase se cogen los eventos generados hasta el momento, y se disparan las reglas correspondientes. La asociación de una

regla con su ocurrencia del evento forma una instanciación de la regla.

- Evaluación (*evaluation*): En esta fase se evalúan las condiciones de las reglas activadas. Cuando la condición se evalúa a verdadero la regla correspondiente se añade al conjunto de reglas del conflicto. El conjunto del conflicto está formado por todas las instancias de las reglas activas en las que sus condiciones son verdaderas [Paton y Díaz, 1999].
- Planificación (*scheduling*): En esta fase se indica cómo se procesa el conjunto de conflicto. Durante la planificación, se seleccionan las reglas activadas en las mismas secuencias en las que han sido insertado en el conjunto de conflicto. Si hay más de una regla definida para un evento, se activan estas reglas al mismo tiempo.
- Ejecución (*execution*): en esta fase se ejecutan las acciones de las reglas seleccionadas. Durante la ejecución de las acciones pueden señalarse otros eventos, los cuales pueden causar el disparo en cascada de otras reglas.

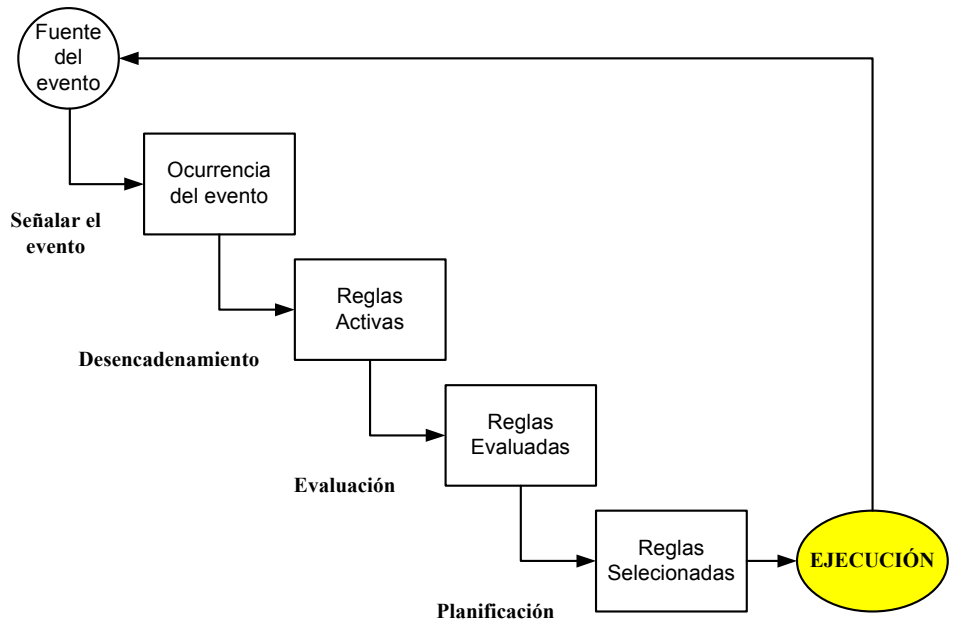


Figura 2.7. Las fases del algoritmo genérico de ejecución de reglas.

En general, existen varios componentes genéricos que son válidos para describir el modelo de la ejecución de las reglas [Paton et al., 1994] [Widom y Ceri, 1996] [Paton, 1998] [Paton y Díaz, 1999]. A continuación se describen los distintos componentes del modelo de ejecución:

A. Modos de acoplamiento

En general, los modelos de ejecución contienen dos modos de acoplamiento (*coupling modes*); evento-condición y condición-acción. En el primer modo se especifica la relación entre el evento correspondiente a una regla y la evaluación de la condición de la misma, mientras en el segundo se especifica la relación entre la evaluación de la condición de una regla y la ejecución de su acción. Los posibles modos de acoplamiento son:

- Modo inmediato (*immediate*): El modo inmediato de acoplamiento evento-condición significa que la condición se evalúa inmediatamente después de la ocurrencia del evento y durante la misma transacción. El modo inmediato de acoplamiento condición-acción significa que la acción se ejecuta inmediatamente después de la evaluación de la condición y también durante la misma transacción.
- Modo diferido (*deferred*): El modo diferido de acoplamiento evento-condición significa que la condición se evalúa al final de la transacción actual. El modo diferido de acoplamiento condición-acción, significa que la acción se ejecuta al final de la transacción en la que se ha evaluado la condición siempre cuando la condición se cumpla.
- Modo separado (*detached*): El modo separado del acoplamiento evento-condición significa que la condición se evalúa en una transacción diferente de aquella en la que ha ocurrido el evento. El modo separado del acoplamiento condición-acción significa que la acción se ejecuta en una transacción diferente de aquella en la que se ha evaluado la condición. Este modo puede ser útil para descomponer una transacción grande producida por la activación de un conjunto grande de reglas a unas transacciones más pequeñas.

B. Proceso en cascada

Otro componente del modelo de ejecución es el proceso en cascada. Se aplica con mucha frecuencia cuando la ejecución de la acción de una regla causa la activación y ejecución de otras reglas. Existen dos procesos para tratar este tema, son:

- Proceso Recursivo: Significa que la ejecución de una regla activa R se suspende para procesar las reglas que han sido activadas por los eventos producidos por la acción de la regla R. El algoritmo utilizado para representar este proceso se muestra en la figura 2.9. Si el evento e_1 activa las reglas R_1 y R_2, estas dos reglas se insertan en el conjunto del conflicto S. Si R_1 tiene prioridad más alta que R_2 entonces R_1 será seleccionada primero para la ejecución. Cuando se activa R_1 su acción genera el evento e_2 que activa R_3 y R_4, estas últimas dos reglas se insertan también en el conjunto de conflicto S en las que las condiciones y las acciones de R_3 y R_4 se ejecutan antes que R_2. En particular, se aplica este proceso en los sistemas gestores que soportan un modo inmediato de acoplamiento como son los SGBDs relacionales.

```
S={R1, R2}
DO WHILE Ri ∈ S
CHOOSE Ri TO EXECUTE;
IF Ri(Condition)=TURE
     THEN DO Ri(Action)
          IF EXISTS S'={R3, R4} THEN
          DO WHILE Rj ∈ S'
          CHOOSE Rj TO EXECUTE;
          IF Rj(Condition)=TURE
          THEN DO Rj(Action)
          ……
          END IF;
          END DO;
END IF
END DO;
```

Figura 2.8. El algoritmo recursivo de las reglas.

- Proceso Iterativo: Significa que las reglas activadas durante la ejecución de una regla se combinan con aquellas reglas activadas por la fuente del evento original. Esto significa que la condición o la ejecución de la acción nunca se suspenden. El algoritmo utilizado para representar este proceso se muestra en la figura 2.9. Si el evento e_1 activa las reglas R_1 y R_2, estas dos reglas se insertan en el conjunto del conflicto S. Como R_1 tiene la prioridad más alta en S, se selecciona primero para la ejecución. Ahora si R_1 genera el evento e_2 que activa R_3 y R_4, estas dos reglas se insertan también en el conjunto S. Pero en este caso, se selecciona R_3 primero para la ejecución porque tiene prioridad más alta que cualquier regla en el conjunto S.

```
S={R1, R2}
DO WHILE Ri ∈ S
CHOOSE Ri TO EXECUTE;
IF Ri(Condition)=TURE
     THEN DO Ri(Action)
          IF EXISTS S'={R3, R4} THEN
          S={R3, R2, R4}
          END IF;
END IF
END DO;
```

Figura 2.9. El algoritmo iterativo de las reglas.

C. Granularidad de la transición

Una transición es un cambio del estado de la BD por la ejecución de una secuencia de operaciones de manipulación de datos DML [Ceri y Widom, 1990]. La granularidad de la transición indica la relación entre la ocurrencia de un evento y la instanciación de la regla que ha sido disparada. Existen dos tipos de granularidad:

- Orientada a tupla o registro: en este caso, si el evento afecta a una o varias tuplas, la instancia de la regla asociada a cada tupla se procesa de forma independiente.
- Orientada a conjunto: en este caso, aunque el evento afecte a varias tuplas, la regla que dispara debe procesarse sólo una vez para todas ellas.

D. Resolución de conflictos

Debido a que el orden de la ejecución de las reglas tiene una gran influencia en el estado final de BD, se utilizan las prioridades para tratar de establecer un orden secuencial para ejecutar las reglas en un conjunto de conflicto. Cuando se activan múltiples reglas al mismo tiempo los sistemas activos utilizan una política general en la que se selecciona la regla con la prioridad más alta para la ejecución [Kim y Chakravarthy, 1995]. La resolución de conflictos es una política que determina qué sucede cuando hay varias reglas disparadas. Posibles políticas para resolver este problema son [Paton y Díaz, 1999]:

- Política Absoluta: En esta política se define un orden global para la ejecución usando un valor absoluto para todas las reglas.
- Política Relativa: En esta política se asegura el orden solo entre un par de reglas, en las cuales una de las dos debe dispararse antes que la otra.
- Política Aleatoria: En esta política el orden de la ejecución de la regla se escoge arbitrariamente.

E. Efecto neto

El efecto de los eventos se puede considerar como efecto aisladamente o un efecto global dentro de la transacción; en este último caso se dice que el efecto es neto. La base de esta estrategia previene que varias actualizaciones en el mismo dato pueden ser consideradas como una sola actualización. Como por ejemplo, si una tupla se ha modificado y luego se borra, el efecto neto es borrar la tupla original, o si una tupla se ha insertado y luego se ha modificado, el efecto neto es la inserción de la tupla modificada, etc. [Ceri y Widom, 1990] [Hanson, 1996].

F. Recuperación de errores

Una política de recuperación de errores en el sistema activo trata de llevar el sistema a un estado correcto, desde el que pueda seguir funcionando. La mayoría de los sistemas activos de BD cuando se produce un error en la ejecución de las reglas adoptan deshacer la transacción (ROLLBACK), y recuperar el estado inicial de la

BD. Sin embargo, existen otras alternativas como (IGNORE), en la que se ignora la regla que ha causado el error y continuar el proceso de otras reglas, o abortar la transacción (ABORT).

2.2.3.2 Arquitectura del sistema activo

Para cubrir las características del sistema activo descritas anteriormente es necesario describir los componentes de la arquitectura del mismo, como se muestra en la figura 2.10. Los principales componentes son:

- Detector de Eventos (*event detector*): el cual comprueba los eventos de interés que han ocurrido en el sistema. Los eventos simples se notifican desde la BD o desde fuentes externas. Los eventos compuestos están construidos con eventos simples más información procedente de eventos pasados que pueden obtenerse del Histórico.
- Monitor de Condiciones (*condition monitor*): comprueba las condiciones de las reglas asociadas a los eventos detectados por el Detector de Eventos.
- Planificación (*scheduler*): compara las reglas que han sido lanzadas recientemente con aquellas que han sido lanzadas previamente, se actualiza el conjunto conflicto y dispara alguna regla que esté lista para ejecutarse inmediatamente.
- Evaluador de Consultas (*query evaluator*): ejecuta las acciones y consultas en la BD. Se accede al estado actual de la BD como a los estados anteriores, para monitorizar o comprobar el desarrollo de una transacción.

La funcionalidad de cada uno de los componentes depende mucho del modelo de conocimiento y el modelo de ejecución del sistema activo. Estructuralmente, las BDs activas se pueden clasificar en dos categorías; (1) Por Capas (*layered architecture*): los mecanismos activos se desarrollan como una capa de software por encima del sistema pasivo, con la ventaja de que se pueden convertir los sistemas pasivos existentes a sistemas activos, sin actualizar el núcleo de la BD. Sin embargo, el hecho de no actualizar el núcleo podría resultar impactante para la ejecución de la

BD activa, y limitar la funcionalidad de detección de los eventos. Un ejemplo de este tipo de arquitectura es el sistema activo SAMOS, que está implementado sobre el sistema gestor comercial ObjectStore [Gatziu y Dittrich, 1998]. (2) Integrado (*integrated architecture*): los mecanismos activos de la BD se desarrollan directamente integrados en el SGBD, cambiando la implementación del sistema pasivo. La detección de eventos, la monitorización de las condiciones y la ejecución de las acciones se pueden realizar de una manera más eficiente. Esta propuesta es mejor porque libera al diseñador de las limitaciones de la propuesta anterior y es preferida para desarrollar sistemas industriales de grandes capacidades. Un ejemplo de este tipo de arquitectura es el sistema activo NAOS, extensión del sistema orientado a objeto DBMSO2 [Coupaye et al., 1994].

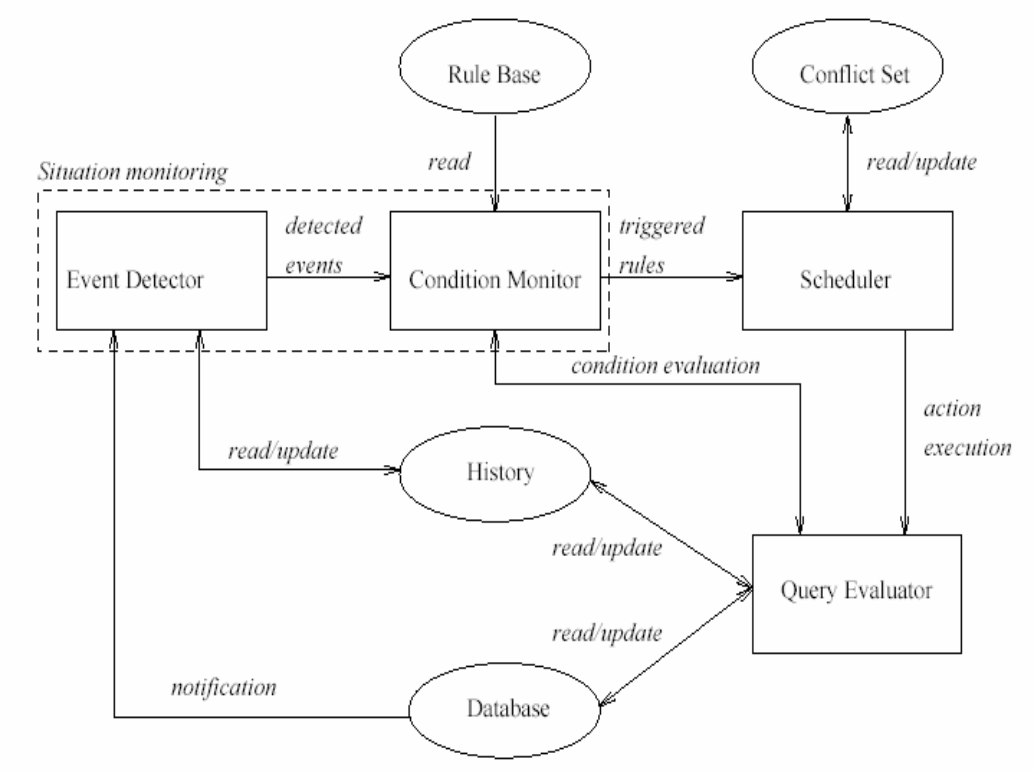


Figura 2.10. Arquitectura genérica de un sistema activo [Paton y Díaz, 1999].

A continuación pasamos a describir de manera más detallada la funcionalidad de cada una de los componentes de la arquitectura activa [Paton y Díaz, 1999]:

1. Detección de eventos

En el proceso *Detector de Eventos* se monitorizan los eventos de interés que han ocurrido en el sistema. Existen dos aspectos principales en la implementación del detector de eventos, la monitorización de los eventos primitivos y la acumulación de información relevante para los eventos compuestos.

La detección de eventos primitivos implica normalmente algunas comprobaciones en el núcleo del sistema de BD, tal que no es posible implementar este proceso como una capa por encima del sistema activo sino que la implementación forma parte del núcleo del sistema activo. Por ejemplo, para detectar una operación de actualización que ha sido realizada, es necesario que la BD sea capaz de identificar el evento primitivo asociado a dicha operación.

La detección de eventos en sistemas activos orientados a objeto es distinta, ya que los eventos son generados como respuesta a la invocación de un método definido por el usuario. En este caso, el detector de eventos debe ser notificado por parte del método invocado sobre el evento del mensaje enviado. En este contexto, existen varias técnicas basadas en el envío de mensajes para detectar eventos primitivos y que a continuación se presentan:

Basado en el remitente (*dispatcher-based*): Esta técnica no conduce a un sistema activo porque la detección de los eventos se realiza en las aplicaciones. La desventaja principal de esta aproximación es que la detección de eventos está replicada, distribuida, y empotrada en programas de las aplicaciones y con ello se compromete la encapsulación y el mantenimiento.

Basado en el receptor (*receiver-based*): En esta técnica está basada principalmente en las capas (*wrappers*). El mensaje enviado activa una operación que envía el código original del usuario y la señal del evento, al detector de eventos. Este tipo de mecanismo de detectar los eventos primitivos, hace que los cambios en el núcleo de la BD no sean necesarios.

2. Monitor de condiciones

Existe una relación estrecha entre el detector de eventos y el monitor de condiciones. El detector de eventos proporciona la información sobre los eventos producidos e inicia el proceso del monitor de condiciones. Lo más complejo de este proceso consiste en seleccionar qué información pasa del detector de eventos al monitor de condiciones. Por ejemplo, en el caso de un evento primitivo como insertar un producto, la información necesaria consistirá únicamente en los valores del producto insertado. Mientras que para los eventos compuestos la información relevante de los eventos es más compleja. Por ejemplo, en una conjunción de eventos debe estar disponible la información de todos los eventos que forman parte de la conjunción cuando se compruebe la condición.

Una vez recuperadas las reglas asociadas a los eventos detectados, estas reglas deben ser pasadas al evaluador de consultas para determinar cual de ellas cumple su condición. El evaluador de consultas es una extensión similar a la que se usa en una BD pasiva, pues las condiciones de las reglas se parametrizan por los parámetros de los eventos, y puede también tener acceso a la BD así como a los estados pasados de la misma (consultando el histórico) si fuera necesario. El contenido histórico de un sistema específico de reglas depende del lenguaje de especificación de eventos, así como del tipo de acceso a través de las condiciones y acciones de las reglas a los estados pasados de la BD.

El diseño y la implementación de un sistema de monitorización de condiciones eficaz en los sistemas activos son importantes: se debe encontrar un equilibrio entre la expresividad del lenguaje usado y la eficiencia del proceso. Cuanto más expresivo sea el lenguaje usado, mejor será la descripción de la situación que se monitoriza y, de esta manera, menor será la frecuencia de invocación de las condiciones de las reglas.

3. Ejecución de las acciones

El planificador es el responsable de ejecutar aquellas reglas cuyas condiciones son satisfactorias así como del mantenimiento del conjunto conflicto de las reglas que

han sido activadas y esperan para ser ejecutadas. La complejidad del planificador varía de un sistema a otro, por ejemplo en POSTGRES [Stonebraker y Kemnitz, 1991] no existe ninguna prioridad que establezca el orden en el que deben ser activadas las reglas. Las reglas son ejecutadas en el orden en el que hayan sido activadas. En STARBURST [Widom, et al., 1991] el planificador es más complejo. Hay prioridades de unas reglas respecto de otras y son ejecutadas siguiendo un orden determinado. En este caso es posible que alguna regla llegue a ser borrada del conjunto conflicto sin haber sido ejecutada debido a que el procesamiento de las reglas se basa en el efecto neto de los cambios en la BD.

Cuando una regla es planificada para su ejecución, se envía su acción al evaluador de consultas, que es el responsable de actualizar la BD y el contenido histórico. La acción necesitará tener acceso a los datos del evento que ha causado la ejecución de la regla así como los datos que han sido calculados para la condición.

2.2.4 Análisis de Reglas

Durante el proceso de la ejecución, las reglas pueden actuar de manera complicada e inesperada [Montesi y Torlone, 2002]. El proceso desencadena las reglas activas como consecuencia de las actualizaciones en la BD, y la ejecución de una regla puede activar otras reglas incluyendo ella misma. El estado final de la BD depende del orden en que se activan y se ejecutan estas reglas. Por lo tanto, desarrollar aplicaciones de reglas activas puede resultar una tarea difícil y complicada. Además, es necesario más métodos y herramientas para ayudar a los diseñadores a realizar esta tarea [Dittrich et al., 1996] [Segev y Zhao, 1994]. La complejidad de este problema aumenta cuando un diseñador añade nuevas reglas a una aplicación existente, tal que las nuevas reglas pueden actuar recíprocamente con las reglas antiguas produciendo un estado final de la BD inesperado. Para evitar este problema se aplica el análisis de reglas que se usa para definir técnicas que permiten al diseñador predecir previamente los aspectos relacionados con el comportamiento de estas. En este contexto, los métodos de análisis representan la base de la

comprobación de la ejecución de las reglas. Estos métodos se clasifican en dos tipos: los métodos estáticos y los métodos dinámicos.

Los métodos estáticos analizan la definición de la regla en tiempo de compilación (*compile-time*). Este tipo de análisis investiga la definición de las reglas en modo estático para determinar, si existe la posibilidad de que una regla pueda activar directamente o indirectamente otras reglas incluyendo ella misma. El análisis estático se realiza a través de un análisis sintáctico y semántico de la definición de la regla simulando el proceso de las reglas con conjunto reducido de datos. Mientras, los métodos dinámicos examinan el comportamiento de la regla en tiempo de ejecución (*run-time*). Estos métodos mejoran el análisis de las reglas activas por considerar la influencia de las condiciones en sus comportamientos. El problema de este tipo de análisis es que muy costoso y necesita muchos recursos para realizarlo. Por eso, cuando no se puede garantizar la ejecución de las reglas en el análisis estático, es crucial y potencialmente muy costoso garantizarlo en el análisis dinámico [Baralis et al., 1998]. Además de esto, aunque el análisis de las reglas puede detectar los problemas en la ejecución, no se puede garantizar totalmente el proceso de las reglas en todas las circunstancias debido a la complejidad de estos sistemas [Berghe, 2000].

En este apartado nos centraremos en el análisis estático de las reglas. Esta técnica también ha sido utilizada en muchos de los trabajos relevantes en este área como [Kim y Chakravarthy, 1995] [Baralis et al., 1995] [Aiken et al., 1995] [Widom y Ceri, 1996] [Paton, 1998] [Lee y Ling, 1999] [Amghar et al., 2000] [Baralis y Widom, 2000] [Couchot, 2001], etc. El principal objetivo del análisis estático es verificar que el conjunto de conflicto de las reglas se ejecuta y termina según se espera. En este contexto, resulta interesante estudiar y garantizar las siguientes propiedades en la ejecución de las reglas:

- Terminación: Indica que la ejecución de cualquier conjunto de reglas debe terminar correctamente. Para garantizar este final debe evitar que las reglas se activen en una forma cíclica.
- Confluencia: Un conjunto de reglas es confluente si la ejecución termina produciendo un estado final único que no depende del orden de ejecución de las

reglas.

- Comportamiento observable: Se garantiza un comportamiento observable idéntico de un conjunto de reglas, cuando la ejecución de estas es concluyente y todas las acciones visibles llevadas 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 garantizar que la ejecución de las reglas termina. Por otra parte, la confluencia y el idéntico comportamiento observable no son esenciales. Con respecto al idéntico comportamiento observable, esta propiedad no juega un papel muy importante en el comportamiento de las reglas y es menos significativo que otras propiedades. ¿Por qué?, porque el idéntico comportamiento observable es una cuestión de cómo el desarrollador percibe el comportamiento de las reglas. Por ejemplo, las dos reglas R_1 (E_1, C_1, A_1:<*mandar un mensaje al usuario*>) y R_2 (E_1, C_1, A_2:<*abortar*>), estas reglas son confluentes pero el comportamiento observable de ellas no es idéntico. Tal que, si R_1 se ejecuta antes que R_2, el usuario primero recibirá un mensaje y luego la transacción se abortará. Mientras, si R_2 se ejecuta después de R_1, la transacción se aborta y no se enviará ningún mensaje al usuario. Para garantizar el idéntico comportamiento observable de las reglas es prescindible garantizar la confluencia, mientras la confluencia requiere garantizar la terminación [Vaduva, 1998].

1. Análisis de terminación

La terminación de un conjunto de conflicto de reglas se garantiza, si el proceso de estas siempre alcanza un estado de ejecución finita. El proceso no termina si las reglas se ejecutan infinitamente. Existen dos maneras para que una regla R_i pueda activar otra regla R_j. La primera es que la acción de R_i genera uno o más eventos que activan R_j, y la segunda cuando la acción de R_i causa que la condición de R_j pueda resultar verdadera. Por ello, en la figura 2.11 se presenta un ejemplo en el que muestra el problema de la no-terminación. El ejemplo consiste en dos tablas Empleados [Emp_Id, Categ, Sueldo] y Primas [Emp_Id, Cantidad]. Cada tupla en

Empleados almacena la categoría y el sueldo del empleado, y cada tupla en Primas almacena la cantidad de la prima otorgada al empleado. La regla ActCateg aumenta la categoría del empleado y la regla ActPrima aumenta la cantidad de la prima. Cada vez que la categoría del empleado se aumenta en 1, la prima se aumenta un 10%. En este ejemplo, se ve claramente la ejecución infinita (ciclo) de estas dos reglas, tal que cuando se modifica la cantidad de una prima activa ActCateg modificando la categoría del empleado, y cuando se modifica la categoría se activa ActPrima modificando la prima.

En general, las grandes aplicaciones de reglas activas suelen ser cíclicas. Sin embargo, sólo unos pocos ciclos son los que provocan situaciones críticas. De hecho, el que una ejecución sea cíclica es condición necesaria pero no suficiente para la no terminación. Por eso, y debido a la complejidad asociada al análisis de terminación, se adoptaron distintas políticas para afrontar la misma [Widom y Ceri, 1996].

En los SGBDs se utilizan las siguientes políticas para controlar la terminación de las reglas. La primera consiste en que el diseñador de las reglas debe asegurar la terminación. La segunda política se limita el número máximo de las reglas activadas en un proceso. Es decir, que el SGBD soporta un límite superior que determina cuántos ciclos permitidos durante el proceso de las mismas. Si se alcanza este límite durante la ejecución de la regla, dicha ejecución se deshace (ROLLBACK). Por ejemplo, en ORACLE el límite máximo por defecto es 32 reglas/proceso, se puede modificar este número a través de la iniciación del parámetro OPEN_CURSORS. Cuando la ejecución de la regla alcanza la terminación sin la intervención externa se llama terminación normal y cuando la ejecución termina explícitamente por el SGBD se llama terminación anormal. La terminación anormal es sólo una solución de emergencia que no garantiza un modelo correcto del comportamiento activo.

```
CREATE RULE ACTCATEG ON PRIMAS
ON UPDATE (CANTIDAD)
DO UPDATE EMPLEADOS SET CATEG = CATEG + 1
WHERE EMP_ID=UPDATED.EMP_ID;
CREATE RULE ACTPRIMA ON EMPLEADOS
ON UPDATE (CATEG)
DO UPDATE PRIMAS SET CANTIDAD = CANTIDAD * 1.1
WHERE EMP_ID=UPDATED.EMP_ID
```

Figura 2.11. Dos reglas en ejecución infinita (cíclico).

Otra política para afrontar el problema de no terminación es restringir el lenguaje de definición de las reglas. Es decir, que los lenguajes de definición de mecanismos activos en los productos comerciales contemplen algunas limitaciones de sintaxis que ayuden a restringir el proceso de las reglas. Al mismo tiempo, estas restricciones y limitaciones reducen la eficacia y la capacidad de los mecanismos activos de los productos comerciales. Por ejemplo, en IBM DB2 se usa la sentencia NO CASCADE BEFORE que restringe el proceso en cascada de los disparadores, tal que no se permite que la acción del disparador de tipo BEFORE dispare otros disparadores. En MS SERVER 2005 no existe la definición de disparadores de tipo BEFORE, se utiliza solo disparadores de tipo AFTER. En ORACLE, los valores NEW/OLD de referencia a tabla no se soportan. También, se levanta un error cuando un disparador examina una tabla que está siendo modificada, denominado, error de tabla mutante.

Por todo esto, se puede decir que el análisis de las reglas es una responsabilidad del diseñador, que debe asegurar siempre la terminación de ejecución utilizando algunas técnicas para el análisis, como por ejemplo, el gráfico de desencadenamiento (*triggering graph*).

La mayoría de los trabajos realizados sobre el análisis estático [Ceri y Widom, 1990] [Baralis et al., 1995] [Aiken et al., 1995] han usado el gráfico de desencadenamiento para solucionar el problema de los ciclos en la ejecución. El gráfico de desencadenamiento TG en la figura 2.12 muestra las interacciones del conjunto de

conflicto S de las reglas activas. Los nodos del gráfico representan las reglas de S. Un arco directo de la regla R_i a R_j significa que, existe la posibilidad de que R_i active R_j (Ri "*may trigger*" Rj). Si el gráfico no tiene ningún ciclo, la terminación de la ejecución está garantizada. Un ciclo en el gráfico significa que por lo menos existe una regla que se activa ella misma o activa el mismo conjunto de reglas. Por ejemplo, en la figura 2.12, la activación de las reglas R2 y R3 está garantizada para terminar, pero la activación de R1 o R4 puede iniciar una ejecución cíclica que no termina.

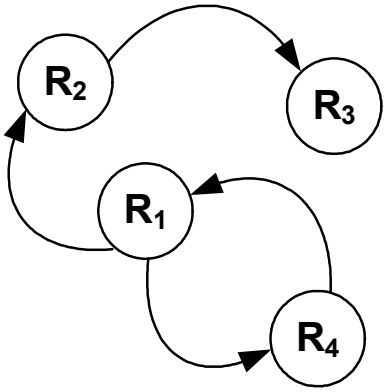


Figura 2.12. El gráfico de desencadenamiento TG.

TG ayuda a detectar ciclos entre las reglas activas, si no hay ningún ciclo, la terminación de ejecución está garantizada. TG se construye a través de un análisis sintáctico de cada regla en el conjunto, es decir, es una aproximación directa que depende de la relación "may trigger" entre dos reglas. Por ejemplo, si se inserta un arco entre las dos reglas R_i y R_j significa que la acción de R_i produce un evento monitorizado por la regla R_j. El análisis de terminación se centra en la identificación y la eliminación de los arcos que podrían introducir ciclos en TG [Debray y Hickey, 2000]. La redefinición de la regla causante del ciclo y la reconstrucción otra vez del gráfico TG, es una buena solución para verificar la terminación del conjunto S.

Para mejorar el análisis de la terminación, se utiliza también el gráfico de activación [Baralis y Widom, 2000] [Faria y Vidal, 1999]. El gráfico de activación (AG) para un conjunto de reglas S, es un gráfico directo dónde cada nodo corresponde a una regla $R_i \in S$. Este gráfico mejora el análisis de las reglas activas considerando la

influencia de sus condiciones en el comportamiento y durante el tiempo de ejecución. Un arco directo (R_i, R_j), con ($i \neq j$), indica que la acción de la regla R_i puede cambiar el valor de la condición de R_j de falso a verdadero, es decir, un arco (R_i, R_j) indica que la condición de la regla R_j puede ser verdadero después de la ejecución de la acción de R_i. De hecho, si no se produce ningún ciclo en AG, esto implica la ausencia de la no-terminación. Aunque, el gráfico AG investiga mejor el conjunto de conflicto de reglas que TG, sus requisitos son muy limitados. AG puede ser aplicado solamente si se detecta una acción que puede cambiar la realidad de una condición del valor falso al valor verdadero. Si no existe ninguna regla que pueda cambiar esta realidad, el gráfico AG no proporciona ninguna información adicional con respecto al gráfico TG.

2. Análisis de confluencia

Desde el punto de vista de las aplicaciones, la confluencia es un problema menos crítico y no es necesariamente indeseable [Ceri y Fraternalli, 1997]. Por ejemplo, en muchas aplicaciones, el estado final que se produce por un conjunto de reglas activas llega a un estado no planeado, pero el resultado puede ser aceptable. Por ejemplo, en una aplicación de empleados, cada tarea debe asignarse a un empleado, y varias tareas pueden permanecer sin asignación. Una política aceptable que puede llevarse a cabo por las reglas activas es asignar las tareas a los empleados sin preocuparse sobre el tipo de la tarea que lleva cada empleado. En este ejemplo, se puede aceptar cualquier estado final de la BD aunque la ejecución de las reglas no sea confluente.

Si se priorizan todas las reglas en un conjunto, o si existiera una clasificación total de las reglas, se garantiza la confluencia del conjunto [Baralis et al., 1998]. La confluencia también se garantiza si no existen dos reglas que se activan al mismo tiempo [Widom y Ceri, 1996]. Un conjunto de reglas es confluente cuando el estado final de la BD debe ser independiente del orden en el que han sido ejecutadas.

Para garantizar la confluencia, es necesario primero garantizar la terminación [Benazet et al., 1995], y segundo, asegurar que cada dos reglas en el conjunto estas son conmutativas. En la figura 2.13, las dos reglas R_1 y R_2 son conmutativas, porque

el resultado es igual para la transición del estado inicial S_1 al estado final S_2, si se activa R_1 primero y luego R_2, o si se activa R_2 primero y luego R_1. Una manera para analizar la confluencia es probar que todos los pares de reglas en el conjunto son conmutativas [Aiken et al., 1995] [Kim y Chakravarthy, 1995] [Widom y Ceri, 1996] [Baralis y Widom, 2000].

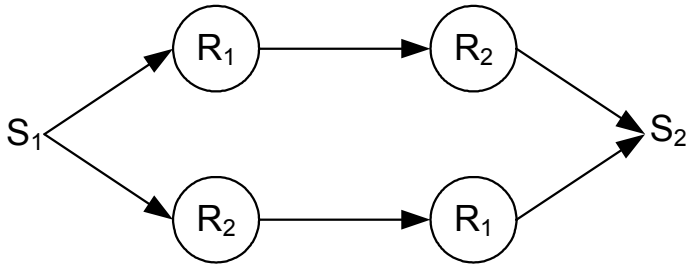


Figura 2.13. Las Reglas conmutativas.

En la figura 2.14 se muestra un ejemplo de análisis de confluencia presentado en [Aiken et al., 1995], donde las dos primeras reglas serán activadas por los cambios de la tabla SALES y realizan una modificación del Salary y Rank en la tabla EMP, respectivamente. La primera regla GOOD_SALES aumenta el sueldo (SALARY) de un empleado por 10, siempre que ese empleado obtenga unas ventas superiores a 50 durante un mes. La segunda regla GREAT_SALES, aumenta la categoría (RANK) de un empleado por 1 siempre que ese empleado obtenga unas ventas superiores a 100 durante un mes.

Aunque las dos reglas GOOD_SALES y GREAT_SALES pueden activarse al mismo tiempo, sus acciones no pueden afectar una a la otra, por eso, no se necesita garantizar la confluencia. Ahora, si existe una tercera regla RANK_RAISE que aumenta el sueldo de un empleado un 10% siempre que la categoría de ese empleado alcance 15, la situación cambiará.

Para garantizar la confluencia, está claro, que se requiere un orden relativo entre las reglas RANK_RAISE y GOOD_SALES, si se activan ambas al mismo tiempo. El orden de la ejecución de las acciones de estas dos reglas influye en el último sueldo de un empleado. Está claro que también es necesario un orden relativo entre las

reglas RANK_RAISE y GREAT_SALES, si se activan ambas al mismo tiempo. La ejecución de GREAT_SALES puede aumentar la categoría RANK de un empleado a 16 antes de que RANK_RAISE conceda el aumento del 10%.

Las dos reglas R_1 y R_2 son conmutativas, si satisfacen una de las siguientes condiciones [Aiken et al., 1995]:

1. R_i no activa R_j.
2. R_i no deshace los cambios producido por la regla R_j.
3. R_i no cambia la evaluación de la condición de R_j a verdadera.
4. R_i no borra los datos que satisfacen la condición de R_j.
5. Las acciones de las dos reglas R_i y R_j son conmutativas.

Aplicando estas reglas se puede asegurar la conmutatividad de dos reglas. Verificar estas condiciones es bastante sencillo, aunque son muy conservadores y probablemente podrían ser refinar utilizando un análisis más complicado, considerando otros casos. Para garantizar la confluencia entre dos reglas no conmutativas, los diseñadores deben asignar prioridades entre ellas. Se puede realizar el análisis de la conmutatividad manualmente o automáticamente. Sin embargo, ninguna herramienta está actualmente disponible para el análisis de la confluencia [Berghe, 2000].

```
CREATE RULE GOOD_SALES ON SALES
ON INSERT
DO UPDATE EMP
SET SALARY = SALARY + 10
WHERE ID IN(SELECT ID FROM INSERTED WHERE NUMBER>50);

CREATE RULE GREAT_SALES ON SALES
ON INSERT
DO UPDATE EMP
SET RANK = RANK + 1
WHERE ID IN(SELECT ID FROM INSERTED WHERE NUMBER>100);

CREATE RULE RANK_RAISE ON EMP
ON UPDATE(RANK)
DO UPDATE EMP
SET SALARY =  1.1 * SALAY
WHERE ID IN
(SELECT ID FROM NEW.UPDATED WHERE RANK = 15);
```

Figura 2.14. Ejemplo de la confluencia.

En algunos sistemas relacionales se garantiza automáticamente la confluencia por activar una sola regla en un momento determinado [Ceri y Fraternalli, 1997], como el caso del estándar SQL3.

2.3 Aproximaciones Existentes en Tecnología Activa

En este apartado se presenta una visión general de algunas aproximaciones existentes dentro de la tecnología activa. Por ello, en la sección 2.3.1 hemos elegido algunos prototipos de investigación para conocer las características de cada uno. Estos prototipos representan los marcos más importantes dentro de distintos modelos de datos. Se presentan el sistema ARIEL como uno de los prototipos más destacados en el modelo relacional y EXACT para el modelo orientado a objetos. Como metodología más representativa dentro del desarrollo de aplicaciones basadas en reglas activas se muestra la metodología IDEA. En la sección 2.3.2 se presentan las características activas que proporciona (los disparadores) el estándar SQL3, mostrando las interacciones entre estos disparadores y las acciones de integridad referencial y por último, en la sección 2.4.2 se presenta algunos SGBDs relacionales comerciales, finalizando con un resumen en el que se compara estos sistemas y los mecanismos activos del estándar SQL3.

2.3.1 Prototipos de Investigación

La idea de tener disparadores en una BD no es nueva. Estos mecanismos aparecieron primero en CODASYL en forma de *On Condition Do*. El sistema R (IBM Aplicación 1976-77) proporcionó los disparadores como un mecanismo para reforzar las restricciones de integridad [Gehani, 1991]. Durante varios años, muchos trabajos de investigación han tenido como objetivo reforzar los sistemas de BD tradicionales con capacidades activas [Berghe, 2000]. Por eso, se han desarrollando muchos prototipos de investigación y la tecnología activa ha sido aplicada con éxito a nuevos dominios de aplicación. A continuación se presentarán algunos prototipos de investigación relevantes, clasificados según los modelos de datos donde se hayan apoyado.

1. El sistema ARIEL

El sistema ARIEL es una aplicación desarrollada dentro de un SGBD relacional que implementa un sistema de reglas activas cuyas características principales son [Hanson, 1996].

- Diseño de un lenguaje conveniente para expresar las reglas activas.
- Diseño de un mecanismo eficiente para comprobar las condiciones de las reglas, y permitir el proceso rápido de la transacción.
- Integración de la comprobación de la condición y la ejecución de la regla con el sistema de proceso de transacción.

En ARIEL, las reglas se activan por las transiciones que actualizan la BD a través de una sentencia o una secuencia de sentencias ejecutadas por el usuario. Una sentencia de actualización de datos en ARIEL es una operación orientada al conjunto de tuplas, estas actualizaciones pueden ser inserción, borrado, o modificación. La granularidad mínima del proceso de las reglas es una operación orientada a nivel de tupla. Las sentencias ejecutadas pueden constituir una transacción entera, por eso la granularidad máxima del proceso de las reglas es una transacción entera. El proceso de las reglas se invoca automáticamente al final de cada transición y será parte de la misma transacción.

Una característica importante de ARIEL que lo distingue de STARBURST o POSTGRES es que el evento es opcional, es decir, ARIEL principalmente soporta las reglas de tipo condición-acción (CA) [Paton y Díaz, 1999]. Las condiciones son comprobaciones sobre el estado de la BD y las acciones pueden ser sucesiones arbitrarias de recuperación y actualización sobre cualquier información en la BD. Si se deshace la acción de una regla (ROLLBACK), se termina inmediatamente el proceso de la transacción y se regresa al estado original de la BD antes de la actualización.

El modelo de ejecución en ARIEL utiliza la estrategia *recogniza-act cycle* para controlar las reglas activas, esta estrategia contiene lo siguiente:

```
Initial match
While (rules left to run) {
Conflict resolution
Act
Match }
```

Cuando un evento ocurre, la fase MATCH encuentra todas las reglas que son elegibles para ejecutar. En la fase CONFLICT RESOLUTION se selecciona una de estas reglas para ejecutar. Al final, la fase ACT ejecuta las sentencias de las actualizaciones contenidas en la acción de la regla. Este ciclo de fases se repite hasta que no se encuentra ninguna regla activa.

En ARIEL, las reglas tienen prioridades numéricas. A cada regla se le asigna un número entre -1000 y 1000, si ningún número se especifica explícitamente entonces el valor que se asigna por defecto es 0. Las asignaciones no necesitan ser únicas. Cuando se activan las reglas múltiples, la resolución del conflicto en ARIEL procede de la siguiente forma:

- Se selecciona la regla con la prioridad más alta.
- Si dos o más reglas tienen la misma prioridad y sus condiciones son satisfactorias, estas reglas se ejecutan juntas, pero si estas condiciones son satisfactorias en distintas transiciones, entonces se consideran sólo aquellas que tienen condiciones satisfechas por la última transición.
- Si queda más de una regla, se selecciona una arbitrariamente.

Finalmente, cuando se procesan las reglas de ARIEL después de una transición, las reglas consideran la estrategia del efecto neto de las actualizaciones. En la mayoría de los casos, el efecto neto será igual que las actualizaciones individuales. Es decir, si una tupla se modifica en varias transiciones, el efecto neto es realizar sola una modificación. Si una tupla se modifica y luego se borra, el efecto neto es borrar la tupla original. Si una tupla se inserta y luego se modifica, el efecto neto es insertar la tupla modificada. Si una tupla se inserta y luego se borra, el efecto neto no realiza ninguna actualización.

Aunque el sistema activo ARIEL es muy potente y tiene muchas características que los SGBDs relacionales comerciales no tienen, las reglas activas de este sistema no son compatibles con el estándar SQL3; nuestro principal objetivo en esta tesis doctoral. Tampoco tiene ningún soporte en una herramienta CASE comercial que puede facilitar a los usuarios el desarrollo de las reglas activas o detectar la no-terminación cuando no existe prioridades en su ejecución.

2. El modelo EXACT

La diversidad de los sistemas activos hace que encontrar una estrategia de ejecución común para todas las aplicaciones sea una tarea complicada. Por eso, las estrategias de ejecución flexibles se convierten en un requisito esencial en el desarrollo de un SGBD activo. EXACT, es un modelo EXtensible de SGBD orientado a objeto ACTivo [Díaz y Jaime, 1997] [Díaz, 1998] que permite aplicar las ventajas del paradigma de modelo orientado a objetos al SGBD activo y tiene una estrategia flexible de ejecución.

En general, el objetivo del modelo orientado a objeto es integrar los aspectos estructurales y los comportamientos de las reglas en los modelos conceptuales [Winter, 1998]. De esta manera, usando el modelo OO no sólo las reglas se describen en el modelo conceptual, sino también la estrategia del proceso de estas reglas. Cuando se describe el proceso de las reglas usando el modelo OO, este modelo dispondrá de una jerarquía de modelos de ejecución que permite al usuario elegir el modelo más conveniente con la semántica de la aplicación. Donde el usuario conoce la semántica del concepto soportado, por eso puede no sólo incluir la definición básica de las reglas sino también la información del control sobre cómo estas reglas tienen que ser ejecutadas. El modelo de ejecución de las reglas no suele aplicarse a una sola regla, porque es compartido con un conjunto de reglas que soportan el mismo concepto o funcionalidad (por ejemplo, el mantenimiento de restricción de integridad, etc.).

El objetivo del proyecto EXACT es la definición del modelo de ejecución de las reglas activas caracterizado por siguientes propiedades:

- Flexible: se puede encontrar distintas estrategias de ejecución de distintos conjuntos de reglas activas, dónde cada conjunto de estas puede soportar una semántica distinta.
- Declarativo: se puede definir el modelo de ejecución de las reglas activas a través de un conjunto de parámetros en lugar de codificar este modelo según la estrategia de la ejecución deseada.
- Extensible: se puede extender el modelo de ejecución fácilmente refinando las características existentes o introduciendo otras nuevas.

EXACT es un sistema activo construido sobre BDOO ADAM [Paton, 1989]. Este sistema soporta las reglas como objetos de primera clase (*first-class objects*), es decir, las reglas se definen y se tratan como cualquier otro objeto en el sistema usando atributos y métodos. Las clases de reglas pueden colocarse como una jerarquía tal y como se muestra en la figura 2.15. La clase de la regla genérica (*generic-rule*) almacena la descripción común de una regla a través de los atributos evento, condición, acción, prioridad, y IS-IT-ENABLED, este último atributo es un atributo booleano que describe el estado de la regla, es decir, si la regla se encuentra activa o no. La regla puede ser SWITCHED ON o SWITCHED OFF a través de la modificación del valor de este atributo. En cuanto al comportamiento activo, se canaliza a través de los métodos: *evaluating-condition* el cual comprueba si la condición de la regla está satisfecha, *execution-action* que ejecuta la acción de la regla, y el método FIRE que invoca la evaluación de la regla y, sí está satisfecha, invoca la ejecución de su acción. Todos estos atributos y métodos son reunidos en la clase de la regla genérica que puede especializarse para responder a las necesidades, donde estos atributos y métodos se introducen con cada una de las subclases *system-rule, integrity-rule,* y *dynamic-display-rule* que representan un uso particular de la regla.

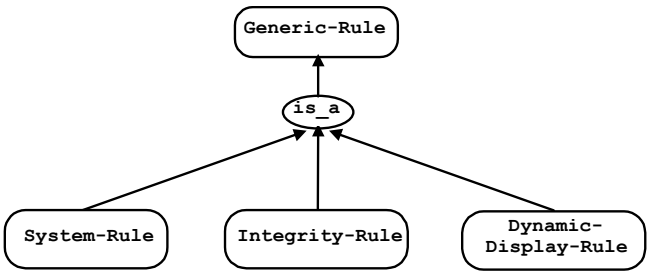


Figura 2.15. Jerarquía de las reglas en EXACT

La aproximación del modelo EXACT consiste en dividir las reglas según la clase a la que pertenecen. Cada clase realiza un uso determinado (por ejemplo, mantenimiento de restricciones de integridad, etc.), y este uso no sólo incluye la descripción de la regla, sino también, el comportamiento de esta regla en tiempo de la ejecución. Esto se realiza usando las metaclases. La figura 2.16 muestra los tres niveles de definición del comportamiento activo en EXACT.

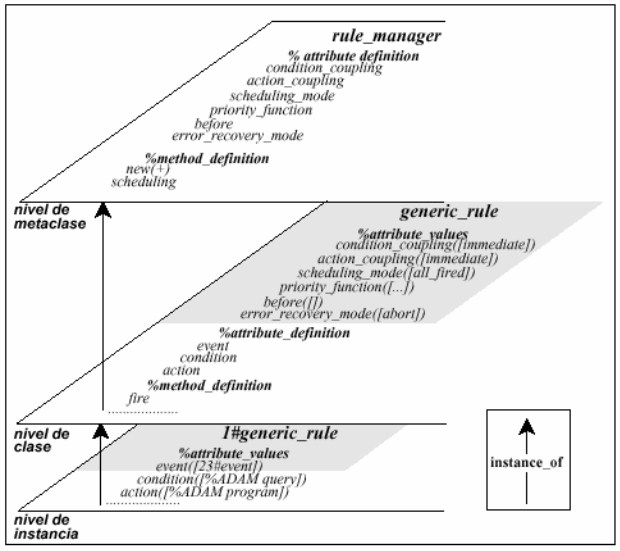


Figura 2.16. Los tres niveles de definición del comportamiento activo EXACT

- En la metaclase *rule-manager* se describe el modelo de ejecución a través de un conjunto de meta-atributos que son: modo evento-condición-acoplamiento, modo condición-acción-acoplamiento, modo de planificación, modo de recuperación de error, y modo de resolución de conflictos. En este último atributo se almacena las

prioridades de ejecución de las reglas.

- En la clase *rule-manager* se almacena la descripción de dos métodos principales que son; el método NEW, que se utiliza para crear instancias de reglas, y el método *scheduling*, que se usa para crear el conjunto de conflicto y dispara las reglas de forma adecuada.
- La instancia 1#GENERIC-RULE donde se definen las reglas individuales.

En la ejecución de las reglas pueden activarse en cascada. La cascada de una regla puede suspender la ejecución de la regla principal, y entrar en un nuevo nivel del proceso de ejecución donde se consideran los efectos de la nueva regla. Se utiliza para solucionar el conjunto entre las reglas activadas, un árbol de ejecución que representa el ciclo de los eventos señalados y las reglas activadas por estos eventos. Por ejemplo, el evento E_1 activa las reglas $\{R_1, R_2, R_3\}$ y el evento E_2 activa las reglas $\{R_4, R_5\}$. Ahora, si durante la ejecución de R_1 se produce E_2, en este caso se suspende la ejecución de R_1 hasta que las reglas asociadas con el evento E_2 terminan (o sea R_4 y R_5). Una vez que todo este proceso termina, R_1 puede continuar su ejecución y el proceso acaba con la ejecución de R_2 y R_3.

Las inconvenientes de este modelo, EXACT, es que algunas dimensiones que caracterizan la ejecución de un módulo aparecen en la expresión de las reglas. Esto puede confundir al programador, y puede conducir a redundancias o incluso a inconsistencias. También el modelo estándar de la ejecución es bastante sencilla (existe sólo una estrategia de la ejecución para un módulo de regla) y el programador tiene que codificar los métodos para llevar a cabo el modelo de la ejecución deseada [Coupaye y Collet, 1998]. Además, el modelo EXACT es un sistema activo orientado a objeto y no tiene extensión a los sistemas relacionales, ni las reglas activas en él son compatibles con el estándar SQL3. Como en el sistema ARIEL, el modelo EXACT no tiene ningún soporte en una herramienta CASE comercial que puede facilitar a los usuarios el desarrollo de las reglas activas o detectar el problema de la no terminación cuando este ocurra.

3. Metodología IDEA

La Metodología IDEA es un nuevo paradigma para diseño de software y por lo tanto, el diseño de una BDOO forma parte de la misma [Ceri et al., 1995] [Ceri y Fraternalli, 1997] [Ceri et al., 1997]. El objetivo principal de este proyecto es investigar y favorecer el uso del modelo orientado a objeto y las tecnologías de las reglas activas en los sistemas de información. La Metodología IDEA contiene varias herramientas para ayudar a los diseñadores a lo largo de todas las fases del ciclo de vida de las aplicaciones. La metodología considera que las reglas son componentes que permiten una representación natural de las necesidades del usuario y facilitan el diseño, implementación, y el mantenimiento de Las BDs.

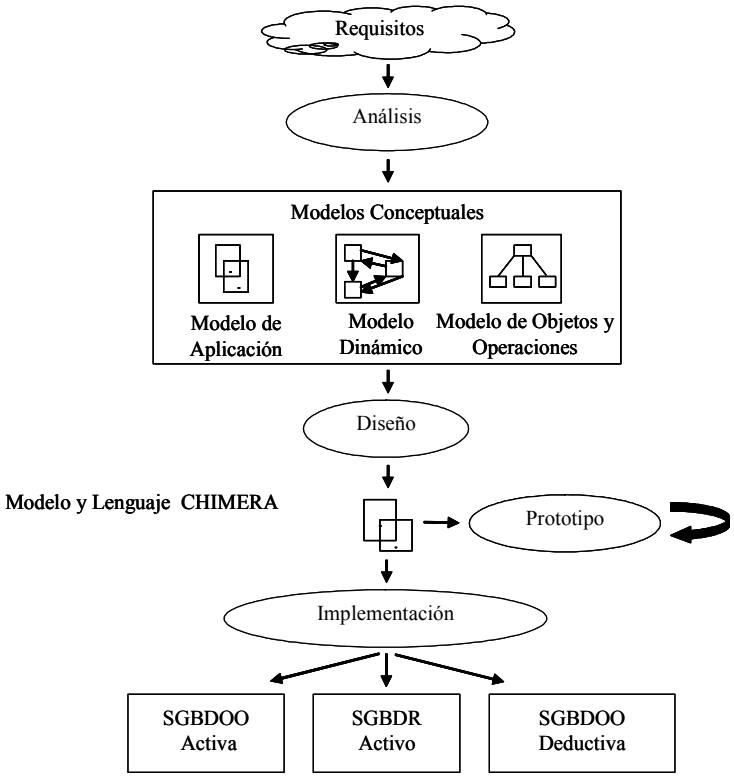


Figura 2.17. Las fases de la metodología IDEA

En la figura 2.17 se muestra la Metodología IDEA que cubre las fases tradicionales del desarrollo del software, es decir, análisis, diseño, prototipado, e implementación de las aplicaciones activas, teniendo en cuenta las ventajas de los modelos del

desarrollo de BD orientadas a objetos. El interés principal y la mayoría de las contribuciones originales de la Metodología de IDEA son el diseño, y la implementación de reglas deductivas y reglas activas que significativamente enriquecen la semántica soportada dentro de Las BDs.

- **Fase de análisis:** El análisis en la metodología IDEA se deriva de las metodologías OMT [Rumbaugh, 1991], FUSIÓN [Coleman et al., 1994], SYNTROPY [Cook y Daniels, 1994], y el Método BOOCH [Booch, 1994]. Se realiza el análisis en tres perspectivas: la estructura (definición de atributos y relaciones), conocimiento (definición de las restricciones y los objeto dinámicos) y los requisitos funcionales (descripción de servicios deseados). En general, y como hemos dicho antes, la Metodología IDEA no tiene como interés principal el enfoque en la fase del análisis, así cualquier diseñador o compañía que está acostumbrado a usar un método de análisis específico y desea usar la Metodología IDEA no tiene porque cambiar su sistema de análisis.
- **Fase de diseño:** Esta fase incluye la especificación de las reglas. La Metodología IDEA usa el lenguaje de *Chimera* [Ceri y Manthey, 1994] [Widom y Ceri, 1996] que consiste de un modelo conceptual (*Chimera Model*) que proporciona las facilidades del modelo orientado a objeto, y un lenguaje conceptual (*Chimera Language*) que proporciona las definiciones de los datos, reglas y restricciones. En esta fase, las restricciones declarativas (*declarative features*) se transfieren a especificaciones de procedimientos de *Chimera*. Por ejemplo, las acciones de integridad referencial del modelo de objeto se convierten en disparadores de *Chimera* que mantienen las relaciones correctas entre los objetos, después de realizar una operación de actualización. En particular, esta fase se divide en, primero el diseño del esquema teniendo en cuenta principalmente las interrelaciones entre los tipos, clases, relaciones, y operaciones, y segundo el diseño de las reglas, que se subdivide en el diseño de las reglas deductivas y el diseño de las reglas activas. El diseño de las reglas activas define los componentes principales como: evento, condición, acción, y prioridad, y sus opciones semánticas de: efecto neto y los modos de acoplamientos.

- **Fase de prototipado:** Esta fase produce una versión ejecutable de la aplicación activa y comprueba la coincidencia de la aplicación con los requisitos de usuario. Las actividades principales realizadas durante el prototipado son las siguientes:
  - La modulación: Un problema serio del diseño de las reglas activas es manejar y controlar la complejidad semántica de un conjunto grande de reglas. La modulación es una técnica que permite hacer la gestión de las reglas más sencilla. Un conjunto de las reglas más pequeñas que construye un módulo es más fácil de manipularlo que la colección de todas las reglas de la aplicación. Se pueden organizar reglas de un tipo de restricciones específicas en una capa de reglas que controla estas restricciones. Si las capas se realizan según un proceso bien definido, se puede asegurar la calidad de unir todas estas capas de una aplicación [Baralis et al., 1996]. Las reglas se dividen en capas para que el diseñador pueda actuar localmente sobre el comportamiento de cada una de estas capas y puede controlar el comportamiento de todas las reglas en las capas. Verificar la consistencia de cada capa consiste en ordenar sus reglas para evitar resultados inesperados.
  - Análisis de Reglas: El análisis de las reglas en IDEA está basado en las aproximaciones de [Baralis et al., 1995]. IDEA soporta el análisis basado en el gráfico de desencadenamiento (*triggering graph*) (sección 2.2.4.1). Para un gráfico dado, el análisis consiste en comprobar cada arco (R_i, R_j), si es posible conocer que la condición de R_j no será verdadera por la ejecución de R_i, en este caso aunque R_j se activa por la acción de R_i se puede descartar este arco del gráfico. El análisis de las reglas se realiza en tiempo de compilación, es decir, cuando se crean las reglas, por lo tanto, es una técnica estática. Los problemas que se solucionan a través de este análisis son de terminación y de confluencia.
- **Fase de implementación:** La implementación es el proceso de mapear las especificaciones conceptuales a esquemas, objetos, y reglas activas del SGBD elegido. Mapear es el proceso de transformar el modelo de objeto de *Chimera* a un SGBD comerciales. Por ejemplo, en el caso de ORACLE, el modelo de

objetos de *Chimera* se transforma a tablas relacionales. Durante este proceso parte de la semántica del esquema orientado a objeto generado por *Chimera* se perderá porque las características del SGBD seleccionado influyen mucho en el proceso de la implementación. Algunas veces, el problema de transformar las reglas de *Chimera* a los disparadores de los SGBDs relacionales comerciales será más difícil de solucionar por las limitaciones de estos sistemas. Se puede realizar esta fase a través de dos aproximaciones como se muestra a continuación:

- *Meta-Triggering*: Esta aproximación se caracteriza en que la mayor semántica de la BD puede traspasarse a un *meta-trigger* que propaga eventos, resuelve conflictos de las prioridades, e invoca procedimientos. Por ejemplo, según esta aproximación la metodología IDEA trata las limitaciones de ORACLE a través de crear un procedimiento almacenado para cada regla de *Chimera* y se lleva a cabo la parte activa de la regla con las estructuras adicionales (principalmente los disparadores y las tablas especializadas). Esta aproximación exige agregar estructuras de datos auxiliares y disparadores para que estos procedimientos almacenados sean invocados correctamente. La ventaja de esta solución es que los procedimientos almacenados conserven la semántica de las reglas *Chimera*. La desventaja consiste en que la activación de los disparadores es indirecta, con un impacto sobre el rendimiento y la transformación más difícil de entender e implementar.
- *Macro-Triggering*: En esta aproximación se agregan las reglas de *Chimera* que se definen sobre una tabla, para constituir un solo disparadores en el SGBD solucionado. Por ejemplo, en el caso de ORACLE, las reglas de *Chimera* se clasifican según los eventos y los objetos. Estas reglas se reúnen para construir un disparador de ORACLE. Esta aproximación se realiza principalmente cuando el conjunto de reglas es pequeño y poca su complejidad.

2.3.2 El Estándar SQL3

Después de haber estudiado los prototipos más relevantes en el marco de la BD activas, en este apartado se presentarán las características activas del SQL3. El estándar más utilizado actualmente para las BDs relacionales es el especificado con SQL-92. Desde inicios del año 2000 se publicó el estándar más reciente de SQL, conocido como SQL99. Esta versión ha sido revisada por el estándar 2003 [Eisenberg y Melton, 2004] en el cual se incluyen aspectos objeto-relacionales para el manejo de BD, mezclando aspectos de los paradigmas relacional y orientado a objetos para su desarrollo.

SQL3 reconoce la noción de BD activa. Esto es facilitado por lo que se conoce como TRIGGERS (disparadores). Un disparador en SQL3 es una regla de evento-condición-acción que se activa por una transición de estado de la BD, como se muestra en la figura 2.18 [Melton y Simon, 2002]. Actualmente, muchos sistemas de BD relacionales comerciales como ORACLE 9i [Oracle, 2005], IBM DB2 [DB2, 2005], MICROSOFT SQL-SERVER 2005 [Microsoft, 2005],...etc., mantienen el soporte de los disparadores, pero en casi todos estos sistemas, no sólo la sintaxis de las especificaciones del disparador es bastante diferente del estándar sino también sus funcionalidades [Türker y Gertz, 2000].

El estándar SQL3 recoge los primeros pasos en la estandarización de estos mecanismos. Pero desgraciadamente, cuando nació el estándar había productos comerciales que ya soportaban mecanismos activos y por eso, el estándar se definió conforme a las características asociadas a estos productos. Por tanto, el estándar adolece de algunas de las carencias detectadas en estos productos y no contempla todas las características de la tecnología activa [Ceri et al., 2000].

La aplicación de la tecnología activa a los SGBDs relacionales tiene otro problema añadido ya que se han de considerar las restricciones de integridad referencial definidas en las tablas y sus interacciones con las reglas activas definidas [Türker y Gertz, 2000].

```
<trigger-def > ::=  CREATE TRIGGER <trigger-name>
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE [OF <attribute-list>]}
ON <table-name> [REFERENCING <old-or-new-values-list>]
[FOR EACH {ROW | STATEMENT}]
[WHEN ( <search-condition> )]
<triggered-SQL-statement>

<old-or-new-values>::={OLD | NEW}
[ROW] [AS]<correlation-name> | {OLD | NEW} TABLE [AS] <table-alias>

<triggered-SQL-statement> ::= <SQL-procedure-statement>
| BEGIN ATOMIC
<SQL-procedure-statement-list>
END
```

Figura 2.18. La definición de un disparador en el estándar SQL3

1. Modelo de conocimiento

Un disparador del estándar SQL3 esta formado por cuatro componentes: una tabla asociada, una operación que causa el disparo (evento), una condición, y una acción. El disparador se activa siempre cuando comienza una operación sobre su tabla asociada y la condición del disparador se evalúa a verdadero. Cuando un disparador se activa, su acción se ejecuta como parte de la ejecución.

Los componentes del modelo de conocimiento de sistema activo del estándar SQL3 se muestran a continuación:

- El evento de un disparador es primitivo puede ser INSERT, DELETE, UPDATE. Si el evento es una operación de modificación UPDATE entonces hace falta una lista de columnas asociadas con esta operación. Un disparador definido sobre una tabla T se activa siempre que:

- Una operación de inserción, borrado, o modificación actualiza la tabla T.
- Una operación de inserción, borrado, o modificación actualiza una vista definida sobre T (*updatable view*).
- Una acción de integridad referencial (*referential action*) sobre la tabla T como resultado de una operación de borrado o modificación sobre la tabla a la que referencia.

- La condición del disparador puede ser cualquier predicado de SQL. Los predicados pueden incluir subconsultas (*subquery*), funciones definidas por los usuarios (*user-defined function*), etc. La especificación de la condición del disparador es opcional. Si se omite la condición del disparador esto significa que su condición es implícitamente verdadera.
- La acción del disparador es una lista de operaciones de SQL dentro de un bloque (BEGIN/SQL BLOCK /END). La acción también puede incluir, SQL/PSM los procedimientos de las estructuras (IF, CASE, WHILE, ASSIGNMENT STATEMENTS, etc.), invocar a los procedimientos almacenados y las funciones definidos por usuarios. Las operaciones que no se pueden incluir en la acción del disparador son aquellas relacionadas con los esquemas, conexiones, y transacciones (COMMIT, ROLLBACK, etc.).

Los nombres del disparador deben ser únicos con respecto a otros disparadores en el mismo esquema. Los nombres del disparador no necesitan ser únicos con respecto a otros objetos del esquema, como las tablas, vistas, y procedimientos. Por ejemplo, una tabla y un disparador pueden tener el mismo nombre, sin embargo, para evitar la confusión, esto no se recomienda.

2. Modelo de ejecución

Las características del modelo de ejecución de sistema activo del estándar SQL3 se muestran a continuación:

- Los modos de acoplamiento, evento-condición y condición-acción, son inmediatos y se ejecutan en la misma transacción con la operación que activa el disparador según vimos en sección 2.2.3.1.
- Cada disparador es asociado con un tiempo de activación, que determina si el disparador se activa antes o después (BEFORE/AFTER) del evento. La especificación del tiempo de activación del disparador es obligatoria. En SQL3 no se permite actualizar la BD a través de las acciones de los disparadores de tipo (BEFORE), para que no produzca ningún nuevo cambio en el estado por la ejecución [Kulkarni, 1998]. Las acciones de este tipo de disparadores se ejecutan inmediatamente después del evento y se verifican completamente antes de la actualización de la sentencia ejecutada. Pero, para cumplir la semántica deseada por la actualización, deben verificarse todas las restricciones de integridad inmediatas después de la realización de la sentencia original y todas las acciones activadas por esta sentencia [Horowitz, 1994]. Por lo tanto, el estándar SQL3 recomienda que los disparadores de tipo BEFORE se puedan utilizar para leer de la BD, y puedan utilizar las sentencias de asignación para modificar los valores introducidos [Cochrane et al., 1996]. Por ejemplo, "Asegúrese que el sueldo nunca sea mayor de 5000", en este caso, se puede emplear un disparador de inserción de tipo BEFORE-FOR EACH ROW para asignar al sueldo 5000 si el nuevo valor es mayor de 5000. Por el contrario, los disparadores AFTER que se ejecutan después del evento, son especialmente útiles para realizar las actualizaciones en otras tablas, o para invocar funciones que realizan tareas útiles fuera de la BD.
- SQL3 soporta los dos tipos de granularidad de transición: tupla (FOR EACH ROW) y conjunto (FOR EACH STATMENT). Los disparadores de tipo tupla se ejecutan una vez por cada tupla, mientras los disparadores de tipo sentencia se ejecutan una vez por todo el conjunto de tuplas afectadas por el evento. La especificación de granularidad del disparador es opcional cuando no viene especificada, la granularidad por defecto es a nivel de conjunto. El conjunto de tuplas afectadas por un evento a veces puede ser vacío. Por ejemplo, durante la

ejecución de una sentencia de modificación (UPDATE SET C=10 WHERE...), la evaluación de la cláusula (WHERE) puede producir cualquier resultado vacío. Un disparador de tipo tupla no se ejecuta si el conjunto de las tuplas afectadas está vacío. Sin embargo, un disparador de tipo sentencia, siempre se ejecuta una vez, aunque el conjunto de tuplas afectadas esté vacío.

- SQL3 soporta dos tipos de valores de transición por variables o por tablas. Estos valores son OLD y NEW cada uno de los cuales debe identificarse por un único nombre en la cláusula de referencias (REFERENCING). El valor identificado por OLD captura el estado original de las tuplas afectadas (es decir, antes de las modificaciones del evento que se aplica a la BD). El valor identificado por NEW captura los nuevos valores de todas las tuplas que resultarían inmediatamente después de que las modificaciones se apliquen a la BD. Se observa que todos los valores de la transición no son válidos para todos los tipos de disparadores. Los disparadores que se especifican para los eventos de inserción pueden ver sólo los valores nuevos, los disparadores que se especifican para los eventos de borrados pueden ver sólo los valores antiguos, y los disparadores que se especifican para los eventos de modificación, pueden ver los valores antiguos y nuevos. Además, sólo se pueden especificar estas variables de transición para los disparadores de tipo tupla.

- SQL3 permite activar múltiples disparadores a la vez. Puede haber varios disparadores que tienen la misma tabla asociada, el mismo evento, y el mismo tiempo de activación. Otro ejemplo de disparadores múltiples puede producirse en las actualizaciones de las restricciones de integridad referencial que se lleva a cabo durante la ejecución de otros eventos, que activan otros disparadores adicionales a la lista de los disparadores activados. Otra situación dónde los disparadores múltiples pueden ser activados es cuando el usuario usa el modo diferido de comprobación de restricciones. En este caso, se verifican las restricciones cuando la transacción está lista para ser realizada (COMMIT). Esto puede producir la ejecución de acciones de múltiples referencias en tablas que a su vez podrían activar disparadores múltiples al mismo tiempo.

- Para ordenar la ejecución de disparadores múltiples, el SQL3 sigue un esquema de prioridad estática determinada por el sistema, basada en el tiempo de creación del disparador, es decir, el orden de los disparadores depende del orden ascendente del tiempo de creación de los mismos. Así, un disparador que se agrega recientemente a una BD, se ejecuta después de todos los otros disparadores que se han definido previamente. Se nota que este esquema garantiza el orden global. Es importante observar que el modelo de ejecución especificado actualmente en el estándar SQL3 no tiene un mecanismo dependiente utilizado para ordenar los disparadores. Por eso, esta tarea se realiza dependiendo de que los usuarios tengan que asegurar la no terminación en la cascada de la ejecución de los disparadores y que el estado final de la BD sea consistente.
- El mecanismo de recuperación del error en SQL3 es deshacer las actualizaciones (ROLLBACK).
- El proceso en cascada del SQL3 es recursivo (sección 2.2.3.1).
- Por último, el SQL3 no sigue la política del efecto neto explicada en la sección 2.2.3.1.

3. Acciones de integridad referencial

En muchas de las propuestas dentro de las BDs activas se ignora uno de los aspectos más importantes en los sistemas activos: integrar las restricciones de integridad referencial y los disparadores [Cochrane et al., 1996]. El modelo de ejecución de muchos prototipos activos en BD, como ARIEL [Hanson, 1996], HiPAC [Dayal, et al., 1996], y STARBURST [Widom, et al., 1991], consideran sólo las reglas activas ECA, y la activación de estas depende sólo de los eventos. Mientras que uno de los aspectos más importantes que distinguen el modelo de ejecución del estándar SQL3 con otros modelos de ejecución, es la interacción entre los disparadores y las acciones de la integridad referencial [Kulkarni, 1998].

En las BDs relacionales, los objetos se representan por tuplas y las conexiones entre ellos se representan a través de referencias. Estas referencias se refuerzan en las BDs relacionales por las restricciones de integridad referencial [Markowitz, 1990]. La integridad referencial es el predicado sobre la BD que el sistema debe evaluar como verdadero, sino las restricciones son violadas y el estado de la BD es inconsistente [Ceri y Widom, 1990]. Estas restricciones se usan para prohibir ciertos cambios en los datos o para mantener la semántica requerida. Sin embargo, el problema se produce cuando tengamos disparadores asociados a tablas referenciadas por acciones de integridad referencial. Estas acciones son parte de las restricciones estáticas que incluyen el conjunto de restricciones del modelo relacional (KEY UNIQUE/PRIMARY, NOT NULL, CHECK, REFERENTIAL INTEGRITY, y ASSERTIONS).

Para definir una acción de integridad referencial en SQL3 se usa la definición de la clave ajena (FOREIGN KEY) con las siguientes especificaciones:

- NO ACTION (RESTRICT): No se puede borrar ó modificar una tupla de la relación referenciada, o no se puede modificar una clave primaria de una tupla en la relación referenciada, si esta tupla está relacionada con otra en la relación que referencia.
- Opciones de CASCADE: Cuando una tupla se borra ó se modifica en la relación referenciada, o cuando la clave primaria de la tupla referenciada se modifica, se borra o se modifican todas las tuplas relacionadas en la relación que referencia.
- SET DEFAULT y SET NULL: Cuando se borra ó se modifica una tupla de la relación Referenciada, o cuando se modifica una clave primaria de la relación referenciada, todas las claves ajenas de las tuplas relacionadas en la relación que referencia se modifican poniendo el valor por defecto o nulo.

Por las actualizaciones de estas tablas puede que se activen distintos disparadores, en la tabla 2.2 se muestran la relación entre los tipos de disparadores con cada acción de integridad referencial. Por ejemplo, en el caso de tener dos tablas relacionadas a través de una clave ajena y existe la definición de la acción ON DELETE SET

NULL entre ellas. Cuando se borra una tupla de la tabla referenciada en este caso se pueden activar los disparadores de borrado sobre la tabla referenciada y también, en cascada los disparadores de modificación que actúan sobre la clave ajena en la tabla que referencia ya que la clave se modificará a nulo.

Tipo de acción de integridad referencial sobre la tabla referenciada	**Tipo de disparadores activados por las actualizaciones de las acciones de integridad referencial**
ON DELETE NO ACTION ON DELETE RESTRICT	Disparadores de borrado sobre la tabla referenciada
ON UPDATE NO ACTION ON UPDATE RESTRICT	Disparadores de modificación sobre la tabla referenciada
ON DELETE SET DEFAULT ON DELETE SET NULL	Disparadores de borrado sobre la tabla referenciada Disparadores de modificación sobre la tabla que referencia
ON UPDATE SET DEFAULT ON UPDATE SET NULL	Disparadores de modificación sobre la tabla referenciada Disparadores de modificación sobre la tabla que referencia
ON DELETE CASCADE	Disparadores de borrado sobre la tabla referenciada Disparadores de borrado sobre la tabla que referencia
ON UPDATE CASCADE	Disparadores de modificación sobre la tabla referenciada Disparadores de modificación sobre la tabla que referencia

Tabla 2.2. Los disparadores activados por cada acción de integridad referencial

2.3.3 SGBDs Relacionales Comerciales

Los SGBDs han hecho posible el almacenamiento de gran volumen de datos y permiten a los usuarios manipular estos datos de una manera eficaz y controlada. Los SGBDs convencionales son pasivos, es decir, que los datos se crean, se recuperan, se modifican y se borran como respuesta a las peticiones emitidas por los usuarios de una forma directa o indirecta por los programas de aplicaciones. Las tendencias en la tecnología de BD están extendiendo los SGBDs con nuevas funcionalidades para cubrir aplicaciones más avanzadas. Un avance muy significativo es la incorporación de mecanismos activos a los SGBDs tradicionales, es decir, que el propio SGBD sea capaz de realizar automáticamente ciertas funciones en la respuesta a ciertos eventos con la satisfacción de ciertas condiciones. Los sistemas pasivos soportan de forma limitada las restricciones de integridad, por ejemplo, la especificación de valores únicos (*key constraints*), o las restricciones de integridad referencial (*referential integrity constraints*). Sin embargo, muchas aplicaciones requieren incorporar más restricciones a las anteriores, como por ejemplo, la especificación de todos los valores que deben estar dentro de un cierto rango (Dominio), o producir un cierto valor por una función agregada en los datos, o la autorización,..., etc. Actualmente, hay varios sistemas comerciales relacionales que soportan reglas activas (TRIGGERS), sin embargo, la implementación de estas reglas es diferente con respecto a la sintaxis, semántica, y capacidades dependiendo del SGBD utilizado [Türker y Gertz, 2000]. Su funcionalidad es bastante limitada comparándola con los sistemas orientados a objetos, sin embargo, proporciona comportamiento activo relativamente complejo. En general, las capacidades de los SGBDs relacionales activos (comerciales) sufren algunas limitaciones que a continuación exponemos [Widom y Ceri, 1996]:

- La falta de estandarización hace que exista distintas formas de especificación de una regla tanto en la sintaxis como en el comportamiento. Como consecuencia esto afecta a la portabilidad de la BD a distintos sistemas gestores.

- La semántica de ejecución es poco clara. Suele ser difícil anticipar cómo se comportarán las reglas cuando varias de ellas siguen diferentes opciones como nivel de tupla, nivel de sentencia, o son de ejecución inmediata o diferida.
- La ausencia de características avanzadas similares a las incluidas en los prototipos de investigación, como técnicas de composición de eventos, modo de acoplamientos, el efecto neto, etc.
- La inclusión de ciertas restricciones a la hora de ejecutar las reglas, como por ejemplo, el número máximo de reglas que se pueden definir o las interacciones entre éstas.

Los SGBDs activos han constituido una de las áreas de investigación más importantes y relevantes durante finales de los ochenta y principios de los noventa. Sin embargo, y como hemos mostrado en la sección anterior, la funcionalidad de los disparadores en estos sistemas generalmente es limitada si comparamos con los prototipos de investigación de BDs activas descritos en el capítulo anterior. Las razones de esta diferencia puede ser que estos sistemas se han desarrollado para proporcionar algunos aspectos y servicios de distintos vendedores. Cada vendedor tiene amplia variedad de productos que pueden usarse directamente con la BD, por eso para estos vendedores no es necesario incluir estos productos como parte de un estándar [Bloor, 2001].

En las siguientes secciones se presentan algunas características en el tratamiento de reglas activas de cuatro sistemas relacionales relevantes considerando las diferencias y las comparaciones entre el estándar SQL3 y estos sistemas.

1. ORACLE 9i

Los disparadores en ORACLE 9i son procedimientos que se almacenan en la BD, y se ejecutan implícitamente ante una operación de actualización específica (véase la figura 2.19). Estos disparadores se ejecutan cada vez que el correspondiente evento ocurre. El evento del disparador puede ser una sentencia (INSERT, DELETE, y UPDATE). Cuando se define el evento modificación (UPDATE) debe asociarse con

el nombre de una o más columnas en la tabla correspondiente. Los eventos compuestos son limitados en ORACLE 9i, donde se permite solo la disyunción entre los eventos.

Las opciones BEFORE y AFTER especifican cuándo se ejecuta el cuerpo del disparador respecto a la sentencia que lo activa, igual que en el estándar SQL3. Además, ORACLE proporciona un tipo adicional de disparador, los disparadores de sustitución (INSTEAD OF), estos disparadores son válidos sólo para vistas (VIEW).

Dentro de un disparador en el que se disparan distintos tipos de eventos DML (INSERT, UPDATE y DELETE), hay tres funciones booleanas que pueden emplearse para determinar de qué operación se trata. Estos predicados son INSERTING, UPDATING y DELETING, si la operación que activa el disparador es de inserción, modificación o borrado respectivamente.

La funcionalidad de la cláusula REFERENCING que permite crear los variables de transición (OLD, NEW) es similar a la cláusula del estándar SQL3 excepto que ORACLE no tiene variables de transición de tipo tabla.

Si se especifica un disparador de tipo FOR EACH ROW, este se activará una vez por cada tupla de la tabla afectada por el evento. Al no especificar la granularidad del disparador esto significa que implícitamente su granularidad es de nivel sentencia exactamente como en el estándar SQL3.

La cláusula WHEN incluye el predicado que tiene que ser verificado por el disparador, este predicado no puede ser incluido en la definición de un disparador de sentencia. El predicado se evalúa para cada tupla afectada por el evento. Si el predicado se evalúa a verdadero, la acción del disparador se ejecuta sobre dicha tupla. Si un error predefinido o una excepción se levanta durante la ejecución de un disparador, se deshace todas las actualizaciones realizadas por la sentencia ejecutada.

El cuerpo de un disparador es un bloque PL/SQL. Cualquier orden que sea legal en un bloque PL/SQL, es legal en el cuerpo de un disparador, con las siguientes restricciones:

- Un disparador no puede emitir ninguno comando de control de transacciones

(COMMIT, ROLLBACK o SAVEPOINT). El disparador se activa como parte de la ejecución del evento que provocó el disparo, y forma parte de la misma transacción que dicho evento. Cuando el evento que provoca el disparo es confirmado o cancelado, se confirma o cancela también las actualizaciones realizadas por el disparador.

- Por razones idénticas, ningún procedimiento o función llamado por el disparador puede emitir órdenes de control de transacciones.

```
CREATE [OR REPLACE] TRIGGER [schema .] trigger_name
{ BEFORE | AFTER | INSTEAD OF }
{ dml_event_clause
| { ddl_event [OR ddl_event]...
  | database_event [OR database_event]...
  }
  ON { [schema .] SCHEMA | DATABASE }
}
[WHEN ( condition ) ]
{ pl/sql_block | call_procedure_statement }

dml_event_clause::=
{ DELETE | INSERT | UPDATE [OF column [, column]...] }
[OR { DELETE | INSERT | UPDATE [OF column [, column]...] }]...
ON { [schema .] table | [NESTED TABLE nested_table_column OF]
[schema .] view }
[referencing_clause] [FOR EACH ROW]

Referencing clause::=
REFERENCING
 { OLD [AS] old | NEW [AS] new | PARENT [AS] parent }
 [ OLD [AS] old | NEW [AS] new | PARENT [AS] parent ].
```

Figura 2.19. La definición de un disparador en ORACLE

Al generar una sentencia de LMD, el modelo de ejecución del sistema activo de ORACLE es el siguiente:

1. Ejecutar los disparadores de tipo BEFORE con nivel de sentencia.

2. Para cada tupla a la que afecte la sentencia:
    - Ejecutar los disparadores de tipo BEFORE con nivel de tupla.
    - Ejecutar la propia sentencia.
    - Ejecutar los disparadores de tipo AFTER con nivel de tupla.
3. Ejecutar los disparadores de tipo AFTER con nivel de sentencia.

Las tablas mutantes (*mutating table*) son uno de los problemas que se producen al trabajar con los disparadores de ORACLE, una tabla mutante es una tabla que está siendo actualizada por una sentencia de actualización (UPDATE, DELETE, INSERT) o es una tabla que podría ser actualizada por los efectos de una sentencia de integridad referencial. 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.

ORACLE permite ejecutar 32 disparadores en forma de cascada en cualquier momento. Sin embargo, se puede aumentar este número modificando este valor en el parámetro de inicialización OPEN_CURSORS. ORACLE ejecuta todos los disparadores del mismo tipo antes de ejecutar los disparadores de un tipo diferente.

Cuando se compile el disparador su estado será activo. Para desactivar el estado de un disparador se usa la sentencia ALTER TRIGGER con la opción DESACTIVE.

2. IBM BD2

Un disparador se activa por ejecutar una operación de actualización (DELETE, INSERT, UPDATE) y define un conjunto de acciones que se ejecutan sobre una tabla especificada (véase la figura 2.20). En el caso del UPDATE si no se especifica la lista opcional nombre-columna, estarán implicadas todas las columnas de la tabla. Por lo tanto, se activa el disparador cuando se modifica cualquier columna de la tabla.

Los criterios que se definen al crear un disparador y que se usan para determinar cuando un disparador debe activarse, son exactamente como en el estándar y ORACLE, la tabla especificada, el evento, el tiempo de activación, y la granularidad.

NO CASCADE BEFORE, específica que la acción asociada del disparador se debe aplicar antes de aplicar a la BD los cambios producidos por la actualización real de la tabla. También especifica que la acción del disparador no provocará la activación de otros disparadores.

INSTEAD OF específica que la acción activada asociada sustituye a la acción que corresponde a la vista sujeto. Sólo está permitido un disparador INSTEAD OF para cada tipo de operación de una vista sujeto determinada.

REFERENCING especifica los variables de transición OLD y NEW, y las tablas de transición OLD_TABLE y NEW_TABLE.

```
CREATE TRIGGER<nombre-activador>
{NO CASCADE BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE OF <nombre-columna>}
ON {<nombre-tabla> | <nombre-vista>}

REFERENCING {OLD [AS] <nombre-correlación>  |
NEW [AS] <nombre-correlación>  |
OLD_TABLE [AS] <identificador> |
NEW_TABLE [AS] <identificador> }

{FOR EACH ROW | FOR EACH STATEMENT}

WHEN <condición-búsqueda>

Sentencia-procedimiento-SQL:
```

Figura 2.20. La definición de un disparador en DB2

Las siguientes normas se aplican a la cláusula REFERENCING:

- El nombre-correlación OLD y el identificador OLD_TABLE sólo pueden utilizarse si el evento del disparador es una operación DELETE o una operación UPDATE. Si la operación es DELETE, el nombre-correlación OLD captura el valor de la fila suprimida. Si la operación es UPDATE, captura el valor de la fila antes de la operación UPDATE. Lo mismo se aplica al identificador OLD_TABLE y al conjunto de filas afectadas.
- El nombre-correlación NEW y el identificador NEW_TABLE sólo pueden utilizarse si el evento del disparador es una operación INSERT o una operación UPDATE. En ambas operaciones, el valor de NEW captura el nuevo estado de la fila como lo proporciona la operación original y modificada por cualquier disparador BEFORE que se haya ejecutado hasta ese momento. Ocurre

exactamente lo mismo con el identificador NEW_TABLE y el conjunto de filas afectadas.

- Los identificadores OLD_TABLE y NEW_TABLE no se pueden definir para un disparador BEFORE.
- Los nombres-correlación no se pueden definir para un disparador FOR EACH STATEMENT.

Si los eventos para dos disparadores tienen lugar simultáneamente (por ejemplo, si tienen el mismo evento, tiempo de activación y tablas sujeto), el primer activador creado es el primero en ejecutarse. La activación de un disparador puede activar múltiples disparadores en forma cascada incluso el mismo disparador de nuevo.

3. SQL SERVER 2005

En Microsoft SQL SERVER 2005, los disparadores son una clase especial de procedimientos almacenados y se definen para ejecutarse automáticamente cuando un evento de actualización (UPDATE, INSERT, DELETE) se genera sobre una tabla o vista (véase la figura 2.21).

Los disparadores del SQL SERVER soportan sólo las temporalidades AFTER y INSTEAD OF, donde AFTER específica que el disparador sólo se activa cuando todas las operaciones especificadas en la sentencia SQL se han ejecutado correctamente. Además, todas las acciones referenciales en cascada y las comprobaciones de restricciones deben ser correctas para que este disparador se ejecute. AFTER es el valor predeterminado, si sólo se especifica la palabra clave FOR. Los disparadores AFTER no se pueden definir en las vistas. Un disparador AFTER se ejecuta sólo después de ejecutar correctamente la operación SQL, incluidas todas las acciones referenciales en cascada y las comprobaciones de restricciones asociadas con el objeto modificado o eliminado. El disparador AFTER ve los efectos de la operación así como todas las acciones UPDATE y DELETE con referencias en cascada que causan la operación.

INSTEAD OF específica que se ejecuta el disparador en vez de la operación SQL, por lo que se suplantan las acciones de las operaciones. Como máximo, se puede definir un disparador INSTEAD OF por cada operación INSERT, UPDATE o DELETE en cada tabla o vista. No obstante, en las vistas es posible definir otras vistas que tengan su propio disparador INSTEAD OF.

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [
UPDATE ] [ , ] [ DELETE ]}
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
[ { IF UPDATE ( column )
[ { AND | OR } UPDATE ( column ) ]
[ ...n ]
| IF ( COLUMNS_UPDATED ( ) )
} ]
sql_statement [ ...n ]
} }
```

Figura 2.21. La definición de un disparador en SQL SERVER

La opción WITH APPEND específica que debe agregarse un disparador nuevo a un disparador existente. La utilización de esta cláusula opcional sólo es necesaria cuando el nivel de compatibilidad es 65 o inferior entre estos dos disparadores. Si el nivel de compatibilidad es 70 o superior, no es necesaria la cláusula WITH APPEND, tal que el SQL Server substituirá cualquier disparador nuevo a los disparadores existentes del mismo tipo, incluso si los nombres de ellos son distintos.

La opción NOT FOR REPLICATION indica que el disparador no debe ejecutarse cuando un proceso de duplicación modifica la tabla involucrada en el mismo.

Las condiciones del disparador especifican los criterios adicionales que determinan si los intentos de las operaciones DELETE, INSERT o UPDATE hacen que se lleven a cabo las acciones del disparador. Los disparadores pueden incluir cualquier número y

clase de operaciones. Un disparador está diseñado para comprobar o cambiar los datos en función de una operación de actualización de datos. Las operaciones de un disparador incluyen a menudo lenguaje de control de flujo. En las cláusulas CREATE TRIGGER se utilizan unas cuantas tablas especiales: DELETED e INSERTED son tablas lógicas (conceptuales). Son de estructura similar a la tabla a la que está asociado el disparador, es decir, la tabla en que se intenta la acción del usuario, y guarda los valores antiguos o nuevos de las filas que la acción del usuario puede cambiar.

El primer y último disparador AFTER que se ejecuta en una tabla se puede especificar mediante SP_SETTRIGGERORDER. Sólo se puede especificar el primer y último disparador AFTER para cada una de las operaciones INSERT, UPDATE y DELETE de una tabla; si hay otros disparadores AFTER en la misma tabla, se ejecutan aleatoriamente. Para especificar el orden de los disparadores se usa el procedimiento SP_SETTRIGGERORDER, las opciones disponibles son:

- FIRST: Específica qué disparador es el primero que activa en un conjunto.
- LAST: Específica qué disparador es el último que activa en un conjunto.
- NONE: Especifica qué no hay ningún orden específico en que el disparador debe dispararse.

Como en el siguiente ejemplo:

```
SP_SETTRIGGERORDER   @triggername = 'MyTrigger',
@order = 'FIRST',
@stmttype = 'UPDATE'
```

SQL Server permite que se creen varios disparadores por cada evento de actualización (DELETE, INSERT o UPDATE). Por ejemplo, si se ejecuta CREATE TRIGGER FOR UPDATE para una tabla que ya tiene un disparador UPDATE, se creará un disparador de actualización adicional.

Los disparadores pueden ejecutarse hasta un máximo de 32 niveles. Si un disparador cambia una tabla en la que hay otro disparador, el segundo se activa y puede, entonces, llamar a un tercero, y así sucesivamente. Si algún disparador de la cadena

causa un bucle infinito, el nivel de ejecución se habrá sobrepasado, con lo que se cancela el disparador.

4. Comparación de aspectos

A la vista de los SGBDs analizados en las secciones anteriores podemos afirmar que existen grandes diferencias entre ellos en la forma de especificar los disparadores. El concepto de disparador no ha sido estandarizado hasta el reciente SQL3, por esta razón, los SGBDs difieren con respecto a la sintaxis, la semántica, y las capacidades. La gramática para definir un disparador en los distintos sistemas es tan diferente que ninguna especificación del disparador que se formula en un sistema puede usarse sin actualización a otros sistemas. Particularmente, las partes de acción difieren en las distintas aproximaciones porque cada sistema tiene su propio lenguaje de programación (*trigger programming language*) [Türker y Gertz, 2000].

A continuación se presentará una comparación de los cuatro sistemas gestores de BD relacionales estudiados (ORACLE, IBM BD2, MICROSOFT SQL SERVER 2005) con respecto del estándar SQL3 en términos de la especificación de los disparadores. Nuestra comparación se basa en las propiedades mostradas en la tabla 2.3.

- Evento Compuesto: sólo ORACLE 9i permite los eventos compuestos en forma muy limitada, donde se permite sólo la disyunción (OR) entre los eventos.
- Disparadores Múltiples: se permiten en todos ellos.
- Tipo de Evento: la activación de los disparadores se generan ante un evento de actualización (INSERT, DELETE, UPDATE).
- INSTEAD OF: Los sistemas ORACLE y SQL SERVER 2005 permiten activar los disparadores de tipo INSTEAD OF, en ORACLE solo sobre vistas y SQL SERVER sobre tablas y vistas.
- Tiempo de Activación: tanto SQL3, ORACLE 9i, y DB2 permiten activar dos tipos de disparadores según el tiempo de activación (BEFORE/AFTER). El sistema SQL SERVER 2005 permite sólo los disparadores de tipo AFTER.

- Granularidad: En la mayoría de los sistemas se permite la granularidad de los disparadores para cada tupla o para cada sentencia (R/S) menos en SQL SERVER 2005 el cual sólo admite la granularidad para cada tupla.
- Valores de transición: DB2 es el único sistema que soporte los dos valores de transición como SQL3. ORACLE 9i y SQL SERVER soportan solo un tipo de valor.
- Estrategia de Cascada: Existen algunas estrategias limitadas para tratar la ejecución en cascada de los disparadores en algunos sistemas. En DB2 existe la opción NO CASCADE BEFORE que especifica que la acción del disparador no generará actividad a otros disparadores. En SQL SERVER 2005 se puede usar (SP_SETTRIGGERORDER) para definir que disparador es el primero y cual es el último.
- Estrategia de Terminación: Todos los sistemas relacionales estudiados tienen una estrategia para evitar la no-terminación de la ejecución de los disparadores, donde se termina la ejecución al llegar a un nivel determinado de ejecución.

	SQL3	**ORACLE 9i**	**DB2**	**SQL SERVER 2005**
Evento Compuesto	NO	SÍ (SOLO OR)	NO	NO
Disparadores Múltiples (N)	N	N	N	N
Tipo del Evento	I, D, U	I, D, U	I, D, U	I, D, U
INSTEAD OF	NO	SÍ	NO	SÍ
Tiempo de Activación	B/A	B/A	B/A	A
Granularidad	R/S	R/S	R/S	S
OLD / NEW Tupla	SÍ	SÍ	SÍ	NO
OLD / NEW Tabla	SÍ	NO	SÍ	SÍ
Estrategia de Cascada	NO	NO	SÍ	SÍ
Estrategia de Terminación	NO	SÍ	SÍ	SÍ

Tabla 2.3. Comparación de aspectos de los disparadores

2.4 Discusión

En resumen, se puede decir que el modelo de regla (ECA) evento-condición-acción es aceptado ampliamente en los sistemas activos de BD, y proporciona la flexibilidad requerida por la mayoría de las aplicaciones. Los sistemas activos de BD relacionales y los sistemas activos de BD orientados a objetos son distintos por la propia naturaleza de los modelos de datos que soportan cada una de ellas. Sin embargo, muchos de los problemas asociados con los sistemas de BD activos son comunes a ambos sistemas. Además, la semántica del proceso de las reglas es muy variada en el mismo marco, por ejemplo, entre los SGBDs relacionales la semántica de ejecución de estas reglas es distinta.

Muchos prototipos de investigación dentro de la tecnología activa se han dedicado a la implementación de los sistemas activos y a la fiabilidad de un conjunto de reglas que aseguren las condiciones de confluencia y terminación. De los estudios anteriores se puede ver la disparidad de las aproximaciones utilizadas para solucionar los problemas asociadas a estos sistemas. En la tabla 2.4 se muestra una comparativa de los distintos enfoques que se han estudiado en este capítulo. Hemos definido algunos criterios relevantes para la motivación de nuestro trabajo, estos criterios son; el marco donde ha sido realizado el estudio, el lenguaje utilizado para definir y representar las restricciones de integridad, si en estos estudios se ha tratado la generación automática de las reglas activas, las aproximaciones utilizadas para el análisis de las reglas activas y la detección del problema de la no-terminación en la ejecución de las mismas. También, se ha tomado como criterio importante los SGBDs elegidos para implementar las distintas propuestas, y la manera de representar el comportamiento activo de las reglas.

Criterios	**[Ceri et al., 1994]**	**[Ceri y Fraternalli, 1997]**	**[Türker y Gertz, 2000]**	**[Decker, 2002]**
El marco	BDOO	BDOO	BDR	BDR
El Lenguaje utilizado para definir RI	Starburst	Chimera	SQL3	SQL
Generación automática de reglas	Herramientas propias	Herramientas propias		
Análisis de Reglas activas	Triggering Hypergraph	Triggering, Activation Graphs		
Implementación en los SGBDs comerciales		Extensión a comercial		
Visualizar la ejecución de reglas	Herramientas propias	Herramientas propias		
Modelar el comportamiento de las reglas				
Mejorar el rendimiento de BD				

Tabla 2.4. Comparación de los estudios anteriores más relevantes

Según los criterios definidos se puede ver claramente que las aproximaciones en el marco de las BDs relacionales, en general, cubren unos aspectos muy reducidos de la investigación aunque este marco se considera el más extendido para la implementación. Según el análisis realizado en la tabla 2.4 podemos concluir que el marco de las BDs relacionales necesita más esfuerzos e investigación dentro de una aplicación eficiente y eficaz de las reglas activas. También, existe una escasez en herramientas que puedan servir como ayuda a los desarrolladores de BDs relacionales para la implementación de reglas activas, de esta forma, se conseguirá una motivación para su uso y se les facilitaría el desarrollo de las aplicaciones. El estudio más relevante es la metodología IDEA [Ceri y Fraternalli, 1997] según los criterios. Aunque esta metodología tiene una extensión a los sistemas comerciales, cabe destacar que las restricciones de integridad se especifican utilizando el lenguaje de *Chimera* para BDs orientados a objetos. Por eso, a la hora de transformar estas

reglas a disparadores de sistemas relacionales se exige agregar estructuras de datos auxiliares, disparadores y procedimientos almacenados y esto significa en algunos casos un impacto sobre el rendimiento y una transformación más difícil de entender e implementar.

Existen distintas propuestas en el marco de las BDs relacionales pero la mayoría de estas propuestas han tratado una parte de la solución de las restricciones de integridad utilizando la tecnología activa. Por ejemplo, el trabajo de [Türker y Gertz, 2000] lo consideramos como una referencia muy importante para el nuestro, por su aportación en el tema del estándar SQL3 y las reglas de la transformación de una restricción de integridad a un disparador pero, en dicho trabajo no se han demostrado la terminación de estas reglas y tampoco se ha implementado la propuesta para verificar y validar estas reglas. En la propuesta de [Decker, 2002] se han derivado los disparadores del estándar SQL3 de las especificaciones de las restricciones de integridad, sin incluir las características dinámicas de estos disparadores y se ha elegido el rechazo como una reacción general de los disparadores. Lo que significa que la BD se comporta de una manera pesimista ante cualquier violación sin la posibilidad de reparar daños que puedan producirse cuando se viola la integridad de los datos.

El estudio y la comparación del control de las restricciones a través de dos enfoques, BDs activas y BDs relacionales, se puede resumir en que las BDs activas son muy potentes para este control pero no son realistas con los SGBDs que se utilizan en el mercado. Por otro lado, las BDs relacionales son las más extendidas comercialmente hablando pero en la mayoría de los casos los desarrolladores utilizan el soporte activo que los SGBDs proporcionan para realizar estudios de auditoría exclusivamente.

En este trabajo de tesis de doctoral, por lo tanto, se intenta aunar las ventajas de la tecnología activa dentro de las BDs relacionales. Las principales aportaciones se dirigen a cubrir todos los criterios especificados en la tabla 2.4. La propuesta se basa en transformar las restricciones de integridad de un esquema conceptual a disparadores del estándar SQL3 aplicando un conjunto de reglas fundamentadas en la

teoría del cálculo relacional. Estos los disparadores serán transformados a un script ejecutable con las características específicas del SGBD elegido. A partir de aquí, nos ocupamos de la fase de implementación de la propuesta y resolver dos problemas asociados con cada una de las tecnologías utilizadas en este trabajo de tesis. El primer problema es generado por la utilización de la tecnología activa y es la no-terminación del conjunto de reglas especificado y el segundo viene dado por el SGBD ORACLE utilizado, las tablas mutantes.

Por último, hemos implementado nuestra propuesta en una herramienta CASE de desarrollo de BD (Rational Rose). La implementación consiste de dos módulos: el primer módulo proporciona una automatización en la transformación de un esquema conceptual en un esquema relacional sin que existan pérdidas de semántica y el segundo módulo, juega un papel muy importante no solo por verificar la ejecución de las reglas, sino que además nos proporciona un interfaz gráfico para mostrar las interacciones entre las reglas y las acciones de integridad referencial.

Capítulo 3

3. Propuesta para la Conservación de Restricciones de Integridad en BD

3.1 Introducción

Cuando se aplica una metodología de desarrollo de BD nos encontramos que los modelos conceptuales que se emplean son cada vez más ricos semánticamente hablando y esto implica que son capaces de recoger con mayor precisión las especificaciones del dominio. El problema surge cuando se quiere mantener esta semántica en la siguiente fase de la metodología, la fase lógica, donde la mayoría de las metodologías coinciden en la utilización del modelo relacional para transformar el esquema conceptual en un esquema cercano a la implementación. Para ello se aplica un conjunto de reglas de transformación. Estas reglas son, en general, muy básicas pues solo especifican la transformación de los elementos más simples y sencillos del modelo conceptual, lo que implica una pérdida de semántica significativa que lleva al diseñador a controlar y comprobar las restricciones de integridad fuera de la BD.

Se puede definir una restricción de integridad como una regla activa (evento, condición, y acción). El *evento* es una operación de actualización como inserción, borrado, o modificación cuya ejecución ha de dar lugar a la comprobación del cumplimiento de la restricción. La *condición* que debe cumplirse, la cual, en general, es una proposición lógica, definida sobre uno o varios elementos del esquema, que

puede tomar uno de los valores de verdad (cierto o falso). La *acción* que debe llevarse a cabo dependiendo del resultado de evaluar la condición.

El principal objetivo de esta tesis doctoral, enmarcada en los sistemas activos de BD relacionales, es definir y representar las restricciones de integridad como una regla activa conceptual utilizando el lenguaje del estándar SQL3. Además, se propone una metodología para la conservación de las restricciones de integridad en una BD relacional a la largo de todas las fases de desarrollo de la misma; se especifican cuáles son las reglas para transformar una restricción de integridad a un disparador del estándar SQL3 y cuáles son los procesos necesarios para transformar estos disparadores a un script ejecutable dentro del SGBD seleccionado. Según nuestra propuesta es insuficiente llegar a producir disparadores ejecutables sin la comprobación del comportamiento activo de estos; por ello, se ha desarrollado una herramienta que puede generar disparadores automáticamente, aplicando la metodología, comprobando y verificando su comportamiento y sus interacciones con las acciones de integridad referencial. Otra de las aportaciones de este trabajo de tesis doctoral es modelar el comportamiento de los disparadores y las interacciones entre ellos utilizando el diagrama de secuencias del UML [Rumbaugh, 2005] [Booch, 2005]. En esta aportación se define la secuencia de ejecución de los disparadores utilizando constructores del UML y se mandan mensajes de aviso a los usuarios cuando se detecten estados de no terminación. Este desarrollo se ha llevado a cabo empleando y la herramienta CASE Rational Rose [Rational, 2003].

A continuación, se presenta una clasificación y definición de las restricciones de integridad en un modelo de datos relacional y se analiza la pérdida de semántica al transformar un esquema conceptual a un esquema lógico. Por último, se detalla la metodología propuesta para solucionar el problema de la pérdida de semántica.

3.2 Manejando Restricciones de Integridad en una Metodología

La realización de un buen análisis, diseño e implementación de una BD deberá estar apoyada en una metodología de desarrollo que guíe al diseñador en el proceso para la obtención de un buen diseño que refleje el mundo real. Durante la aplicación de la metodología se puede producir pérdida de semántica y generalmente, son las restricciones de integridad las más afectadas en este proceso. Las restricciones de integridad son fundamentales en cualquier metodología y son responsables de recoger la mayor semántica posible del UD.

Así pues, en este apartado se presentará en la sección 3.1.1 la definición y la clasificación de las restricciones de integridad dentro del modelo de datos relacional. En la sección 3.1.2 se presenta un estudio sobre la pérdida de semántica producida al transformar un esquema conceptual a un esquema relacional.

3.2.1 Definición de Restricción de Integridad Dentro de un Modelo de Datos

Los modelos conceptuales poseen la característica de tener un alto nivel de abstracción y nos servirán para validar la modelización del UD con los expertos del dominio. Los modelos conceptuales deben ser intuitivos y ofrecer distintos mecanismos para recoger toda la semántica del problema. Un modelo de datos es un conjunto de conceptos, reglas y convenciones que permiten describir y manipular los datos de un cierto mundo real que se desea almacenar en la BD [Miguel y Piattini, 1999]. Las ventajas de utilizar un modelo de datos son las siguientes:

- A partir de los modelos de datos se definen los lenguajes de datos. Por ejemplo, el lenguaje SQL se construye añadiendo al modelo relacional una sintaxis.

- Permiten realizar una formalización de las estructuras permitidas y las restricciones con el fin de representar los datos de un sistema de información.
- Son una de las herramientas básicas de una metodología de desarrollo de BD y facilitan la evaluación del impacto en los cambios de un sistema de información.

Todo modelo de datos se caracteriza por una parte estática, estructura de los datos con sus restricciones, y una parte dinámica, que se compone de un conjunto de operadores que se definen sobre las estructuras de los datos. Los lenguajes de definición de datos se corresponden con la parte estática de un modelo más una sintaxis y los lenguajes de manipulación con la parte dinámica más la sintaxis. La estática de un modelo de datos está compuesta por dos partes, como se muestra en la figura 3.1 [Cuadra, 2003]. La primera parte controla de los elementos permitidos que no son los mismos para todos los modelos de datos (varían especialmente en terminología), pero en general, son objetos, asociaciones entre objetos, propiedades o características de los objetos o asociaciones, y dominios que son conjuntos nominados de valores homogéneos sobre los que se definen las propiedades. A estos elementos permitidos se les podrán aplicar aquellas abstracciones reconocidas por el modelo. La representación de estos elementos depende de cada modelo de datos, pudiendo hacerse en forma de grafos (modelo ER, o en el Codasy) o de tablas (modelo relacional). La segunda parte la forman los elementos no permitidos o restricciones. No todos los valores, cambio de valor o estructuras están permitidos en el mundo real; por ejemplo, un niño de tres años no puede estar casado, ni una persona puede pasar directamente de soltera a viuda, etc. Además, cada modelo de datos impone por sí mismo limitaciones a las estructuras que admite. Estás limitaciones se dividen en dos tipos; restricciones inherentes y restricciones de integridad o semánticas.

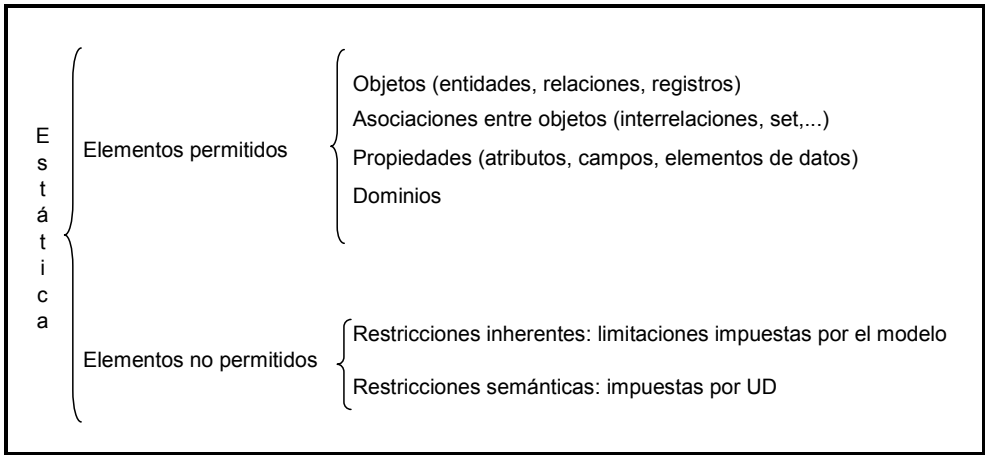


Figura 3.1. Parte estática de un modelo de datos

En el modelo relacional, el marco de investigación de este trabajo, existen distintas clasificaciones de las restricciones de integridad [Mannila y Räihä, 1992]. Una de estas clasificaciones considera que las restricciones de integridad se dividen en restricciones estáticas y dinámicas, también llamadas restricciones de estado y de transición [Date, 1990] [Elmasri y Navathe, 2004]. Las restricciones estáticas tienen que ver con los estados válidos que puede tomar la BD y suelen ser condiciones establecidas sobre atributos, tuplas, interrelaciones. Las restricciones dinámicas tienen que ver con las transiciones válidas entre los estados posibles de la BD. Las restricciones dinámicas son propiedades del UD que restringen la posible evolución en el tiempo de la BD. Es decir, dado el valor actual de la BD, cuáles son los posibles valores que puede tomar a continuación; por ejemplo, los precios de los artículos no se pueden reducir.

Además, en el modelo relacional se distingue entre las restricciones de integridad según el elemento sobre el que se definen. Según el estándar SQL92 [Date y Darwen, 1997], los tipos de restricciones de integridad definidas en el esquema de la BD son: el dominio, la integridad referencial, la clave primaria, el check, la aserción, y otras restricciones, como se muestran en la figura 3.2.

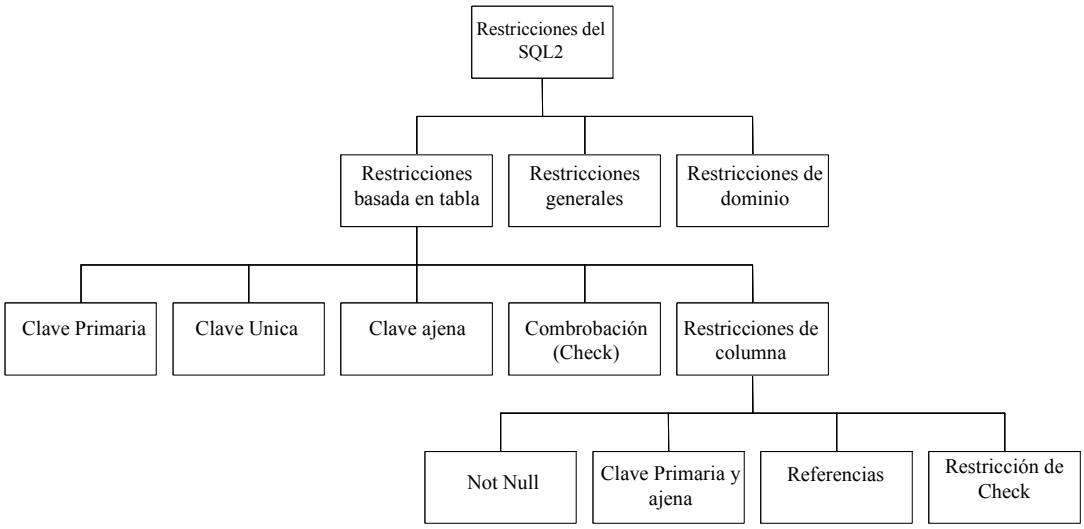


Figura 3.2. La clasificación de las restricciones de integridad [Date y Darwen, 1997]

En el estándar SQL3 [Melton y Simon, 2002] se han incluido todas las restricciones de integridad definidas en el SQL92 más los disparadores, los cuales se diferencian de otros tipos de restricciones en los siguientes aspectos: el disparador se activa cuando haya ocurrido un cierto evento; inmediatamente después de la ocurrencia del evento este disparador verifica la condición y una vez que se satisface la condición, se ejecutan las acciones asociadas al mismo por el SGBD.

Se pueden especificar de una forma estandarizada utilizando el SQL3 todas las restricciones de integridad de una BD como reglas activas, aunque el paso inverso siempre no es cierto; puesto que puede haber reglas ECA que responden a una política específica de una compañía que no incluye ninguna restricción; por ejemplo, se pueden utilizar reglas ECA para controlar eventos de cambio de temperatura. Puesto que las restricciones de integridad son reglas ECA, los tres componentes de la regla ECA (evento, condición y acción) son, lógicamente, esenciales en la definición de cualquier restricción.

Si tenemos en cuenta los tres componentes fundamentales (evento, condición y acción) en la definición de las restricciones de integridad puede establecer una clasificación de estas restricciones. Así, el posible distinguir entre las restricciones que son hechos que debe controlarse en el SGBD y las que se apoyen en los

programas de aplicación [Geppert y Dittrich, 1995]. Nosotros consideramos una buena idea comenzar la clasificación distinguiendo entre las restricciones impuestas por el modelo y las usadas por los diseñadores para recoger la información incluida en el UD (véase la figura 3.3). Así, dependiendo de fuente de la imposición, se puede distinguir dos tipos de restricciones:

A. Restricciones inherentes impuestas por el modelo de datos. Estas restricciones no necesitan ser definidas por el usuario, por lo tanto, son obligatorias y se definen dentro del modelo. Se activan en la fase de definición del esquema, cuando haya una posibilidad de violación por cualquier causa la operación se rechaza. Este tipo de restricciones introduce rigidez en el modelo; por ejemplo, la obligatoriedad de la clave primaria en el modelo relacional y responde a la idea de que no puede haber dos objetos idénticos en el mundo real.

B. Las restricciones de integridad (también se llaman restricciones semánticas o de usuario) son especificadas por los diseñadores y permiten capturar la semántica del UD. Se utilizan para comprobar la exactitud de los datos almacenados en la BD y se activan cada vez que la BD vaya a actualizarse. Cualquier operación que no respete la restricción puede ser rechazada, o activar otros mecanismos para controlar la violación. Este tipo de restricción ayuda a capturar la semántica de los datos y alcanzar la consistencia de la BD. Ejemplos de este tipo de restricción en el modelo relacional son: la integridad referencial, la especificación de si un atributo puede o no tomar valores nulos, cardinalidades en el modelo ER, etc. Dentro de este tipo, es posible distinguir entre dos tipos de restricciones; restricciones externas y restricciones internas.

B.1. Las restricciones externas se especifican en los programas de aplicación y, por lo tanto, no se definen en el esquema de la BD y no se almacenan en la metabase. El SGBD no puede verificar si estas restricciones son consistentes entre si, pues el SGBD no sabe nada sobre ellas. Proporcionan una mayor flexibilidad aunque representan una carga considerable en términos de programación y mantenimiento por estar incluidas en cada programa que modifica los elementos que las contienen. Por otra parte, su presencia en los

programas de aplicación contamina éstos con las características asociadas más a la semántica de los datos que al proceso realizado por estos programas, con las dificultades de mantenimiento consecuentes si la semántica cambia. Existen dos tipos de restricciones externas:

B.1.1. *Lenguaje de propósito general*: esta categoría abarca todas las restricciones externas especificadas en un lenguaje de programación. Este tipo de lenguaje no proporciona ninguna característica para la especificación de las restricciones y, como no se adaptan al uso de la BD requieren mecanismos adicionales para poder utilizar los lenguajes normales para hacer consultas sobre la BD.

B.1.2. *Lenguaje del SGBD*, este grupo incluye las restricciones externas que tienen una cierta característica, herramienta o lenguaje proporcionado por el fabricante del SGBD. Por ejemplo, en la herramienta Form Builder[1], es posible especificar la opción ON UPDATE CASCADE para la integridad referencial aunque el SGBD en sí mismo no la considera.

B.2. Las restricciones internas se especifican en el esquema y se almacenan en la metabase. Ninguna operación de actualización puede violarlas pues el SGBD por sí mismo está encargado de su control. La programación y el mantenimiento son más simples que en las restricciones externas pues están en un almacén centralizado único. La desventaja principal de éstas es que la funcionalidad que proporcionan depende de las características de cada SGBD específico. Entre las restricciones internas se puede distinguir, dependiendo del tipo de la acción que realizan: en las que se puede especificar la acción (acción general) y en la que solo hay una acción (acción específica), generalmente de rechazo. En el segundo tipo, una condición

[1] © Oracle Corporation.

tiene que expresarse explícitamente por medio de una consulta lógica como condición general, o bien implícitamente como condición específica.

B.2.1. En las restricciones de *acción general* es obligatorio indicar la acción (así como la condición). Son procedimentales (por lo menos en parte, puesto que la acción se especifica siempre con un procedimiento) y representan siempre una carga de programación. Es muy difícil (prácticamente imposible en la mayoría de los casos) que el SGBD verifique su consistencia. Este tipo de restricciones se pueden dividir en dos subtipos dependiendo de los mecanismos usados para controlar la restricción. El primer grupo requiere la invocación explícita de la restricción, mientras que en el segundo, la restricción se activa de manera autónoma cuando la actualización ocurre y dependiendo de la verificación de la condición:

B.2.1.1. *Procedimientos almacenados*, pueden ser tan complejos como la semántica del mundo real requiera tanto en la condición como en la acción.

B.2.1.2. *Los disparadores*, combinan las restricciones declarativas en la acción y las restricciones procedimentales en la condición. En términos de la acción, pueden ser tan complejos como la semántica del mundo real requiera (dependerá de las características proporcionadas por el lenguaje del SGBD), y la condición tan compleja como una consulta lógica. La evaluación de la condición a verdadero ejecuta la acción.

B.2.2. En las restricciones de *acción específica*, las acciones están implícitas en la restricción, así que no es necesario definirla (se predetermina). Se definen de una manera declarativa. Cuando se viola la condición se debe aplicar una acción. Pueden definirse usando un lenguaje estándar de definición de restricciones en la BD. El SGBD puede comprobarlas por si mismo. No representan una carga de programación, solamente una carga en la definición. Este tipo de

restricciones se puede dividir en dos subtipos dependiendo de si la condición tiene que ser definida explícitamente o implícitamente en la restricción.

B.2.2.1. Restricciones de *condición general* (acción de rechazo). La acción no se especifica pues es siempre el rechazo. Es obligatorio declarar la condición por medio de una consulta lógica permitiendo condiciones complejas arbitrarias. Estas restricciones se dividen en:

B.2.2.1.1. Restricciones de verificación (CHECK) se incluyen en la definición del elemento afectado por la restricción. Aunque pueden afectar a más de un elemento, en tales casos pueden ser más complejos. Pueden o no pueden tener un nombre.

B.2.2.1.2. Las restricciones de aserción (ASSERTION), se definen de una forma independiente dentro del esquema y pueden afectar a más de un elemento del mismo. Se identifican por un nombre.

B.2.2.2. Las restricciones de *condición específica*, vienen definidas en el modelo (o por un estándar como SQL). No se especifica ninguno de los componentes de la restricción (ninguna operación, ninguna condición, ninguna acción). No son muy flexibles pues permiten solamente las opciones consideradas en la restricción (por ejemplo, las opciones del borrado y la actualización en la integridad referencial).

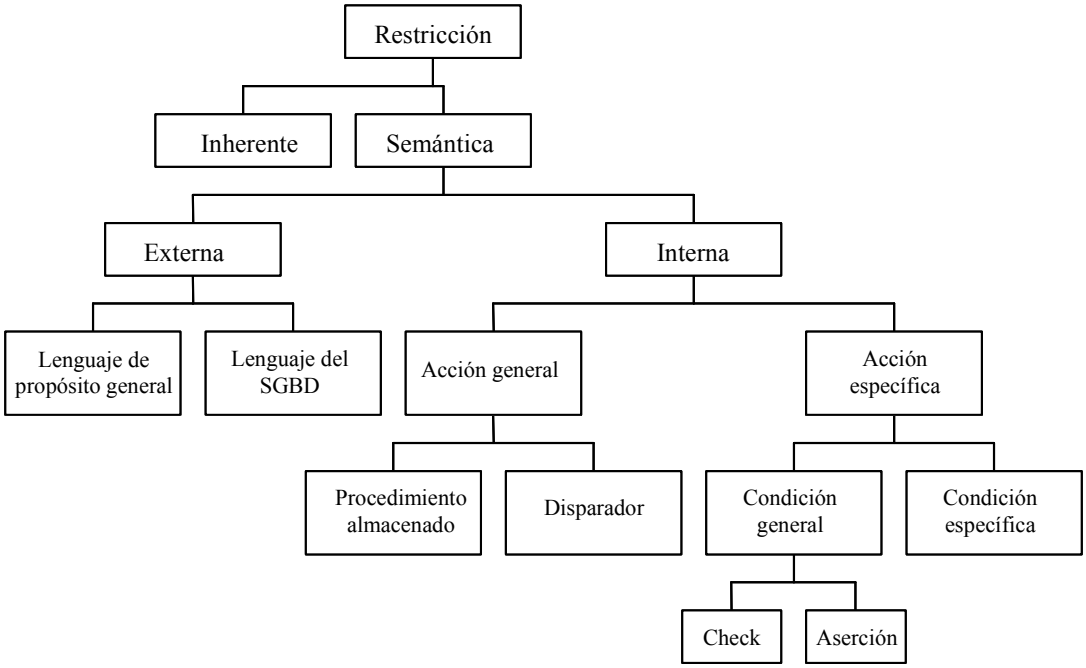


Figura 3.3. Una clasificación de las restricciones de integridad

En general, el SGBD proporciona dos maneras para conservar estas restricciones de integridad; mecanismos declarativos y mecanismos procedimentales que incluyen los disparadores. Los mecanismos declarativos proporcionan una base fundamental y estrategia tradicional para reforzar las reglas básicas de negocio. Para ello, los sistemas gestores ofrecen cláusulas declarativas que expresan restricciones sobre los valores permitidos de atributos (como las restricciones de dominio y claves únicas), y para expresar un tipo sencillo y frecuentemente utilizado en las dependencias funcionales (como las acciones de integridad referencial). Sin embargo, estas cláusulas no son capaces de expresar algunas restricciones, como por ejemplo, "el sueldo total de los empleados que trabajan en un departamento no debe exceder el presupuesto del departamento al que pertenecen".

Por otro lado, los documentos de SQL indican generalmente que los disparadores, los procedimientos almacenados y los programas incluidos en las aplicaciones tienen tendencia a consumir menos recursos que las cláusulas tradicionales CHECK o ASSERTION [Decker, 2002]. Aunque algunos SGBDs (como OCELOT SQL [Ocelot, 2003]) contienen la cláusula ASSERTION, en este trabajo no se han

considerado estos sistemas pues nos interesa contemplar otro tipo de acciones aparte del rechazo cuando no se cumple la restricción. También la utilización de la misma puede producir un impacto sobre el rendimiento de SGBD, y la mayoría de los productos comerciales no la soportan [Türker y Gertz, 2000].

Los mecanismos procedimentales o disparadores pueden proporcionar una base fundamental para conseguir un nivel muy alto de independencia de conocimiento de la BD [Paton, 1998]. Se pueden utilizar estos mecanismos para verificar cualquier restricción de integridad en general, y en particular las restricciones de integridad de tipo *condiciones generales* y *acciones generales* o *condiciones generales* y *acciones específicas* que los mecanismos declarativos no son capaces de verificar. Aunque todos los tipos de restricciones de integridad pueden ser recogidos utilizando disparadores, por razones de rendimiento es recomendable utilizar mecanismos declarativos siempre que sea posible [Türker y Gertz, 2000]. Así, se puede concluir que cada tipo de los mecanismos anteriores tiene sus ventajas y sus desventajas, y están disponibles en muchos sistemas comerciales. Además, no es común encontrar aplicaciones que hayan sido implementadas empleando solo un tipo de mecanismo. Los SGBDs tienen sus capacidades y sus limitaciones, y los diseñadores de BD son responsables de elegir y emplear estas capacidades para recoger la mayor parte de la semántica de la BD y que, además, sea eficiente.

3.2.2 Análisis de la Pérdida de Semántica

El modelo Entidad-Interrelación (ER) propuesto por [Chen, 1976] es el de mayor aceptación en el área de desarrollo de BD, por la simplicidad y claridad de sus conceptos estructurales que permiten a los usuarios modelar los objetos del mundo real [Fahrner y Vossen, 1995]. Pero este modelo, tal como fue propuesto, adolecía de cierta falta de semántica en sus constructores (entidades, interrelaciones y atributos). Por esta razón, son muchos los trabajos que se proponen con la finalidad de extender el modelo dotándolo de mayores capacidades semánticas. Aunque la definición de los constructores del modelo ER son sencillas, los experimentos descritos en [Batra y Zanakis, 1994] [Batra y Antony, 1994], demuestran que aunque los diseñadores inexpertos tienen poca o ninguna dificultad para representar entidades y atributos, no ocurre lo mismo con la identificación de las interrelaciones. Por ejemplo, si hay dos entidades, existen cuatro posibilidades para el tipo de correspondencia N:M, 1:N, N:1 y 1:1, lo que implica que según aumenta el grado de la interrelación más complicada es su detección y más son los errores cometidos por el usuario [Cuadra et al., 2002].

Es de sobra conocido que en la transformación de interrelaciones al modelo relacional se pierde semántica. Esto es debido a que se aplican unas reglas básicas que especifican la transformación de los elementos más simples y sencillos del modelo conceptual. En este apartado se propone un enfoque sistemático para conservar la semántica de estas interrelaciones cuando se transforman al modelo relacional. Por ello, es necesario estudiar cómo la inserción, el borrado, y la modificación pueden afectar a los objetos de una interrelación [Díaz, 1996].

A continuación, se van a definir las restricciones de cardinalidad, [Storey, 1993] una de las propiedades principales de una interrelación y que por lo general, no suele considerarse cuando se transforman interrelaciones al modelo relacional.

La restricción de cardinalidad nos ayuda a describir la interrelación entre dos o más entidades y es fundamental para recoger la semántica del mundo real [Jones, Song,

1996]. Sin embargo, cuando se realiza la transformación de esquemas ER a esquemas relacionales, cubriendo las fases de la metodología de desarrollo de BD, se producen pérdidas de semántica ya que la mayoría de las propuestas solo consideran las cardinalidades máximas o de tipo de correspondencia para realizar esta transformación.

El objetivo de una restricción de cardinalidad es limitar el número de ocurrencias de las entidades asociadas en una interrelación. Las cardinalidades mínima y máxima se definen en [Chen, 1976] como el número mínimo y máximo de ejemplares de un tipo de entidad asociada con un ejemplar de la otra entidad en el tipo de la interrelación, que puede ser 1 ó N. En [Rochfeld y Tardieu, 1983] se definen las cardinalidades mínimas y máximas como el mínimo y máximo número de veces que un ejemplar de un objeto participa en los ejemplares de la interrelación. En [De Miguel et al., 1999] se definen las cardinalidades mínimas y máximas de los tipos de entidad que participan en una interrelación binaria como el número mínimo y máximo de ejemplares de un tipo de entidad que puede relacionarse con un único ejemplar del otro. En [Cuadra, 2003] una restricción de cardinalidad limita la combinación de instancias de los elementos que participan en una interrelación (familia ER), asociación (familia OO) o predicado (familia *Object-Role* OR). Se puede distinguir cardinalidades mínimas que imponen un límite inferior a esta combinación y las cardinalidades máximas que establecen un límite superior. Gráficamente, las restricciones de cardinalidades se representan por una etiqueta, (0,1), (1,1), (0,N), (1,N), etc., situada en la línea que conecta el tipo de entidad con el rombo que representa el tipo de interrelación.

Aunque la restricción de cardinalidad es una de las más importantes restricciones que pueden establecerse en un esquema conceptual, sin embargo, la definición y la representación en el modelo conceptual de esta restricción admite diversas variantes. La representación de la restricción de cardinalidad de [Rochfeld y Tardieu, 1983], especifica el número mínimo y máximo de instancias de un tipo de entidad que pueden participar en las instancias de una interrelación. Por ejemplo, según esta representación la cardinalidad mínima de la entidad A (0, N) mostrada en la figura

3.4 indica que la participación del tipo de entidad A dentro del tipo de interrelación es opcional, es decir, no toda instancia de la entidad A interviene en la interrelación. Por el contrario, la cardinalidad mínima de la entidad B (1, N) obliga a que cada instancia del tipo de entidad B intervenga en 1 o i instancias de la interrelación. Las cardinalidades máximas pondrán el límite superior a esta participación.

La representación de [Chen, 1976] fija una instancia de una entidad y cuenta el número mínimo y máximo de veces que participa con distintas instancias de la otra entidad que comparte la misma interrelación. En el mismo ejemplo de la figura 3.4, y según esta representación, la cardinalidad mínima del tipo de entidad A (1, N) obliga a que cada instancia del tipo de entidad A intervenga en *1* o *i* instancias de la interrelación, mientras la cardinalidad mínima de la entidad B (0, N) dentro del tipo de interrelación es opcional.

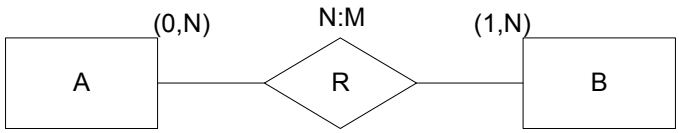


Figura 3.4. Cardinalidades mínimas y máximas para una interrelación binaria

En este trabajo se ha estudiado la transformación de interrelaciones con el fin de que pueda implementarse los mecanismos que controlen dichas interrelaciones en una herramienta CASE. En el siguiente apartado se describe la transformación de interrelaciones binarias según la cardinalidad asociada del modelo ER al modelo relacional analizando la pérdida semántica. Para ello, se va a utilizar la definición propuesta por [Chen, 1976] para analizar el problema.

Transformación de interrelaciones binarias

La semántica contemplada por la definición de la cardinalidad asociada a una entidad en una interrelación determina el tipo de la participación opcional u obligatoria de las instancias de esa entidad en la interrelación. Así pues, se fija una instancia de una

entidad y se cuenta el número mínimo y máximo de veces que participa con distintas instancias de la otra entidad en la interrelación.

Una interrelación binaria de tipo N:M se transforma a una relación/tabla que tiene como clave primaria la composición de las claves primarias de las entidades asociadas; si la interrelación contiene atributos, se incluyen como atributos en la nueva relación. La interrelación binaria 1:N se transforma propagando la clave de la entidad referenciada en la relación de la entidad que referencia. La transformación de la interrelación 1:1 puede considerarse tanto un caso particular de las N:M como de las 1:N [Teorey, 1999].

A continuación se estudiarán las cardinalidades mínima 0 ó 1 y la máxima 1 o N. La tabla 3.1 muestra las distintas combinaciones de cardinalidades asociadas a estas interrelaciones binarias que han sido estudiadas en este trabajo de tesis.

N:M		**1:N**		**1:1**	
A	**B**	**A**	**B**	**A**	**B**
1,N	1,N	1,N	1,1	0,1	0,1
0,N	1,N	1,N	0,1	0,1	1,1
0,N	0,N	0,N	1,1	1,1	1,1
		0,N	0,1		

Tabla 3.1. Los distintos casos de cardinalidades asociadas a una interrelación binaria

En la tabla 3.2 se presenta el análisis de la pérdida de semántica en la transformación de la interrelación binaria de tipo N:M. Por ejemplo, el caso (1) mostrado en la tabla 3.2 es el más crítico; se transforman las dos entidades a dos tablas, *A* (ID_A, ….) y B (ID_B, ….). La tabla *R* (ID_A, ID_B) representa la transformación de la interrelación y se especifican las opciones de acción de integridad referencial como el borrado en cascada (DC) y la actualización en cascada (UC).

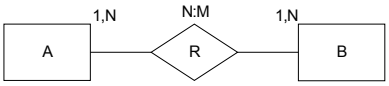
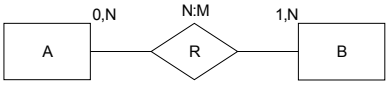
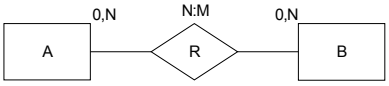
Casos	Transformación	Pérdidas de semántica
Caso (1) 	A (<u>ID_A</u>, ….) B (<u>ID_B</u>, ….) R (<u>ID_A, ID_B</u>) DC[2], UC[3] A_en_R: Obligatorio B_en_R: Obligatorio	INSERT en A ó B DELETE de A ó B ó R UPDATE en R (ID_A ó ID_B)
Caso (2) 	A (<u>ID_A</u>, ….) B (<u>ID_B</u>, ….) R (<u>ID_A, ID_B</u>) DC, UC A_en_R: Obligatorio B_en_R: Opcional	INSERT en A DELETE de B ó R UPDATE en R (ID_A)
Caso (3) 	A (<u>ID_A</u>, ….) B (<u>ID_B</u>, ….) R (<u>ID_A, ID_B</u>) DC, UC A_en_R: Opcional B_en_R: Opcional	No hay

Tabla 3.2. La pérdida de semántica en la transformación de una interrelación binaria de tipo N:M

Las restricciones de cardinalidad del caso (1) especifican que existe una participación obligatoria de *A* en *R* (A_en_R: Obligatorio), es decir, un ejemplar de *A* puede relacionarse con uno o varios ejemplares de *B*. También, existe una participación obligatoria de *B* en *R* (B_en_R: Obligatorio). En esta definición existen problemas con respecto a la cardinalidad mínima de *A*, y con respecto a la cardinalidad mínima de *B*. Por lo tanto, se debe controlar que la BD no quede inconsistente cuando se produzcan inserciones de nuevas tuplas en *A* ó *B* (INSERT *A* ó *B*), cuando se borren tuplas de las relaciones *A* ó *B* ó *R* (DELETE *A* ó *B* ó *R*) y cuando se modifiquen en *R*

[2] DC: ON DELETE CASCADE

[3] UC: ON UPDATE CASCADE

cualquiera de las claves ajenas, UPDATE en *R* (ID_A ó ID_B). El caso (3) de la tabla 3.2 es más sencillo porque las dos participaciones son opcionales y no es necesario utilizar ningún mecanismo para conservar estas cardinalidades.

En la tabla 3.3, se presenta el análisis de la pérdida de semántica en la transformación de una interrelación binaria de tipo 1:N. En el caso (1) de la tabla 3.3 es el más crítico, pues existe una participación obligatoria de *A* en *B* (A_en_B: Obligatorio y B_en_A: Obligatorio), es decir, un ejemplar de *A* puede relacionarse con solo un ejemplar de *B*, mientras un ejemplar de *B* puede relacionarse con uno o varios ejemplares de *A*. Existen problemas con respecto a la cardinalidad mínima de *A*, y con respecto a la cardinalidad mínima de *B*. Por lo tanto, se debe controlar que la BD no quede inconsistente cuando se produzcan inserciones de nuevas tuplas (INSERT en *A* ó *B*), cuando se borren tuplas de la relación (DELETE de *B*) y cuando se modifique en *B* la clave ajena que referencia a *A*, UPDATE en *B* (ID_A). En esta transformación el caso (4) es el más sencillo porque las dos participaciones son opcionales y no es necesario ningún mecanismo para conservar estas cardinalidades.

Pueden existir casos en los que la cardinalidad máxima N sea conocida o la cardinalidad mínima mayor de uno. En esos casos, se debe controlar que no se introduzcan más tuplas de las indicadas en la cardinalidad máxima o no se borren menos tuplas de las indicadas en la cardinalidad mínima. No existe problema si la cardinalidad máxima es uno ya que, se controla con la clave primaria. Sin embargo, en los otros casos si se precisará el empleo de disparadores. La tabla 3.4 muestra los casos más críticos de los dos tipos de interrelaciones binarias N:M y 1:N. En estos casos, además de controlar la semántica al insertar en los tipos de entidad A y B hay que controlar la inserción en el tipo de la interrelación R con respecto al tipo N:M.

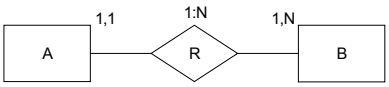
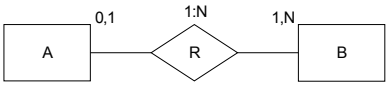
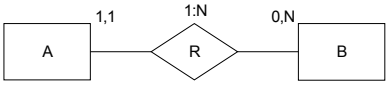
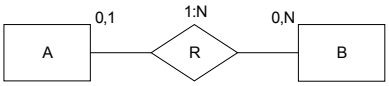
Casos	Transformación	Pérdidas de semántica
Caso (1) 	A (ID_A, ….) B (ID_B,…, ID_A(NN[4])) DC, UD A_en_B: Obligatorio B_en_A: Obligatorio	INSERT en A DELETE de B UPDATE en A (ID_B)
Caso (2) 	A (ID_A, ….) B (ID_B,…, ID_A) DSN[5], UC A_en_B: Opcional B_en_A: Obligatorio	INSERT en A DELETE de B UPDATE en B (ID_A)
Caso (3) 	A (ID_A, ….) B (ID_B,…, ID_A(NN)) DC, UD A_en_B: Obligatorio B_en_A: Opcional	No hay
Caso (4) 	A (ID_A, ….) B (ID_B,…, ID_A) DSN, UD A_en_B: Opcional B_en_A: Opcional	No hay

Tabla 3.3. La pérdida de semántica en la transformación de interrelación binaria de tipo 1:N

[4] NN: NOT NULL para controlar la obligatoriedad de B_en_A cuando se inserta en la tabla correspondiente

[5] NSN: ON DELETE SET NULL

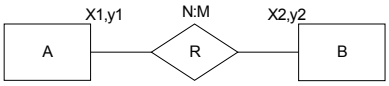
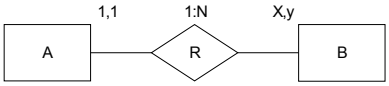
Casos	Transformación	Pérdidas de semántica
Caso (1)  	A (ID_A, ….) B (ID_B, ….) R (ID_A, ID_B) DC, UD A_en_R: Obligatorio B_en_R: Obligatorio	INSERT en A ó B ó R DELETE de A ó B ó R UPDATE en R (ID_A o ID_B)
Caso (2)  	A (ID_A, ….) B (ID_B, ….) DC, UD A_en_R: Obligatorio B_en_R: Obligatorio	INSERT en A ó B DELETE de B UPDATE en B (ID_A)

Tabla 3.4. La pérdida de semántica en la transformación de interrelaciones con cardinalidad máxima conocida y cardinalidad mínima mayor que uno

Generalización

Una generalización o jerarquía (correspondiente al concepto ES-UN) consiste en dos o más tipos de entidades que comparten varios atributos y/o interrelaciones, de donde se deduce la existencia de un tipo de entidad de nivel superior que contiene los atributos y los tipos de interrelación comunes a todos los subtipos [De Miguel et al., 1999]. Para representar una jerarquía en el modelo ER se utiliza un triángulo invertido, con la base paralela al supertipo y conectada a los subtipos. Se caracteriza porque toda ocurrencia del subtipo es una ocurrencia del supertipo, pero puede no ocurrir que toda ocurrencia del supertipo pertenezca a un subtipo. Por tanto, las cardinalidades son siempre (1,1) en el supertipo y (0,1) en los subtipos.

Las restricciones que se pueden definir sobre una jerarquía son:

- Totalidad o parcialidad. Existe totalidad cuando toda ocurrencia del supertipo debe pertenecer a uno de los subtipos. Si existe parcialidad, pueden existir ocurrencias del supertipo que no estén en ningún subtipo.
- Exclusividad o solapamiento. Existe exclusividad si una ocurrencia del supertipo sólo puede pertenecer a uno de los subtipos. Si existe solapamiento, una ocurrencia puede pertenecer a varios subtipos.

Existen diferentes posibilidades de transformación de esta interrelación al modelo relacional, con la consiguiente pérdida de semántica en cada una de ellas [Teorey, 1999] [Elmasri y Navathe, 2004]. El más común es crear una relación para el supertipo y para cada uno de los subtipos con sus atributos, aunque también es posible transformar toda la jerarquía a una única relación con todos los atributos o crear sólo relaciones para los subtipos que contendrán tanto los atributos propios como los comunes.

La elección de una u otra posibilidad depende del tipo de jerarquía a representar y de lo que el usuario desee hacer primar (eficiencia o semántica). Por ejemplo, cuando se transforma una jerarquía creando sólo relaciones para los subtipos que contengan además de los atributos propios, los atributos comunes; se aumenta la eficiencia ante determinadas consultas (las que afecten a todos los atributos, tanto comunes como propios de un subtipo) pero puede disminuir ante otras consultas. Esta solución es la que más semántica pierde, además, si existe solapamiento se introduce redundancia que debe ser controlada si se quiere evitar inconsistencia [De Miguel et al., 1999].

En este trabajo se ha considerado la jerarquía de tipo Total y Exclusiva porque es el más común y el más crítico. Si suponemos la interrelación jerárquica representada en la tabla 3.5, todo ejemplar de *A* debe corresponder con un ejemplar de *B* o un ejemplar de *C*, pero no con los dos a la vez. El atributo discriminante es Tipo, que va a permitir clasificar los ejemplares de *A*. Se precisa un CHECK que controle el valor del atributo Tipo para que no pueda tomar valor nulo ni tampoco tomar otros valores.

La transformación y el análisis de la pérdida de semántica en este caso se muestran en la tabla 3.5. Cuando se inserta una ocurrencia en el supertipo *A* o cuando se borra una ocurrencia de los subtipos de *B* o *C*, existe la posibilidad de perder la restricción de totalidad de la jerarquía, es decir, tener un ejemplar del supertipo sin pertenecer a ningún subtipo. Sin embargo, cuando se inserta en los subtipos *B* o *C* existe la posibilidad de perder la restricción de exclusividad de la jerarquía, es decir, tener una ocurrencia del supertipo que pertenezca a más de un subtipo. Esta pérdida de semántica se puede controlar utilizando disparadores que pueden verificar cualquier modificación en estas relaciones.

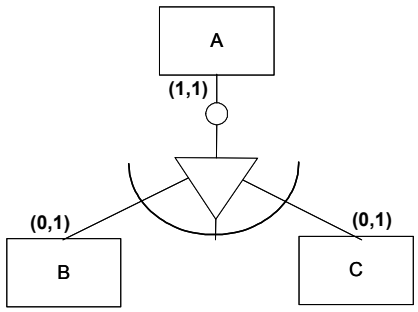
Casos	**Transformación**	**Pérdidas de semántica**
Caso (1) Totalidad y Exclusividad 	A (ID_A, …, Tipo) B (ID_A, ….) DC, UD C (ID_A, ….) DC, UD A_en_ RJ: Obligatorio B_en_RJ: Opcional C_en_RJ: Opcional	INSERT en A ó B ó C DELETE de B ó C

Tabla 3.5. La pérdida de semántica en una jerarquía Total y Exclusiva

3.3 Metodología Propuesta para la Conservación de Restricciones de Integridad en BD

Una vez vistos los distintos mecanismos para contemplar las restricciones de integridad, en este apartado se describirá la metodología propuesta para la conservación de las restricciones de integridad en las BDs relacionales. También, se definirán las reglas que se utilizarán para transformar una restricción de integridad a un disparador del estándar SQL3. Se realizará un seguimiento completo de las restricciones de integridad empezando por la especificación de estas restricciones en el esquema conceptual y terminando por generar mecanismos para conservar estas restricciones y verificar estos mecanismos.

A partir de un esquema conceptual, en la primera fase se obtienen las especificaciones de las restricciones de integridad definidas en el esquema conceptual en un lenguaje formal como el cálculo relacional de tuplas (TRC) [Elmasri y Navathe, 2004]. La segunda fase transforma estas cláusulas TRC a disparadores del estándar SQL3; por ello se han refinado las reglas de transformación encontradas en la literatura [Türker y Gertz, 2000] [Decker, 2002] para transformar las cláusulas TRC obtenidas de la fase anterior a disparadores formalizados según el estándar SQL3. Estas reglas de transformación se basan en: detectar las operaciones críticas que pueden violar una restricción, determinar la granularidad del disparador, especificar la condición del disparador, y determinar el tiempo de activación de cada disparador. La tercera fase transforma los disparadores del estándar SQL3 a disparadores de un SGBD determinado. Para la implementación de esta fase se ha utilizado el SGBD ORACLE [Oracle, 2005] por ser el más extendido. En esta fase se detectan los problemas asociados con los disparadores del SGBD elegido para evitar la no terminación y los problemas específicos el mismo. A continuación se explicarán estas fases (véase la figura 3.5) con más detalle.

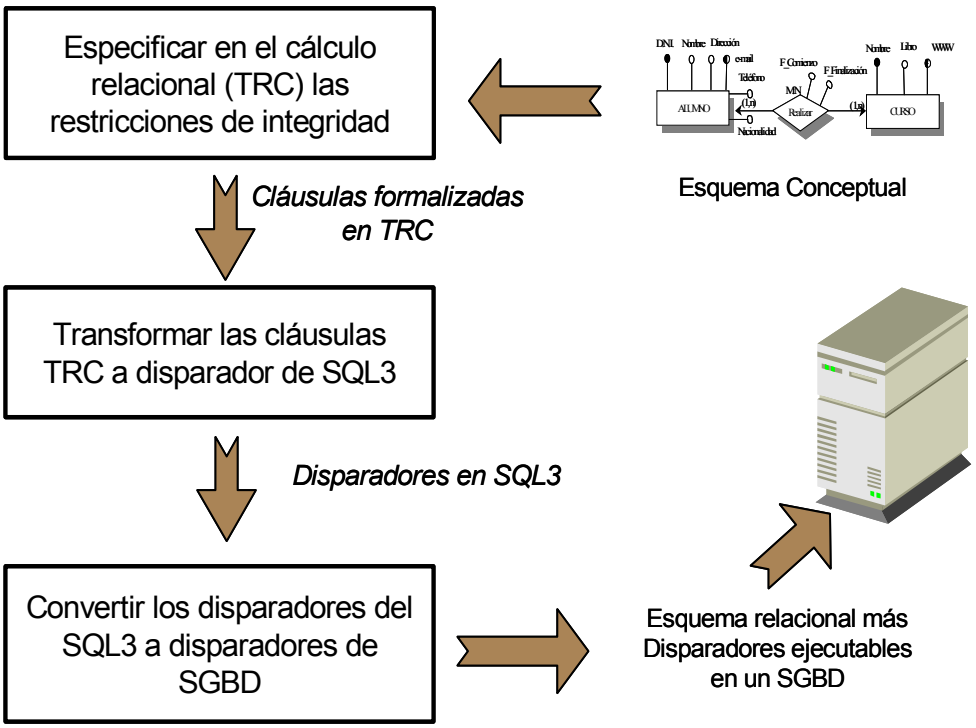


Figura 3.5. Las fases de la metodología propuesta

FASE 1. Especificar en el Cálculo Relacional las Restricciones de Integridad

Como se ha explicado anteriormente, en este trabajo se representan las restricciones de integridad mediante reglas activas (evento, condición, y acción). El evento es una operación de actualización, la condición es una proposición lógica definida sobre uno o varios elementos del esquema, y la acción es el código que se ejecuta dependiendo del resultado de evaluar la condición.

La derivación de reglas activas a partir de la especificación de las restricciones de integridad de la BD es un tema muy estudiado en la literatura. La mayoría de los lenguajes utilizados para especificar las restricciones se derivan de las notaciones formales del cálculo relacional de tuplas (Tuple Relational Calculus-TRC) [Ceri et al., 1994] [Ishakbeyoglu y Ozsoyoglu, 1998] [Elmasri y Navathe, 2004] [Decker,

2002]. El cálculo relacional de tuplas se basa en la especificación de un cierto número de variables de tupla, lo que significa que una variable puede adoptar como valor cualquier tupla de una relación.

Típicamente, se puede especificar una restricción de integridad de tal manera que todas las tuplas cualificadas de una tabla o de una combinación de tablas tengan que satisfacer un predicado que contenga operadores de comparación ($\leq \geq \neq < = >$) u operadores lógicos ($\wedge$ AND, $\vee$ OR, $\neg$ NOT). Por ejemplo, "el sueldo de un director tiene que ser igual o mayor que 1500 €" (Empleado.sueldo $\geq$ 1500 € $\Rightarrow$ Empleado.tipo = 'Director"). Como se muestra a continuación:

Una restricción de integridad en TRC es:

$\forall e_1 \in R_1, e_2 \in R_2, ..., e_n \in R_n$: <COND>

Donde:

e_1, e_2, ….: Son las variables de tuplas de las relaciones R_1, R_2, ….., R_n. Una restricción de integridad puede asociar con una o más tablas del esquema $\Re = \{ R_1, R_2, ..., R_n \}$.

<COND>: La condición es una expresión lógica sobre las columnas de las relaciones $R_1, R_2, ..., R_n$.

Donde:

```
COND ⇒ COND ∧ COND
     | COND ∨ COND
     | ¬ COND
     | (∀ t: COND)
     | (∃ t: COND)
     | ATOM
```

ATOM es una o más formulas, como; e $\in$ R, o e.a Θ c donde Θ: $\{\leq \geq \neq < = >\}$

Una cláusula del cálculo relacional que representa una restricción de integridad puede definirse de manera manual, utilizando este lenguaje para especificar las restricciones, como por ejemplo, la propuesta presentada en [Olivé, 2003] que utiliza

el cálculo relacional para describir restricciones de integridad en los métodos del diagrama de clases de UML. También pueden definirse de manera automática, utilizando un motor que derive cláusulas del cálculo relacional de las restricciones de integridad contempladas en el diagrama conceptual.

En la metodología propuesta se ha utilizado la segunda opción porque el objetivo es escoger e implementar todas las restricciones de integridad del modelo conceptual que no tienen una transformación directa. Las restricciones de integridad vistas en el apartado anterior 3.1.2 (las interrelaciones binarias de todos los tipos N:M, 1:N, 1:1 y las jerarquías de tipo totalidad y exclusividad) tienen representación gráfica en el modelo conceptual pero no se transforman a ningún mecanismo para conservarlas en el modelo relacional.

Por ello, y para facilitar el trabajo a los desarrolladores de BD, en esta metodología propone un mecanismo para derivar las cláusulas del cálculo relacional automáticamente a partir de un diagrama conceptual.

Así pues, para una restricción de cardinalidad de tipo N:M se puede representar cada rol en la interrelación utilizando una cláusula del calculo relacional, como se muestra a continuación:

Caso (1), véase la tabla 3.2.

```
Rol (A_en_R: Obligatorio): TIPO N:M
     ∀ a ∈ A:
     ∃ r ∈ R: r.ID_A = a.ID_A
Rol (B_en_R: Obligatorio): TIPO N:M
     ∀ b ∈ B:
     ∃ r ∈ R: r.ID_B = b.ID_B
```

En el caso de que cualquiera de estos roles sea una participación opcional entonces la cláusula que lo representa será nula, es decir, se elimina del proceso de transformación porque es rarísimo utilizar disparadores para controlar la participación opcional. Los nombres de las tablas, las claves primarias, y las claves

ajenas son parámetros que pueden definirse al principio del proceso para personalizar estas cláusulas para cada rol en la interrelación.

Para una restricción de cardinalidad de tipo 1:N se puede representar cada rol que se juega en la interrelación utilizando una cláusula del calculo relacional, como se muestra a continuación:

Caso (1), véase la tabla 3.3

```
Rol (A_en_B: Obligatorio): TIPO 1:N
     ∀ a ∈ A:
     ∃ b ∈ B: b.ID_A = a.ID_A
Rol (B_en_A: Obligatorio): TIPO 1:N
     ∀ b ∈ B:
     ∃ a ∈ A: a.ID_A = b.ID_A
```

El rol B_en_A indica que cualquier ocurrencia en B tiene que relacionarse con una en A. En este caso, utilizar un disparador para recoger esta restricción se considera redundante porque se puede controlar utilizando la cláusula NOT NULL a la hora de definir el campo de la clave ajena. Quedar por controlar la primera restricción utilizando disparadores.

En el caso de las restricciones de cardinalidad máxima conocida o mínima mayor que uno, cada rol en estas interrelaciones se representa utilizando las cláusulas que se muestran a continuación (véase la tabla 3.4):

```
Rol (A_en_R: Obligatorio): TIPO N:M
     ∀ a ∈ A:
     ∃ r ∈ R: (COUNT(r.ID_A)>=x2 ∧ COUNT(r.ID_A)<=y2)
Rol (B_en_R: Obligatorio): TIPO N:M
     ∀ b ∈ B:
     ∃ r ∈ R: (COUNT(r.ID_B)>=x1 ∧ COUNT(r.ID_B)<=y1)
```

Donde las variables x1 y x2 son las cardinalidades mínimas; y1 e y2 son las cardinalidades máximas de las participaciones de cada entidad en la interrelación.

Para las jerarquías se ha contemplado solo el tipo de totalidad y exclusividad por ser el más común y el más crítico. Según la tabla 3.5, la totalidad que indica que toda ocurrencia del supertipo debe pertenecer a uno de los subtipos, se representa:

```
Rol (A_en_RJ: Obligatorio): TIPO Supertipo/Jerarquía
     ∀ a ∈ A:
     (∃ b ∈ B: a.ID_A = b.ID_A) ∨ (∃ c ∈ C: a.ID_A =
     c.ID_A)
```

Para controlar la exclusividad de una jerarquía que indica que una ocurrencia del supertipo sólo puede pertenecer a uno de los subtipos se definen las siguientes cláusulas:

```
Rol (B_en_RJ: Opcional): TIPO Subtipo/Jerarquía
     ∀ b ∈ B:
     ¬∃ c ∈ C: b.ID_A = c.ID_A
Rol (C_en_RJ: Opcional): TIPO Subtipo/Jerarquía
     ∀ C ∈ C:
     ¬∃ b ∈ B: c.ID_A = b.ID_A
```

A partir de estas cláusulas, ya se puede abordar la siguiente fase en la metodología propuesta que se ocupa de transformar estas cláusulas del cálculo relacional a disparadores del estándar SQL3.

FASE 2. Transformar una Restricción de Integridad a un Disparador de SQL3

La transformación de restricciones de integridad semánticas abordadas en este trabajo se ha llevado a cabo en el marco de BD relacionales y especificación de disparadores en SQL3. Aunque todos los sistemas activos usan distintas sintaxis de disparadores y distintos modelos de ejecución, la transformación de una restricción de integridad a una regla activa sigue algunas reglas comunes independientes de las características del sistema comercial [Türker y Gertz, 2000].

1. Detección de las operaciones críticas que afecten las restricciones de integridad

Para una restricción de integridad es importante detectar primero las operaciones críticas (INSERT, DELETE, o UPDATE) que pueden afectar esta restricción, es decir, las actualizaciones que pueden producir una violación de una restricción de integridad. La importancia de esta fase viene motivada por saber exactamente ¿Cuáles son las operaciones que pueden violar una restricción determinada? Cuando una operación alerta a una restricción esto significa que es necesario implementar un mecanismo para controlar esa restricción.

Para mostrar cómo se detectan las operaciones críticas primeramente se presentarán varios ejemplos para a continuación mostrar las reglas.

RI1: "La experiencia en años de un director debe ser mayor que la de sus empleados":

```
∀ d ∈ Empleado, e ∈ Empleado:
      d.tipo = 'dir' ∧ e.tipo = 'emp'
      ⇒ d.año > e.año
```

- DELETE: El borrado un director o un empleado no alerta RI1.
- INSERT: La inserción de un director o de un empleado si puede violar RI1 porque se deben verificar que los valores nuevos de los atributos que aparecen (año o tipo) en la tupla insertada coinciden con la especificación de RI1.
- UPDATE: Se puede considerar que una actualización es una operación de borrado de los valores antiguos y una operación de inserción de nuevos valores, y por ello, una operación de actualización puede violar cualquiera de los atributos que aparecen en RI1 (año y tipo).

RI2: "Cada departamento gestiona al menos un proyecto":

```
∀ d ∈ Departamento:
      ∃ p ∈ Proyecto: d.id = p.deptno
```

- DELETE: El borrado de un proyecto puede violar RI2 porque puede suceder que se borre el último proyecto relacionado con un departamento.
- INSERT: La inserción de un departamento nuevo puede violar RI2 porque cada departamento tiene que relacionarse con al menos un proyecto.
- UPDATE: La actualización de la columna (deptno) en la tabla proyecto puede violar RI2 porque puede ocurrir que se actualice el último proyecto relacionado con un departamento.

RI3: "El sueldo de un empleado puede aumentar como máximo en un 5%":

$$\forall\ e \in \text{Empleado},\ e' \in \text{Empleado}:$$
$$e.id = e'.id \wedge e'.tipo = \text{'emp'}$$
$$\Rightarrow e.sueldo \leq e'.sueldo * 1.05$$

- DELETE: El borrado de un director no viola RI3.
- INSERT: La inserción de un director no viola RI3.
- UPDATE: La actualización de la columna (sueldo) puede violar RI3 donde se debe comprobar que el aumento no supera al 5%.

RI4: "El sueldo total de los empleados que trabajan en un departamento no debe exceder el presupuesto del departamento":

$$\forall\ d \in \text{Departamento}:$$
$$d.total \geq SUM\{\{e.sueldo \mid e \in Empleado \wedge e.deptno = d.id\}\}$$

- DELETE: El borrado de un empleado no viola RI4.
- INSERT: La inserción de un empleado puede violar la RI4 donde se debe controlar que la suma total de los sueldos de los empleados que pertenecen al departamento no supera el presupuesto del mismo.
- UPDATE: La actualización de la columna (total) de un departamento puede afectar a la RI4, así como la actualización de uno o más sueldos de los empleados.

Según los resultados obtenidos en [Türker y Gertz, 2000] se han derivado y refinado un conjunto de reglas para detectar las operaciones críticas a tener en cuenta cuando se realice la transformación de cláusulas relacionales a disparadores. A continuación se presentan las reglas:

1. Las restricciones que se especifican por variables cuantificadas universalmente (∀), tendrán como operaciones críticas las inserciones en la tabla asociada a estas variables. Ejemplos de estas reglas son RI1 (Empleado), RI2 (Departamento), RI4 (Departamento).
2. En las restricciones que se especifican mediante variables cuantificadas existencialmente (∃), las operaciones críticas serán los borrados de la tabla asociada a estas variable, por ejemplo RI2 (Proyecto).
3. En ambos casos (a) y (b), las operaciones de actualización de las columnas utilizadas en las comparaciones también serán críticas. Por ejemplo, RI1 (Empleado.sueldo, Empleado.tipo), RI2 (Proyecto.deptno), RI4 (Departamento.total).
4. En el caso de haber una comparación entre valores de variables o tablas de transición vistas en la sección 2.3.1.2, las operaciones de actualización de las columnas asociadas son críticas.

   Como por ejemplo ocurre en RI3 (Empleado.sueldo).

2. Determinar la granularidad del disparador

Como se explica en el capítulo 2, la granularidad de transición indica la interrelación entre la ocurrencia de un evento y la instanciación de la regla que ha sido disparada. Existen dos tipos de granularidad: orientada a tupla y orientada a conjunto. En esta sección se describe las reglas generales que se pueden utilizar para definir la granularidad del disparador que se va a crear para cada operación crítica. Es decir, ver si la comprobación de cada restricción de integridad se realiza para cada tupla individual o para todas las tuplas afectadas por esa operación. Si la operación afecta a solo una tupla, entonces se utiliza un disparador de granularidad orientada a tupla,

mientras que si la operación afecta al conjunto de las tuplas afectadas se utilizará un disparador de granularidad orientada a conjunto.

Las reglas que se han utilizado para determinar las granularidades de los disparadores son:

(a) Toda restricción puede verificarse por un disparador orientado a conjunto. Según la propuesta de [Türker y Gertz, 2000] por razones de rendimiento se considera que los disparadores orientados a tupla son preferibles porque permiten verificar condiciones solamente en las tuplas actualizadas.

Por ejemplo, en la restricción de integridad RI1 se pueden utilizar los dos tipos de disparadores pero, en el caso de utilizar un disparador orientado a tupla, ese disparador debe verificar RI1 solamente cuando se inserta una tupla nueva en la tabla Empleado con independencia del tamaño de la tabla. Si se utiliza un disparador orientado a conjunto, el disparador verificará todas las tuplas que puede haber en la tabla y esto es computacionalmente muy costoso.

(b) Las restricciones de integridad que incluyen funciones agregadas requieren siempre por lo menos un disparador orientado a conjunto para la tabla y las tuplas especificadas en la agregación.

Por ejemplo, para RI4 se requiere un disparador orientado a conjunto para las operaciones insertar un empleado y modificar el sueldo de un empleado.

(c) Las restricciones con condiciones generales complejas, según lo presentado en la sección 3.1.1, pueden ser verificadas solamente utilizando disparadores orientados a tupla ya que son los únicos que en el estándar SQL3 utilizan la cláusula WHEN.

(d) Toda restricción que compare valores que ya están en la BD con los valores nuevos que se van a actualizar serán verificadas a través de disparadores orientados a la tupla debido a que son los únicos disparadores en el estándar SQL3 que permiten utilizar variables de transición (NEW, OLD). Por ejemplo, RI3 se transforma a un disparador orientado a tupla.

3. Especificar el tiempo de activación del disparador

Aunque en el estándar SQL se permiten los dos tipos de tiempo de activación (BEFORE y AFTER) sin ningún tipo de restricción, la mayoría de los sistemas comerciales tienen algunas restricciones sobre el tiempo de activación. Por ejemplo, los sistemas ORACLE 9i y DB2 permiten activar dos tipos de disparadores según el tiempo de activación (BEFORE/AFTER), y el sistema MS-SQL SERVER 2005 permite solo los disparadores de tipo AFTER. El estándar SQL3 recomienda que los disparadores de tipo BEFORE se utilicen para leer de la BD, y los de tipo tupla (FOR EACH ROW) sean los que utilicen las sentencias de asignación para modificar los valores introducidos [Cochrane et al., 1996]. Por ejemplo, "Asegúrese que el sueldo nunca sea mayor de 5000 €", en este caso, se puede emplear un disparador de inserción de tipo BEFORE-FOR EACH ROW para garantizar que el Sueldo = 5000, si el nuevo valor es mayor de 5000.

Por ello, las reglas que se van a utilizar en nuestro trabajo para especificar el tiempo de activación de cada disparador son las siguientes:

(a) Cuando se tenga en la cláusula TRC solo la variable de transición NEW en la condición, entonces se utilizará un tiempo de activación BEFORE, ya que no es necesario comparar con otros valores que ya están en la BD.

(b) En los demás casos se utilizará un tiempo de activación AFTER, como por ejemplo sucede en las restricciones RI1, RI2, RI3, y RI4.

4. Convertir las cláusulas del cálculo relacional a condiciones de SQL3

Una vez establecidas las operaciones críticas y las granularidades de los disparadores, el siguiente paso es convertir las cláusulas de TRC a condiciones expresadas en SQL3. Estas condiciones serán la negación lógica de la restricción de integridad. Por ejemplo, si la restricción indica que en la base de datos sólo pueden existir empleados mayores de 18 años, la condición para que se active un disparador será cuando esta restricción no se cumpla, es decir, cuando la edad del empleado sea menor que 18.

A continuación se presentan cómo se aplica este paso a los ejemplos estudiados anteriormente.

¬RI1: "La experiencia en años de un director debe ser menor o igual que la de sus empleados":

```
≡ ∀ d ∈ Empleado, e ∈ Empleado:
      d.tipo = 'dir' ∧ e.tipo = 'emp'
      ⇒ d.año ≤ e.año
≡ EXISTS
      (SELECT * FROM Empleado e1, Empleado e2
      WHERE e1.tipo= 'dir' AND e2.tipo= 'emp' AND
      e1.año =< e2.año);
```

¬RI2: "Existen departamentos que no gestionan proyectos":

```
≡ ∀ d ∈ Departamento:
      ¬ ∃ p ∈ Proyecto: d.id = p.deptno
≡ EXISTS
      (SELECT * FROM DEPARTAMENTO WHERE id NOT IN
      (SELECT deptno FROM PROYECTO))
```

¬RI3: "El sueldo de un empleado puede aumentar en más de un 5%":

```
≡ ∀ e ∈ Empleado, e' ∈ Empleado:
      e.id = e'.id ∧ e'.tipo = 'emp'
      ⇒ e.sueldo > e'.sueldo*1.05
≡ EXISTS
      (SELECT * FROM Empleado
      WHERE NEW.id = OLD.id AND OLD.tipo = 'emp'
      AND NEW.sueldo > OLD.sueldo*1.05);
```

¬RI4: "El sueldo total de los empleados que trabajan en un departamento puede exceder el presupuesto del departamento":

```
≡ ∀ d ∈ Departamento:
      d.total < SUM{{e.sueldo| e ∈ Empleado ∧
      e.deptno = d.id}}
≡ EXISTS
      (SELECT * FROM Empleado e, Departamento d
      GROUP BY e.deptno
      HAVING d.total < SUM(e.sueldo));
```

Así pues, para convertir las cláusulas TRC a condiciones del SQL3 se tendrá que utilizar la siguiente sintaxis:

EXISTS (SELECT *Columnas*

FROM *Tablas*

WHERE *Condición*);

A continuación, pasamos a determinar las incógnitas de esta sintaxis como son las columnas, las tablas y las condiciones dependiendo de la cláusula TRC.

Columnas:

- Las columnas son los atributos que aparecen en la cláusula TRC, en la mayoría de los casos se ha utilizado el asterisco (*) porque lo más importante es saber si la consulta devuelve resultado o no; si la consulta devuelve cualquier resultado significa que existe una violación de la restricción y hay que controlar esa violación.

Tablas:

- Todas las tablas que aparecen en la cláusula TRC aparecen en la cláusula FROM.
- En el caso de que aparezca la misma tabla más de una vez con distintas variables, se utilizarán tantos alias como variables. La restricción RI1 muestra esta situación (FROM Empleado d, Empleado e).

Condición:

- En la condición se especifican todas las formulas (*atoms*) introducidas en la cláusula TRC.
- En el caso de tener una restricción condicional con el símbolo de implicación (⇒), este símbolo se transforma al operador AND. Un ejemplo se muestra en RI1 y RI3.
- En el caso de tener variables cuantificadas existencialmente (∃) especificadas en la condición, se convierte la cláusula a una sentencia SELECT *columna* FROM *tabla*; con los nombres de la columna y la tabla que aparecen en la misma cláusula.
- En el caso de tener funciones agregadas MIN, MAX, SUM, COUNT, o AVG se transformarán estas funciones a SQL con la especificación de la sentencia GROUP_BY *clave* HAVING *condición*. Donde *clave* es la clave compartida entre las tablas, y *condición* es la condición que gobierna la agrupación, como sucede en RI4.

FASE 3. Convertir los Disparadores de SQL3 a Disparadores de un SGBD

Aunque los disparadores de los SGBDs relacionales tienen casi los mismos componentes que los disparadores del estándar SQL3 (véase tabla 2.3), en algunos casos la transformación de un disparador del estándar SQL3 a un disparador correspondiente en un SGBD no es directa. Además, la ausencia de las variables de transición de tipo tabla en algunos SGBDs como ORACLE, han obligado a transformar un disparador del SQL3 a dos disparadores BEFORE y AFTER en este SGBD. Más concretamente, los disparadores de tipo BEFORE se han utilizado para guardar los valores de las tuplas modificadas mientras los disparadores de tipo AFTER se han utilizado para comprobar la semántica. Para pasar variables entre estos dos disparadores se ha necesitado crear paquetes. En el caso de necesitar dos

disparadores, la semántica incluida en el disparador de tipo SQL3 se transforma al disparador de tipo AFTER del SGBD.

La principal tarea asociada a esta transformación es solucionar el problema de la no-terminación en la ejecución de los disparadores, garantizando que la ejecución de cualquier conjunto de reglas debe terminar correctamente. Para garantizar este final se debe evitar que las reglas se activen de forma cíclica.

En este contexto, en esta propuesta se han tratado tres elementos básicos como un caso de estudio de restricciones de integridad problemáticas en el modelo ER. Estos elementos son: la interrelación binaria de tipo N:M, interrelación binaria de tipo 1:N, y la jerarquía total y exclusiva. Utilizar disparadores para controlar la semántica de cardinalidades en las interrelaciones binarias N:M y 1:N no produce ningún ciclo infinito en la ejecución de estos disparadores porque la acción que realizan estos disparadores siempre es el rechazo, es decir, están diseñados para actuar solo sobre una tabla sin modificar cualquiera otra tabla. Mientras, si se produce no-terminación y un ciclo infinito en el caso de la jerarquía total y exclusiva porque según este trabajo estos disparadores pueden lanzar acciones para controlar la semántica. Por ejemplo, cuando se borra una tupla de un subtipo se borra del supertipo porque ya no tiene sentido estar sin subtipo, o cuando se inserta en un subtipo se lanza una operación de borrado sobre los otros subtipos para que se garantice que no existe ningún otro subtipo. Así que, en estos casos si hemos intervenido para solucionar el problema de la no-terminación utilizando semáforos como funciones de valor booleano. Estos semáforos pueden intervenir para desactivar los disparadores problemáticos que puedan producir un ciclo.

Se han implementado los semáforos porque en la mayoría de los SGBDs sólo se pueden crear o eliminar disparadores, es decir, no se permite realizar una modificación de los mismos (ALTER) lo que implica la compilación siempre que se ejecuten. Una vez que el disparador se crea no se permite modificar su definición en tiempo de ejecución. La necesidad de modificar algunas propiedades del disparador durante este tiempo (*runtime*) han motivado la aparición de dos funciones que se definen en el cuerpo de los disparadores que pueden producir ciclos [Gehani, 1991]

[Díaz y Jaime, 1997] [Díaz, 1998]. Estos semáforos o funciones se muestran a continuación:

- La primera función es IS_IT_ENABLED (*trigger name*). Esta función devuelve un valor booleano que describe el estado de cada disparador, es decir, si el disparador puede ser activado o no. Esta función se incorpora como condición en el cuerpo del disparador y se evalúa antes de ejecutar la acción. Si el valor de esta función es verdadero, se ejecuta la acción del disparador; si no lo es, se termina la ejecución del disparador inmediatamente. La modificación del valor de esta función puede hacerse de manera dinámica y por lo tanto cambiar el estado del disparador evitando que se ejecute para impedir la aparición de ciclos.
- La segunda opción es utilizar la función DISABLE (*trigger name*) que devuelve también un valor booleano que cambia el estado del disparador problemático de activado a desactivado.

Con estas dos funciones se logra evitar que un disparador problemático se ejecute dentro de un conjunto de disparadores con estado activo. El estado activado de este disparador se transforma de manera automática cuando se consume todo el conjunto de disparadores en cuestión.

La segunda tarea es la optimización de los disparadores. En esta tarea se optimiza el número de disparadores generados mejorando la eficacia de la BD. La optimización se realiza en la fase de la transformación de los disparadores SQL3 a sus correspondientes dentro de la plataforma utilizada, dependiendo de las características específicas de esta plataforma. En esta fase se reúnen los disparadores que comparten los mismos atributos (Tabla, Evento, Tiempo, Granularidad) en solo un disparador.

La tercera tarea es solucionar los problemas específicos asociadas con la ejecución de los disparadores en cada SGBD. Por ejemplo, el problema de tablas mutantes en ORACLE. Una tabla “mutante” es una tabla que está siendo actualizada por una operación de actualización (INSERT, DELETE o UPDATE). También, puede ser una tabla que podría ser modificada por los efectos de las acciones de integridad referencial (ON DELETE CASCADE, ON UPDATE CASCADE).

3.4 Casos de Estudio: Restricciones de Integridad Problemáticas

En este apartado se va a presentar la aplicación de la metodología propuesta para solucionar la pérdida semántica de algunas restricciones de integridad problemáticas vistas en el apartado 3.1.2. Concretamente, nos centraremos en la aplicación de las reglas estudiadas en el apartado anterior para las interrelaciones binarias y las jerarquías, al ser elementos conceptuales que no tienen una transformación directa al modelo relacional. La importancia de aplicar la metodología propuesta sobre estos constructores estriba en mantener la cardinalidad correcta entre las instancias de las entidades asociadas con estas interrelaciones.

La metodología propuesta comienza a partir de un esquema conceptual, detectando y analizando todas las restricciones contempladas dentro del esquema en cuestión. Cualquier restricción que no tiene una transformación directa se convierte a una cláusula del cálculo relacional de tupla (TRC). Esta cláusula se transforma a un disparador del estándar SQL3 y en el último paso, se convierte a disparadores de un SGBD determinado. En esta fase final se solucionan algunos problemas de la ejecución de disparadores. El resultado final es un esquema relacional más un conjunto de disparadores y estructuras auxiliares que aseguran la semántica reflejada en el esquema conceptual.

Así pues, en este apartado se van a exponer como ejemplos de estudios solo los casos más críticos de las restricciones vistas en el apartado 3.1.2 aplicando las fases de la metodología propuesta.

Caso 1: Interrelación binaria N:M con participación obligatoria

El primer caso de estudio se corresponde con una interrelación binaria N:M con participación obligatoria de ambas entidades. El análisis de la transformación se muestra resumido en la tabla 3.2.

La redefinición de estas restricciones se realiza de forma automática cuando se utiliza una herramienta generadora. Ya que toda restricción de cardinalidad que indique una participación obligatoria tendrá la misma sintaxis, lo que variará será el nombre de la entidad y de la interrelación. Así pues, se presenta a continuación solo una restricción de integridad RI1 como muestra de este tipo de interrelación:

Sea A un tipo de entidad con participación obligatoria en R. La cláusula relacional que específica esta restricción será:

RI1: A_en_R:

```
∀ a ∈ A:
     ∃ r ∈ R: r.ID_A = a.ID_A
```

La transformación de esta interrelación al modelo relacional se muestra a continuación:

```
CREATE TABLE A (ID_A,……);
CREATE TABLE R (ID_A, ID_B,
     FOREIGN KEY (ID_A) REFERENCES ALUMNO
          ON DELETE CASCADE, ON UPDATE CASCADE,……);
```

Según las reglas de transformación del apartado 3.2, se puede deducir que las operaciones críticas que pueden violar esta restricción son:

- INSERT.A.NEW.ID_A;
- DELETE.A.ID_A:=OLD.ID_A;
- DELETE.R.ID_A:=OLD.ID_A;
- UPDATE.R.OLD.ID_A:= NEW.ID_A;

En este trabajo no se controla la semántica cuando se inserta en las tablas procedentes de entidades mientras si se controla la semántica cuando se inserta en las tablas procedentes de la interrelación.

Se pasa ahora a determinar la granularidad de los disparadores que se van a utilizar en este caso. Vistas las reglas de transformación en la FASE 2.2, se puede deducir que las granularidades de los disparadores son orientadas a tuplas porque el estándar SQL3 permite utilizar la cláusula WHEN solamente en este tipo de disparadores. Además, la necesidad de utilizar los valores de transición OLD y NEW nos obliga a utilizar este tipo de granularidad que permiten utilizar estos valores. Así, como resumen, la granularidad utilizada en todos los disparadores es orientada a tupla (FOR EACH ROW).

Se pasa ahora a la fase de determinar el tiempo de activación de los disparadores, si el disparador se activa antes o después (BEFORE/AFTER) del evento. Según las reglas propuestas en la FASE 2.3 se puede deducir que se utilizará un tiempo de activación BEFORE en el caso de los disparadores de inserción y AFTER en el caso de los disparadores de los borrados y las actualizaciones.

Por último, se convertirán las cláusulas a condiciones en el SQL3. Para ello, se negarán la restricción RI1.

```
¬RI1: ∀ a ∈ A:
      ¬∃ r ∈ R: r.ID_A = a.ID_A
```

Aplicamos las reglas para transformar una cláusula de TRC a una condición de SQL3. Como resultado se puede convertir RI1 a la siguiente condició del SQL3:

```
WHEN EXISTS
      (SELECT * FROM A WHERE ID_A NOT IN
      (SELECT ID_A FROM R));
```

Finalmente, se debe señalar aquí que algunos eventos en este caso son redundantes, es decir, cuando se borra una tupla de la tabla A o B se borran en cascada todas las tuplas relacionadas en la tabla R. Por eso, se puede considerar que los siguientes eventos:

(DELETE.A.ID_A:=OLD.ID_A) y (DELETE.B.ID_B:=OLD.ID_B)

Son solo un evento:

DELETE.R.ID_A:=OLD.ID_A ∧ ID_B:=OLD.ID_B)

Así que, en este caso se optimizan implementando solo un disparador sobre la tabla R. Se generan los disparadores de los eventos restantes agregando los resultados de las transformaciones. Se asigna un nombre para cada disparador y una acción que es el rechazo, como se muestra en el siguiente ejemplo:

```
CREATE TRIGGER TDEL_R
AFTER DELETE FROM R
FOR EACH ROW
WHEN EXISTS
     (SELECT * FROM A WHERE ID_A NOT IN
     (SELECT ID_A FROM R)) OR
     (SELECT * FROM B WHERE ID_B NOT IN
     (SELECT ID_B FROM R));
ROLLBACK;
```

A continuación mostraremos un ejemplo concreto para que resulte más sencilla la explicación de cómo aplicar las fases propuestas.

En la figura 3.6 se muestra una interrelación binaria de tipo N:M; las cardinalidades muestran que un alumno puede estudiar una o mas asignaturas y que una asignatura puede ser estudiada por varios alumnos.

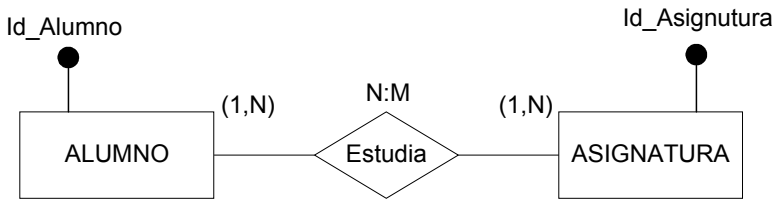


Figura 3.6. Interrelación binaria N:M Estudia

Para controlar el borrado en esta interrelación se utiliza el disparador de SQL3 que se muestra a continuación:

```
CREATE TRIGGER TDEL_ESTUDIA
AFTER DELETE FROM ESTUDIA
FOR EACH ROW
WHEN EXISTS
     (SELECT * FROM ALUMNO WHERE ID_ALUMNO NOT IN
     (SELECT ID_ALUMNO FROM ESTUDIA)) OR
     (SELECT * FROM ASIGNATURA WHERE ID_ASIGNATURA NOT IN
     (SELECT ID_ASIGNATURA FROM ESTUDIA));
ROLLBACK;
```

Para controlar la modificación en esta interrelación se utiliza el disparador de SQL3 que se muestra a continuación:

```
CREATE TRIGGER TUPD_ESTUDIA
AFTER UPDATE FROM ESTUDIA
FOR EACH ROW
WHEN EXISTS
     (SELECT * FROM ALUMNO WHERE :OLD.ID_ALUMNO NOT IN
     (SELECT ID_ALUMNO FROM ESTUDIA)) OR
     (SELECT * FROM ASIGNATURA WHERE :OLD.ID_ASIGNATURA
      NOT IN (SELECT ID_ASIGNATURA FROM ESTUDIA));
ROLLBACK;
```

Caso 2: Interrelación binaria 1: N con participación obligatoria

El segundo caso de estudio se corresponde con una interrelación binaria 1:N con participación obligatoria de ambas entidades. El análisis de la transformación se muestra resumido en la tabla 3.3. Para convertir estas restricciones a disparadores del SQL3 seguimos los pasos de la metodología mostrados en la sección 3.2.

En este caso, las restricciones de las dos entidades *A* y *B* que participen en la interrelación *R*, se transforma de una manera distinta. Ya que la participación obligatoria de la entidad A se puede controlar a través de definir la clave ajena de la entidad A en B como no nulo. Así se consigue que todas las tuplas existentes en la

tabla *B* tengan por lo menos una referencia en la tabla *A*. Mientras, si se necesitan disparadores para controlar la participación obligatoria de la tabla de entidad *A* con *B,* es decir, cada tupla en la tabla *A* debe relacionarse por lo menos con una tupla en la tabla *B*. A continuación se muestra la especificación de esta restricción en TRC: Sea A un tipo de entidad con participación obligatoria en B. La cláusula relacional que específica esta restricción será:

```
RI1: A_en_B
∀ a ∈ A:
     ∃ b ∈ B: b.ID_A = a.ID_A
```

La transformación de esta interrelación al modelo relacional se muestra a continuación:

```
CREATE TABLE A (ID_A,……);
CREATE TABLE B (ID_B,……,ID_A);
     FOREIGN KEY (ID_A) REFERENCES DEPARTAMENTO
     ON DELETE CASCADE, ON UPDATE CASCADE);
```

Según las reglas de transformación del apartado 3.2, se puede deducir que las operaciones críticas son:

- `INSERT.A.(NEW.ID_A, ….);`
- `DELETE.A.ID_A:=OLD.ID_A;`
- `UPDATE.A.OLD.ID_A:=NEW.ID_A;`

La granularidad y el tipo de activación utilizadas en este caso son FOR EACH ROW y BEFORE. A continuación se muestra solo un disparador como ejemplo de este caso:

```
CREATE TRIGGER TDEL_A
AFTER DELETE ON A
FOR EACH ROW
WHEN EXISTS
     (SELECT * FROM A WHERE A
          NOT IN (SELECT A FROM B))
ROLLBACK;
```

Un ejemplo de este caso se muestra en la figura 3.7, una interrelación binaria de tipo 1:N, las cardinalidades muestran que un profesor puede pertenecer a un departamento, y que un departamento puede tener uno o varios profesores.

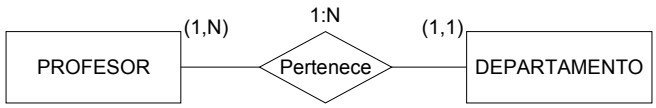


Figura 3.7. Interrelación binaria 1:N Pertenece

Los disparadores necesarios para controlar esta interrelación se muestran a continuación:

```
CREATE TRIGGER TDEL_PROFESOR
AFTER DELETE ON PROFESOR
FOR EACH ROW
WHEN EXISTS
     (SELECT * FROM DEPARTAMENTO WHERE DEPARTAMENTO
          NOT IN (SELECT DEPARTAMENTO FROM PROFESOR))
ROLLBACK;

CREATE TRIGGER TUPD_PROFESOR
AFTER UPDATE ON PROFESOR
FOR EACH ROW
WHEN EXISTS
     (SELECT * FROM DEPARTAMENTO WHERE :OLD.DEPARTAMENTO
      NOT IN (SELECT DEPARTAMENTO FROM PROFESOR))
ROLLBACK;
```

Caso 3: Interrelación binaria N:M con cardinalidad máxima conocida o mínima mayor de uno

Este caso es similar al caso mostrado en la figura 3.6 diferenciable en las cardinalidades mínimas y máximas que son (x1, y1), (x2, y2). La especificación de esta restricción en TRC se muestra a continuación:

```
RI1: ∀ a ∈ A:
     ∃ r ∈ R: (COUNT(r.ID_A)>=x2 ∧ COUNT(r.ID_A)<=y2)
RI2: ∀ b ∈ B:
     ∃ r ∈ R: (COUNT(r.ID_B)>=x1 ∧ COUNT(r.ID_B)<=y1)
```

El disparador utilizado para controlar este caso se muestra a continuación:

```
CREATE TRIGGER TDEL_R
AFTER DELETE FROM R
FOR EACH ROW
WHEN EXISTS
     SELECT COUNT (*) FROM R
          GROUP BY ID_A
          HAVING COUNT (*) < X2 OR COUNT (*)> Y2
     OR EXISTS
     SELECT COUNT (*) FROM R
          GROUP BY ID_B
          HAVING COUNT (*) < X1 or COUNT (*)> Y1
ROLLBACK;
```

De forma similar se define el disparador TUPD_R, TINS_R sobre la tabla R para controlar la modificación y la inserción, respectivamente.

Caso 4: Jerarquía total y exclusiva

El análisis de la transformación se muestra resumido en la tabla 3.5. Para convertir estas restricciones a disparadores del SQL3 seguimos los pasos de nuestra metodología mostrados en la sección 3.2.

La totalidad de esta jerarquía indica que una ocurrencia del supertipo *A* debe pertenecer a una de los subtipos, y exclusividad indica que solo puede pertenecer a uno de los subtipos. La transformación de esta interrelación al modelo relacional se muestra a continuación:

```
CREATE TABLE A (ID_A,……, tipo);
CREATE TABLE B (ID_A,……);
     FOREIGN KEY (ID_A) REFERENCES A
     ON DELETE CASCADE, ON UPDATE CASCADE,
CREATE TABLE C (ID_A,…….);
     FOREIGN KEY (ID_B) REFERENCES A
     ON DELETE CASCADE, ON UPDATE CASCADE);
```

La especificación de las restricciones en TRC de esta jerarquía se muestra a continuación:

```
RI1: A_en_RJ (Totalidad)
∀ a ∈ A:
     ∃ b ∈ B: a.ID_A = b.ID_A ∨
     ∃ c ∈ C: a.ID_A = c.ID_A
RI2: B_en_RJ (Exclusividad)
∀ b ∈ B:
     ∃ c ∈ C: b.ID_A = c.ID_A
RI3: (Exclusividad)
∀ c ∈ C:
     ∃ b ∈ B: c.ID_A = b.ID_A
```

Aplicando las reglas de transformación vistas en FASE 2.1, se puede deducir que las operaciones criticas son: primero, la inserción en las tablas *A*, *B*, y *C* porque las tres tablas han sido especificadas por variables cuantificadas universalmente (∀). La inserción en la tabla *A* produce una pérdida de semántica de la totalidad porque esta persona no sería de ninguno de los dos subtipos. Además, en la inserción en *B* o *C* puede que se pierda la semántica de la exclusividad porque es posible que se inserte una clave de identificación que existe en la otra tabla. Segundo, se pierde semántica

también cuando se borra de *B* o *C*.

- `INSERT.A.(NEW.ID_A, ….);`
- `INSERT.B.(NEW.ID_A, ….);`
- `INSERT.C.(NEW.ID_A, ….);`
- `DELETE.B.ID_A:=OLD.ID_A;`
- `DELETE.C.ID_A:=OLD.ID_A;`

En todas las restricciones de las interrelaciones binarias se ha utilizado la acción rechazar (ROLLBACK) para controlar la semántica cuando se produce una violación. En las jerarquías se ha cambiado la estrategia de las acciones utilizadas por el disparador cuando se detecta una violación. Se ha podido utilizar el rechazo como acción de los disparadores que controlan las jerarquías sin ningún problema pero se ha visto que utilizar acciones reparadoras resulta más eficaz y más sencillo a la hora de trabajar con las jerarquías. Por ejemplo, para controlar la totalidad de la jerarquía se ha utilizado la acción borrar del supertipo cuando se borra de un subtipo. Así pues, cada vez que se borra un supertipo se borran en cascada el subtipo relacionado y el contrario también es correcto, cada vez que se borra de un subtipo se borra en cascada del supertipo relacionado. Para controlar el primer caso se utiliza la acción de integridad referencial, mientras que para controlar el segundo caso se utiliza un disparador.

Para controlar la exclusividad de la jerarquía; cada vez que se inserta en un subtipo la acción del disparador de la tabla modificada tiene que comprobar que no existe la clave primaria insertada en la otra tabla. Si está duplicada el disparador tiene que borrarla para que conserve la exclusividad de la interrelación.

Los disparadores utilizados en estos casos se muestran a continuación:

```
CREATE TRIGGER TDEL_B (Totalidad)
AFTER DELETE ON B
WHEN EXISTS (SELECT ID_A FROM A
     WHERE ID_A
     NOT IN (SELECT ID_A FROM B)
     AND ID_A NOT IN (SELECT ID_A FROM C);
```

```
     DELETE FROM A
          WHERE ID_A =: OLD.ID_A;
```

De forma similar se define el disparador TDEL_C sobre la tabla C para controlar la totalidad.

Para controlar la exclusividad de la jerarquía se puede utilizar el siguiente disparador:

```
CREATE TRIGGER TINS_B (Exclusividad)
AFTER INSERT ON B
WHEN EXISTS (SELECT ID_A FROM B
     WHERE :NEW.ID_A
     IN (SELECT ID_A FROM C);
     DELETE FROM C WHERE ID_A=:NEW.ID_A;
```

De forma similar se define el disparador TINS_C sobre la tabla PROFESOR para controlar la exclusividad.

Cabe señalar aquí que estos disparadores pueden producir una no-terminación en la ejecución porque contienen acciones que pueden provocar modificaciones en otras tablas y la activación de otros disparadores incluso de ellos mismos. Por ello, a la hora de implementar estos casos se deben solucionar estos problemas utilizando las funciones vistas en la FASE 3.

En la figura 3.8 se muestra un ejemplo de este tipo de jerarquía.

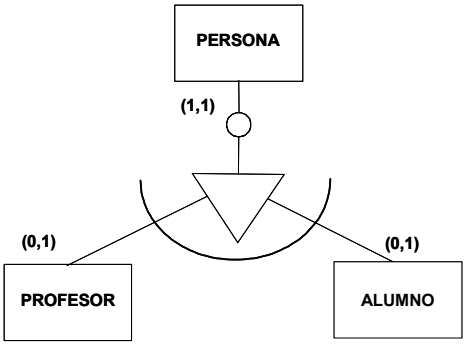


Figura 3.8. Jerarquía de Persona

Los disparadores utilizados para controlar la totalidad se muestran a continuación:

```
CREATE TRIGGER TDEL_PROFESOR
AFTER DELETE ON PROFESOR
WHEN EXISTS (SELECT ID_PERSONA FROM PERSONA
     WHERE ID_PERSONA
     NOT IN (SELECT ID_PERSONA FROM PROFESOR)
     AND ID_A NOT IN (SELECT ID_PERSONA FROM PROFESOR);
     DELETE FROM PERSONA
          WHERE ID_PERSONA =: OLD.ID_PERSONA;
```

De forma similar se define el disparador TDEL_PROFESOR sobre la tabla PROFESOR para controlar la totalidad.

Los disparadores utilizados para controlar la exclusividad se muestran a continuación:

```
CREATE TRIGGER TINS_PROFESOR
AFTER INSERT ON PROFESOR
WHEN EXISTS (SELECT ID_PERSONA FROM PROSEFOR
     WHERE :NEW.ID_PERSONA
     IN (SELECT ID_PERSONA FROM PROFESOR);
     DELETE          FROM          PROFESOR          WHERE
ID_PERSONA=:NEW.ID_PERSONA;
```

De forma similar se define el disparador TINS_PROFESOR sobre la tabla PROFESOR para controlar la exclusividad.

3.5 Un Caso Especial: Obligatoriedad de Participación en Interrelaciones Binarias

En esta sección se presentarán dos propuestas para tratar el problema de la pérdida de semántica en las participaciones obligatorias en las interrelaciones. La primera propuesta es relajar estas participaciones para facilitar la implementación y mejorar el rendimiento. La segunda propuesta es controlar la inserción de nuevas tuplas para conservar la semántica de estas participaciones.

3.5.1 Relajar la Obligatoriedad de Participación

La implementación de una gran cantidad de mecanismos para conservar las restricciones de integridad en la BD puede llevar consigo un impacto significativo sobre el rendimiento del SGBD. Existe una relación entre la semántica recogida de la BD, la complejidad de implementación utilizada para conservar la semántica y el rendimiento del SGBD [Al-Jumaily et al., (a) 2004]. Esta relación puede representarse como un triángulo como se muestra en la figura 3.9. Los vértices de este triángulo pueden moverse en dos direcciones, vertical y horizontal. Se puede ver que al extender el vértice superior hacia arriba, la base del triángulo se reduce. Esto significa que al reforzar la semántica de la BD a través de aumentar la complejidad de la implementación, produce como resultado un mayor rendimiento en el SGBD. Mientras, al extender los vértices inferiores hacia los dos lados, obtiene una reducción en la complejidad de la implementación y al mismo tiempo aumenta el rendimiento de SGBD.

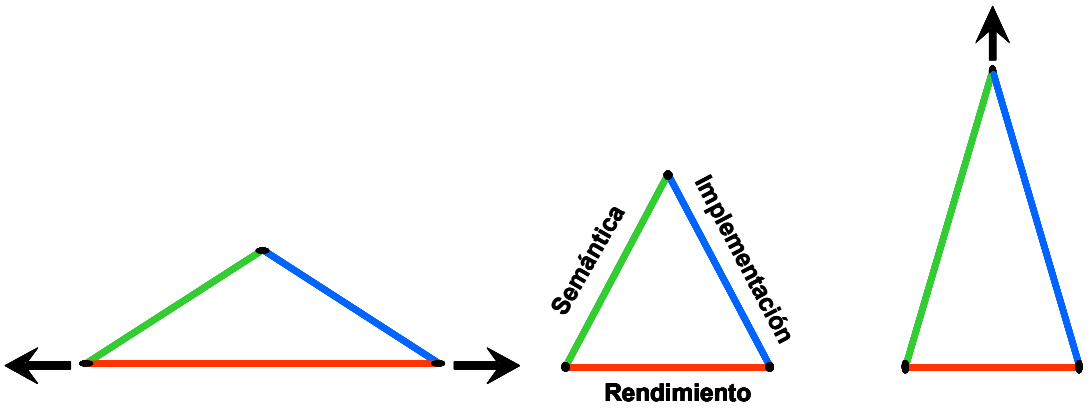


Figura 3.9. La relación entre la semántica, la implementación, y el rendimiento

Los problemas de conservar las restricciones de integridad en general, aparecen principalmente en las BDs que gestionan un gran volumen de datos y poseen muchos elementos y restricciones que controlar.

En general, las restricciones de integridad que más inciden en el rendimiento son las restricciones de cardinalidad, concretamente, cuando la mayoría de las cardinalidades son de participación obligatoria. Y no sólo se ve afectado el rendimiento del SGBD sino que la mayoría de las operaciones de actualización son rechazadas a causa de no cumplirse la obligatoriedad de participación. Esta fue la principal motivación de los resultados que a continuación se presenten. El objetivo perseguido se basa en proporcionar a los desarrolladores un conjunto de medidas para identificar qué restricciones de cardinalidad son más relevantes para poner el peso del control en ellas.

Para aplicar este método hay que definir un conjunto de métricas que puntúan el grado de cumplimiento de las restricciones de integridad en cada tabla de la BD. A través de estas métricas (cuantificadores difusos relativos) los desarrolladores pueden tomar decisiones sobre qué restricciones controlar.

Los cuantificadores difusos relativos expresan mediciones sobre el número total de elementos que cumplen cierta característica dependiendo del total de elementos posibles, por lo que la verdad del cuantificador depende de dos cantidades (0,1) [Urrutia, et al., 2003].

$$q_{ri} = T_{Ej} * \left(R^t\right)^{-1}$$

$$Q_{Ri} = \sum_{j=1}^{j=n} \left(T_{Ej} * \left(R^t\right)^{-1}\right)$$

Donde:

- q_{ri} es el cuantificador relativo de un Rol.
- Q_{Ri} es el cuantificador relativo en una interrelación.
- T_{Ej} las tuplas de entidad E_j que no cumplen la semántica en el rol r_i.
- R^t el número de todas las tuplas que pertenecen a la relación R y que r_i es una parte de esta.

Para aplicar esta métrica hay que tener en cuenta lo siguiente:

1. Hay que aceptar un margen que permita una pérdida de la semántica en cada interrelación en un esquema de BD; este margen será como un límite superior que puede establecerse temporalmente. Este límite superior ayuda a los diseñadores a la evaluación de la semántica en cada elemento del esquema en cuestión.
2. El diseñador del esquema conceptual debe confiar en su diseño, porque él es el único que puede decidir el margen de este límite, y si este límite superior es aplicable a todo el esquema o si puede aplicar distintos límites un en cada interrelación del esquema de la BD. Está claro que la definición de estos límites superiores depende de la importancia de cada elemento en el esquema.
3. Hay que someter la BD a un sondeo (*polling*) en cada periodo de tiempo para controlar y corregir los márgenes de las pérdidas de semántica en cada elemento del esquema.

En una BD real se puede calcular fácilmente estos cuantificadores a través de aplicar dichas fórmulas directamente para obtener la pérdida de semántica en cada elemento. El diseñador puede definir previamente un rango que representa un máximo de pérdida de semántica aceptada. Si se aumenta la pérdida de semántica a dicho rango,

el desarrollador puede activar los mecanismos necesarios para corregir y controlar la pérdida de semántica en los elementos correspondientes.

En nuestra propuesta, la activación o desactivación de los mecanismos que controlan la semántica puede relacionarse con las pérdidas existentes en la BD. Por ejemplo, un diseñador puede aceptar un límite superior de (0.001%) como un máximo de pérdida de semántica en una restricción determinada; si la pérdida sufrida en esta restricción es mayor a dicha cantidad el diseñador debe activar cualquier mecanismo que pueda corregir esta pérdida, sino se pueden quedar estos mecanismos desactivados para que no se sobrecargue la BD.

A continuación, se representa la aplicación de esta propuesta sobre un tipo determinado de las restricciones de integridad.

Nuestra idea en este contexto es relajar algunas participaciones totales u obligatorias (cardinalidad mínima uno) a participaciones opcionales (cardinalidad mínima cero) para facilitar el control de estas restricciones.

Así por ejemplo, dado el esquema conceptual mostrado en la figura 3.10, las cardinalidades mínimas del mismo son todas obligatorias, es decir, que cada departamento tiene que ser relacionado con un mínimo de un curso y viceversa. Cada curso tiene que ser relacionado con un mínimo de alumno y viceversa. Esto implica que cuando se actualiza cualquiera de estos elementos hay que comprobar la semántica de cardinalidad mínima para conservarla.

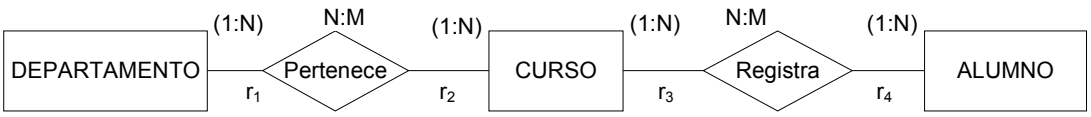


Figura 3.10. Un ejemplo de esquema conceptual para aplicar la propuesta de mejorar el rendimiento

La transformación de este esquema conceptual al modelo relacional contempla cinco tablas que se muestran en la tabla 3.4. Como las cardinalidades mínimas se convierten a participaciones opcionales podemos no existen disparadores activos

para controlar estas restricciones. Dentro de un estado de la BD, después de sucesivas operaciones de actualización, se analizan en cada tabla las tuplas que no cumplen las condiciones de la semántica de cardinalidad, véase la tabla 3.4.

Tabla	**Tuplas totales**	**Tuplas que no cumplen la semántica**			
DEPARTAMENTO	28	r_1	0		
CURSO	1244	r_2	4	r_3	6
ALUMNO	16538	r4	13		
Pertenece	3318				
Registra	114564				

Tabla 3.6. Las tablas y sus contenidos del ejemplo de mejorar el rendimiento

A la hora de aplicar esta propuesta se pueden ver claramente los elementos que sufren más pérdida de semántica y donde hay que intervenir para controlar esta semántica. Los resultados se muestran a continuación:

$$Q_{Pertenece} = (T_{Departo} + T_{Curso}) * (R^t_{Pertenece})^{-1}$$

$$= (0* 3318^{-1} + 4* 3318^{-1})*100 \cong 0.12\%$$

$$Q_{Registrar} = (T_{Curso} + T_{Alumnos}) * (R^t_{Registrar})^{-1}$$

$$= (6*114564^{-1} + 13*114564^{-1}) * 100 \cong 0.017\%$$

Esto significa que la pérdida de semántica de la tabla *Pertenece* es mayor que en la tabla *Registra*. Por ello, si existe un problema en el rendimiento del SGBD y el desarrollador tiene que elegir entre las dos tablas, *Pertenece* y *Registra*, se apoyará en estas medidas. Según los resultados se debería controlar la tabla *Pertenece* porque sufre mayor pérdida de semántica (0.12%) que la tabla *Registra* (0.017%), como se muestra en la figura 3.11.

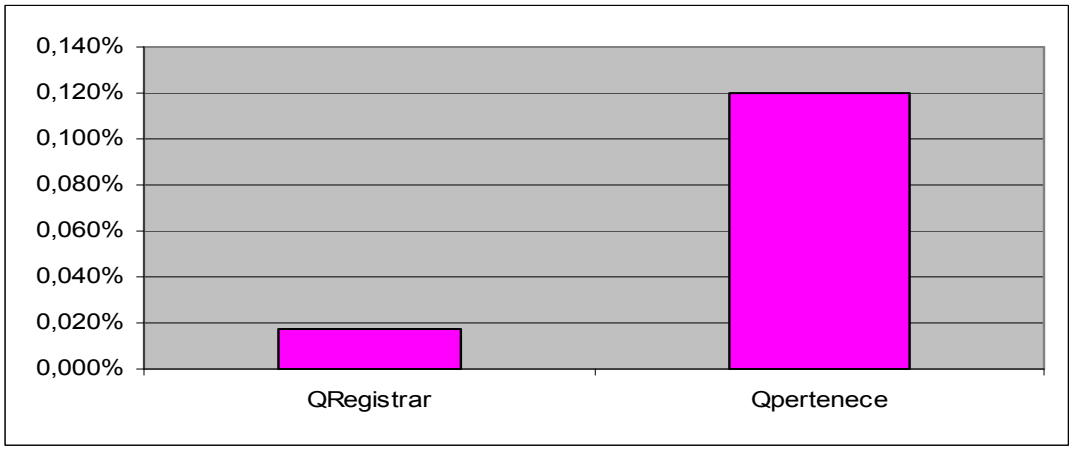


Figura 3.11. Los límites superiores de las pérdidas de semántica en cada tabla

Además, aplicando esta propuesta se puede saber exactamente en que rol se produce mayor pérdida de semántica. Para saberlo, se puede aplicar los siguientes cálculos:

$$q_{r1} = T_{r1} * (R^{t}_{Pertenece})^{-1} = 0 * 3318^{-1} * 100 = 0$$

$$q_{r2} = T_{r2} * (R^{t}_{Pertenece})^{-1} = 4 * 3318^{-1} * 100 = 0{,}12\%$$

$$q_{r3} = T_{r3} * (R^{t}_{Registrar})^{-1} = 6 * 114564^{-1} * 100 = 0{,}005\%$$

$$q_{r4} = T_{r4} * (R^{t}_{Registrar})^{-1} = 13 * 114564^{-1} * 100 = 0{,}011\%$$

Se observa claramente que la mayor pérdida de semántica se produce en el rol r_2. Es importante que se refuerce este rol más que los otros, figura 3.12.

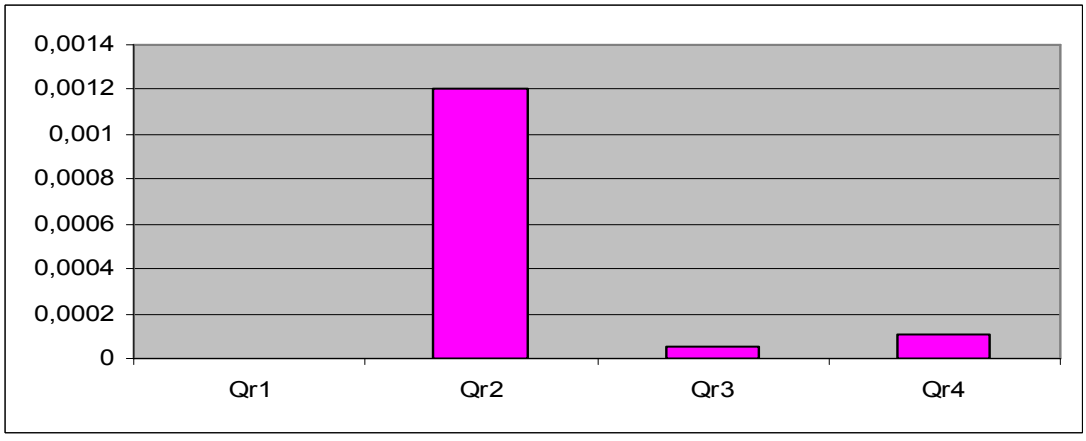


Figura 3.12. Los límites superiores de las pérdidas de semánticas en cada rol

Se puede aplicar otras métricas para obtener los resultados similares a los que hemos visto. Estas métricas se pueden calcular a través de la siguiente formula:

$n_{ri} = (R^t) * (E_j)^{-1}$

Donde;

n_{ri}: es la posibilidad que tiene una tupla de relacionase con otra en un rol de una interrelación.

R^t: el numero total de las tuplas en la tabla de interrelación *R*.

E_j: el número total de las tuplas en la tabla de entidad E_j.

Aplicando esta fórmula al mismo ejemplo en la tabla 3.6, se pueden obtener los siguientes resultados:

$n_{r1} = (R^t_{Pertenece}) * (E_{DEPARTAMENTO})^{-1} = 3318 * 28^{-1}$

$\cong 118$ (cada departamento tiene posibilidad de relacionarse con 118 Cursos).

$n_{r2} = (R^t_{Pertence}) * (E_{CURSO})^{-1} = 3318 * 1244^{-1}$

$\cong 3$ (cada curso tiene posibilidad de relacionarse con 3 departamentos).

$n_{r3} = (R^t_{Registrar}) * (E_{CURSO})^{-1} = 114564 * 1244^{-1}$

$\cong 92$ (cada curso tiene posibilidad de relacionarse con 92 alumnos).

$n_{r4} = (R^t_{Registrar}) * (E_{ALUMNO})^{-1} = 114564 * 16538^{-1}$

$\cong 7$ (cada alumno tiene posibilidad de relacionarse con 7 cursos).

Los resultados mostrados en la figura 3.13 confirman los resultados obtenidos anteriormente, donde Nr2 supone que cada curso tiene 3 posibilidades de relacionarse con departamento, mientras un departamento tiene 118 posibilidades de relacionase con un curso. Esto implica que insertar un curso sin mecanismos para controlar la semántica resulta más peligro que insertar un departamento, porque el curso tiene mayor posibilidad de relacionarse y se puede perder fácilmente.

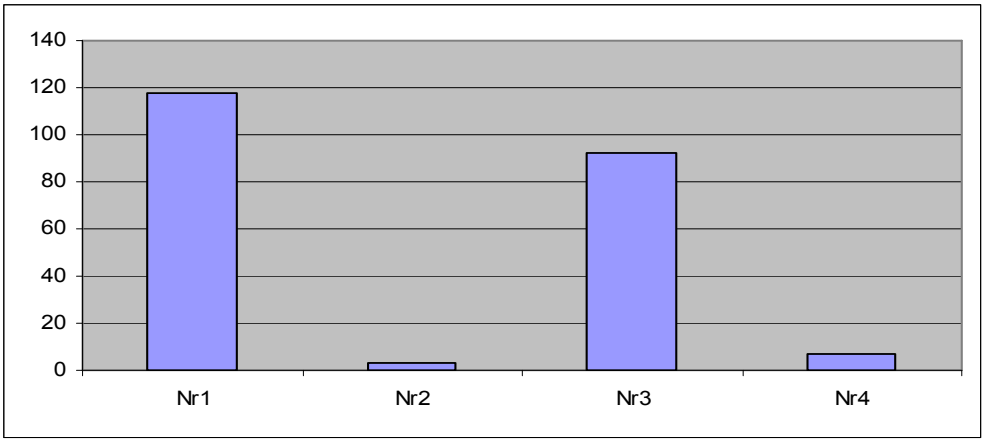


Figura 3.13. Las posibilidades de cada tupla para relacionase

Con estos cuantificadores para relajar las restricciones de cardinalidad de participación obligatoria terminamos la parte de la metodología para la conservación de la semántica a través de mecanismos activos. En el siguiente capítulo veremos las herramientas que se han implementado para dar soporte a esta metodología.

3.5.2 Tratamiento de la Inserción para Obligatoriedad de Participación

Cómo ya se ha mostrado en las secciones anteriores, uno de los problemas más complicados en el mantenimiento de las restricciones de integridad es la inserción de tuplas en las relaciones que provienen de entidades cuya participación en una determinada interrelación es obligatoria. Mantener la obligatoriedad de estas relaciones no ha sido tratado de una manera adecuada, ya que en las propuestas estudiadas se resuelve el problema interactuando con el usuario o a través de aplicaciones que acceden a la BD. Esto supone una repercusión muy fuerte en el rendimiento de la BD, al tener que estar esperando una respuesta por parte de aquellos usuarios que interactúan con ella, en el primer caso, y en el segundo, una implementación fuera de la BD para controlar esta restricción.

A continuación pasamos a enumerar algunos trabajos que han tratado de solucionar el problema de la inserción de distintos modos aunque con un denominador común, la actuación del usuario para completar la semántica requerida. En [Balaban y Shoval, 2002] se ha tratado el problema de conservar las restricciones de cardinalidad generando automáticamente métodos estructurales sobre un esquema EER. La propuesta depende de la interacción con los usuarios, es decir, preguntar a los usuarios para elegir el tipo de acción a realizar para reforzar una restricción. En la propuesta de [Pastor, 2004] se han utilizado transacciones seguras para controlar la semántica asociada a la restricción de cardinalidad. Ambas propuestas han tratado el tema de la inserción fuera del campo de las BDs activas. El problema de las BDs relacionales activas es que los disparadores actúan de forma inmediata después de la generación del evento. Por eso, en la propuesta de [Cuadra, 1999] se ha tratado el tema de la inserción utilizando dos funciones que sirven como interfaz con el usuario para la entrada de datos. Estas funciones son: PEDIR_PK (*tabla, clave_primaria*), que muestra un formulario al usuario y le solicita la clave primaria de la relación que se indica como parámetro, y PEDIR_RESTO (*tabla, resto_atributos*), que muestra un formulario al usuario y facilita todos los atributos de la relación, menos los que componen la clave primaria, y permite que el usuario deje en blanco aquellos atributos que no sean obligatorios. Por ahora no existe la posibilidad de implementar esta propuesta porque durante la ejecución de los disparadores no se permite la interrupción de éstos para facilitar información a los usuarios.

Esta es la principal motivación para introducir una propuesta de solución que no dependa del usuario. El objetivo de nuestra propuesta es que la solución sea transparente al usuario y no repercuta en la eficiencia de la BD. Se ha hecho una propuesta para solucionar este problema utilizando lo que se denominará *tablas auxiliares*. En la sección 3.4 se ha propuesto un método para relajar algunas restricciones de integridad siempre y cuando el diseñador lo vea necesario. Aplicando las métricas vistas en la sección anterior, también el diseñador puede determinar exactamente qué elemento sufre más pérdida de semántica. Para estos elementos, es indicada esta propuesta de solución.

El procedimiento a seguir, a partir de la detección de los elementos que más semántica contienen, se muestra en la figura 3.14.

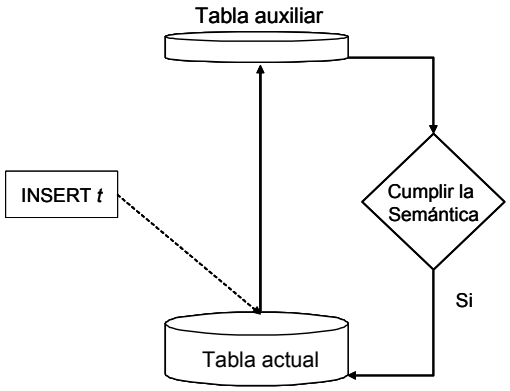


Figura 3.14: El mecanismo de la inserción según la propuesta de tablas auxiliares

Supongamos una interrelación binaria *R* con participación obligatoria (1,N)-(1,N), que se transforma a dos tablas *A*, *B* y una tabla para la interrelación *R*. Ahora se puede tratar la pérdida de semántica en la tabla *A* utilizando una tabla auxiliar *A'*. Esta tabla tiene la misma estructura de la tabla principal *A* pero está vacía. Cuando se inserta en *A* una tupla *t* se activa un disparador de inserción que desvía la inserción de *t* a la tabla *A*' ya que *t* todavía no tiene relación con una tupla de *B*. Por lo tanto, se mantendrá en las tablas auxiliares aquellas tuplas que todavía no están relacionadas con alguna tupla de B.

Los disparadores del SQL3 utilizados para controlar este mecanismo se muestran a continuación:

- El siguiente disparador se utiliza para desviar la inserción de la tupa que no cumple la semántica y también, borrar la misma tupla de la tabla principal.

```
CREATE TRIGGER Inserción_E1
AFTER INSERT ON A
FOR EACH ROW
BEGIN
INSERT INTO A'
   VALUES (:New.Pk_A, :New.restoatributos_A );
```

```
DELETE FROM A WHERE ID_A=:New.Pk_A;
END Inserción_A;
```

- El siguientes disparador se utiliza para recoger un tupla de la tabla auxiliar e insertar la en la tabla principal cuando se inserta su interrelación en *R*.

```
CREATE OR REPLACE TRIGGER Inserción_R
BEFORE INSERT ON R
BEGIN
IF ID_A IN
   (SELECT ID_A  FROM A' ) THEN
INSERT INTO A
VALUES (:ID_A, resto_atributos_A );
END IF;
IF ID_B IN
   (SELECT ID_B  FROM B' ) THEN
INSERT INTO B
VALUES (:ID_B, resto_atributos_B );
END IF;
DELETE FROM A' WHERE ID_A= :NEW.ID_A;
DELETE FROM B' WHERE ID_B= :NEW.ID_B;
END Inserción_R;
```

Estos disparadores serán un complemento para aquellas tablas que hayamos detectado importantes para su control. Estas tablas vendrán determinadas por lo aplicación de las métricas estudiadas en la sección anterior.

Capítulo 4

4. Herramienta de Ayuda para el Control de Restricciones de Integridad

4.1 Análisis de las Herramientas CASE Comerciales

El diseño de la BD contiene diferentes aspectos como son el modelado conceptual, lógico, físico, y el conocimiento o comprensión del dominio sobre el sistema de información que debe diseñarse [Miguel y Piattini, 1999]. Por eso, se necesita un gran esfuerzo para llevar a cabo la tarea de abstracción del mundo real y representarlo a través de un modelo de datos. Para la fase de diseño conceptual se han desarrollado diversos modelos [Elmasri y Navathe, 2004] pero por su importancia y alto contenido semántico cabe destacar en este trabajo el modelo ER.

En la última década, se han desarrollado diversos esfuerzos para dar el enfoque más sistemático a la resolución de problemas del modelado. Uno de tales esfuerzos ha sido la automatización del proceso de diseño de BD usando herramientas CASE (*Computer-Aided Software Engineering*). La contribución principal de las herramientas CASE en el desarrollo de las BD's es proporcionar mecanismos automáticos para desarrollar todas las fases de una metodología concreta. La razón para usar las herramientas CASE en el desarrollo de las BD surgió por la necesidad de compartir, manejar y reutililizar la información en los proyectos de desarrollo de software.

Aunque se han introducido estas herramientas CASE para soportar, facilitar y automatizar algunos ciclos de la vida de la BD entre las herramientas comerciales se advierte que existen algunas de ellas que adoptan enfoques más profesionales no pasan de ser simples utilidades para dibujar, disponiendo a lo sumo una BD para el almacenamiento no redundante de los objetos. Muchas de ellas no disponen siquiera de un soporte metodológico o no son lo suficientemente estrictas en su aplicación, con lo cual el diseñador no encuentra el camino correcto para realizar su tarea [Iglesias et al., 2002]. Otro estudio [Cuadra, 2003] ha señalado que tanto las herramientas comerciales como los prototipos de investigación, no se adecuan, generalmente, al propósito con que fueron diseñadas y por tanto no pueden emplearse para la construcción de sistemas de información integrados. Las deficiencias se resumen en:

- No abarcan el ciclo de diseño completo.
- No emplean técnicas avanzadas de representación.
- No exhiben comportamiento inteligente.
- No son metodológicamente estrictas.
- No contemplan diversas notaciones y metodologías, carecen de adaptabilidad.
- No favorecen la reutilización.

Así pues, en general todas las herramientas CASE comerciales, como (*Designer2000, ERwin, Power Designer*..., etc.) soportan la definición del esquema de la BD global, admitiendo representaciones gráficas del modelo conceptual, según las metodologías de diseño de BD que soportan. Se guarda toda la información sobre la estructura y el diseño de la BD en un repositorio que es una memoria compartida del diseño y las actividades de mantenimiento [Mokrane et al., 2000].

Por ejemplo, en la herramienta Designer de ORACLE [Oracle, 2005] solo aplica las reglas básicas de transformación con la consiguiente pérdida de semántica. El script generado no contemplará todas las especificaciones realizadas en el esquema conceptual. Y en el caso de existir una jerarquía se pregunta al diseñador que transformación quiere aplicar, si una sola tabla para el supertipo, solo las tablas para

los subtipos o la habitualmente adoptada por la mayoría de las herramientas CASE, una tabla para el supertipo y tantas tablas como subtipos. También, esta herramienta Designer/2000 proporciona un editor (*trigger definition*) que permite al usuario definir los disparadores diferentes.

En ERwin [CA, 2006] además de aplicar las reglas básicas de transformación con la consiguiente pérdida de semántica, la herramienta tiene la capacidad de generar disparadores para reforzar las acciones de integridad referenciales. Cualquier relación puede tener hasta seis disparadores; tres para el padre y tres para el hijo. Pero estos disparadores no pueden mantener las restricciones de cardinalidades al modificar estas relaciones [Al-Jumaily et al., (a) 2003]. Por ejemplo, si las restricciones de cardinalidades entre las entidades (AIRPORT-CITY) en la figura 4.2, son (1:N)-(1:N) y se mantiene las integridades entre estas dos utilizando las opciones en cascada del borrado y la actualización. Los disparadores generados del SGBD ORACLE de la relación AIRPORT que se muestran en la figura 4.1 son exactamente similares a los de la relación CITY. Aunque estos disparadores se pueden compilar sin existir errores de sintaxis la mayoría de los casos existen errores de ejecución que el diseñador debe depurar. También, estos disparadores se consideran redundantes porque realizan las mismas acciones que puedan realizar por las acciones de integridad referenciales.

En Rational Rose [Rational, 2003] también se puede transformar los elementos básico del esquema conceptual al esquema relacional con la consiguiente pérdida de semántica.

```
CREATE TRIGGER TD_AIRPORT
AFTER DELETE ON AIRPORT
FOR EACH ROW
BEGIN
/*AIRPORT R/2 AIRPORT_CITY ON PARENT DELETE CASCADE*/
    DELETE FROM AIRPORT_CITY
      WHERE
      AIRPORT_CITY.AIRPORTCODE =:OLD.AIRPORTCODE;
END;

CREATE TRIGGER TI_AIRPORT
AFTER INSERT ON AIRPORT
FOR EACH ROW
BEGIN
/* AIRPORT R/2 AIRPORT_CITY ON PARENT INSERT CASCADE*/
/* APPLICATION SPECIFIC. LEFT OUT INTENTIONALLY.*/
END;

CREATE TRIGGER TU_AIRPORT
AFTER UPDATE ON AIRPORT
FOR EACH ROW
BEGIN
/*AIRPORT R/2 AIRPORT_CITY ON PARENT UPDATE CASCADE*/
  IF :OLD.AIRPORTCODE <> :NEW.AIRPORTCODE
  THEN
  UPDATE AIRPORT_CITY
    SET AIRPORT_CITY.AIRPORTCODE = :NEW.AIRPORTCODE
    WHERE AIRPORT_CITY.AIRPORTCODE = :OLD.AIRPORTCODE;
  END IF;
END;
```

Figura 4.1. Disparadores de ORACLE generados en ERwin para la relación AIRPORT

Como resumen, se puede deducir que los modelos conceptuales que manejan las herramientas, en general, bastante pobres semánticamente, la transformación de esquemas conceptuales son incompletos. Solo aplican las reglas básicas de transformación y los scripts generados no son completos, ya que parte de la semántica se ha perdido.

En general las herramientas CASE comerciales no soportan la validación de todas las restricciones expresadas en el modelo conceptual, pues se puede especificar restricciones de integridad referencial desde la tabla referenciada a la que referencia y al contrario. Además, existen más opciones que las presentadas en el estándar SQL3. En ERwin, por ejemplo, estas restricciones se transforman directamente a disparadores. Los cuales controlaran la modificación y el borrado tanto en la tabla referenciada como en la que referencia.

Por ello, el objetivo principal de nuestro trabajo es cubrir unas fases más del ciclo de vida del desarrollo de BD (análisis, diseño e implementación). Enriqueciendo las reglas de transformación del esquema conceptual al esquema relacional controlando más semántica del UD.

4.2 Arquitectura de la Herramienta de Ayuda

En esta sección, se explica la integración de nuestras herramientas para desarrollar disparadores en la herramienta Racional Rose (RR), como se muestra en la figura 4.2. RR es una herramienta CASE que ha sido desarrollada por los creadores del lenguaje UML (Booch [Booch, 1994], Rumbaugh y Jacobson [Rumbaugh, 1991]). La cual cubre todo el ciclo de vida de los proyectos informáticos como concepción y formalización del modelo, construcción de los componentes, transición de los usuarios y certificación de las distintas fases del diseño. Hemos aplicado nuestra aproximación en la herramienta RR por su flexibilidad para agregar módulos *add-in* fácilmente. El *add-in* es un módulo que puede instalar menús, archivos de ayuda, ejecutables, servidores OLE, etc. [Rational, 2003].

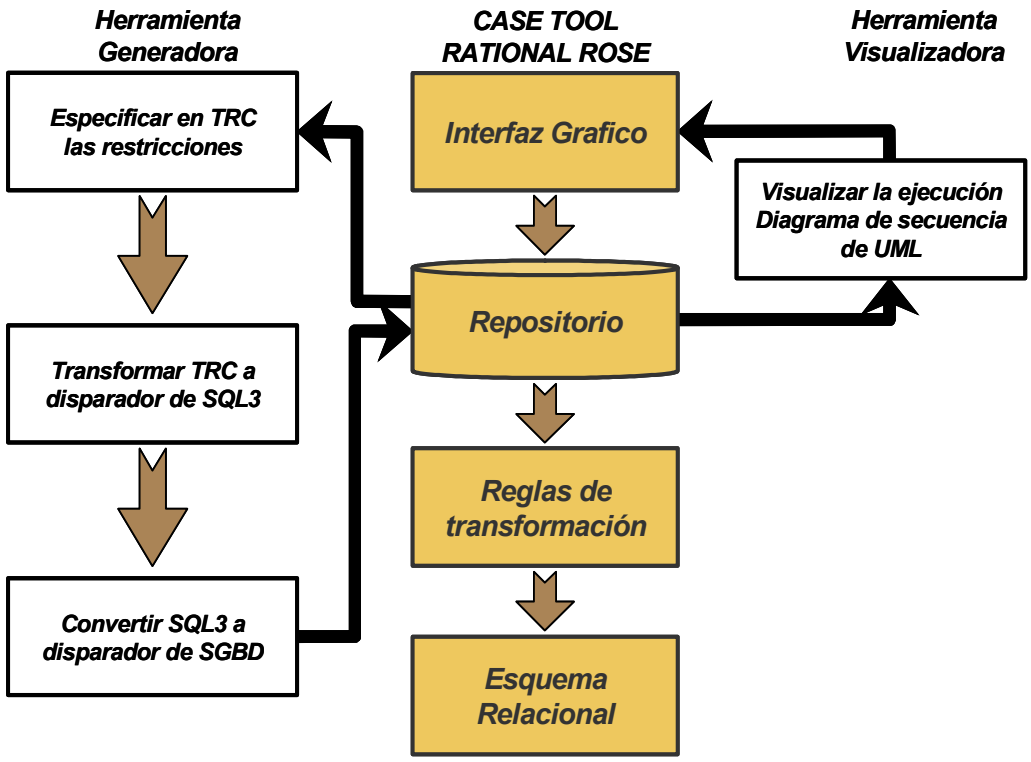


Figura 4.2. La integración de las herramientas en Racional Rose

Según la figura 4.2 la parte izquierda, muestra nuestra herramienta generadora integrada sobre el repositorio de la herramienta RR. A partir de un esquema conceptual que está guardada en el repositorio de RR, se aplicará el primer paso de nuestras herramientas (*Especificar en TRC las restricciones*) que se derive las especificaciones de las restricciones de integridad definidas en el esquema conceptual en cuestión al TRC. El segundo paso (*Transformar TRC a disparador de SQL3*) transforma las cláusulas TRC obtenidas del paso anterior a disparadores del estándar SQL3 aplicando las reglas de transformación explicadas en la sección 3.2 del capítulo anterior. El tercer paso transforma los disparadores del estándar SQL3 a disparadores de un SGBD determinado. En esta paso se detectan los problemas asociados con los disparadores del SGBD elegido para evitar la no terminación y los problemas específicos el mismo. Al final de este paso se asociarán los disparadores del SGBD en el repositorio en los elementos correspondientes del esquema en cuestión de RR como operaciones de (*Stereotype-Trigger*) para facilitar la modificación del contexto del disparador. Asociar los disparadores en el esquema tiene como objetivo principal permitir a los desarrolladores acceder al disparador para modificar o añadir a su código lo que se considere oportuno.

La parte derecha de la figura 4.2 muestra nuestra herramienta visualizadora que simula los disparadores integrados en el repositorio utilizando el diagrama de secuencia del UML. A través de los escenarios de los diagramas de secuencias los desarrolladores pueden comprobar el comportamiento de los disparadores de una forma estática detectando el problema de la no-terminación si existiera.

Las dos herramientas se han desarrollando utilizando el lenguaje *Basic Script Language* [Summit, 1996]. Se puede acceder a ellas a través del menú principal de RR y la opción Tool, como se muestra en la figura 3.5.

A continuación se explicarán estas herramientas con más detalle.

4.2.1 Módulo de Generación de Disparadores

Esta herramienta tiene como objetivo principal generar disparadores para conservar las restricciones de integridad asociadas a un esquema conceptual de RR. Se puede acceder a esta herramienta a través de la barra de herramienta TOOL y la opción "*Enforce Cardinality Constraints*", y como se muestra en la figura 4.3.

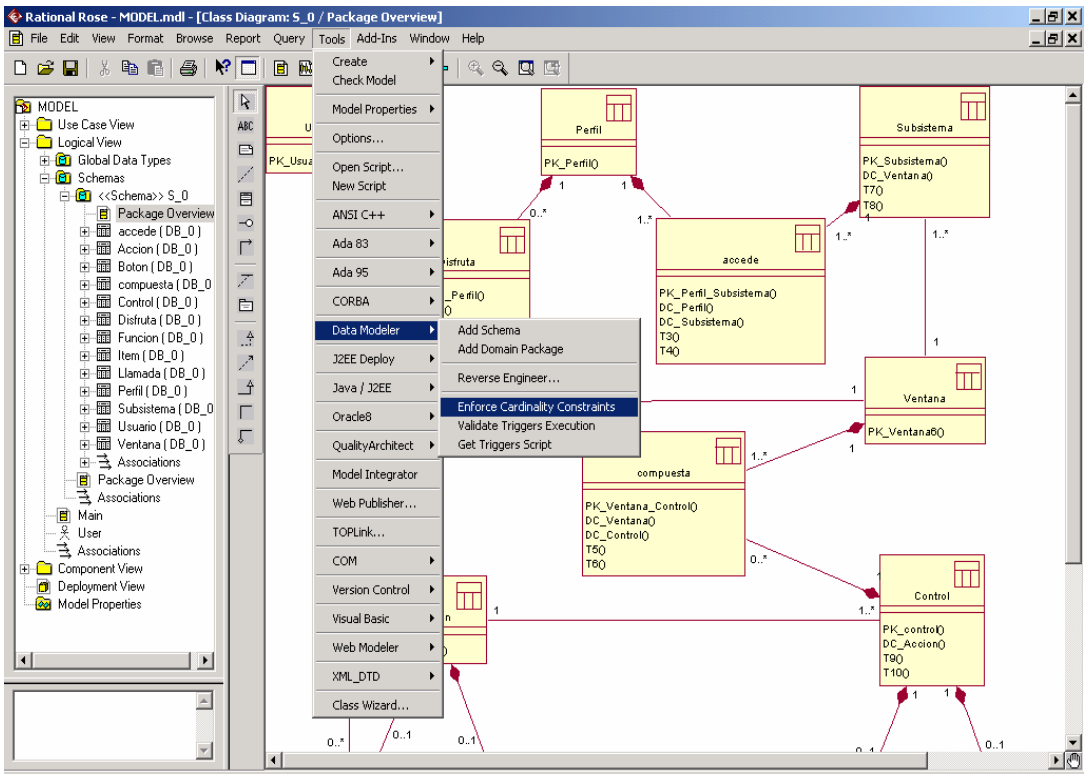


Figura 4.3. El acceso a las herramientas desarrolladas

1. Los objetivos específicos

Al tener un esquema conceptual creado en RR con sus restricciones establecidas, la herramienta generadora será capaz de realizar las tareas que se muestran a continuación:

1. La transformación automática de las restricciones de integridad (asociación, agregación, y generalización) especificadas en el esquema conceptual a cláusulas

del calculo relacional (TRC). A partir de aquí se convierten estas cláusulas a disparadores del SQL3 para posteriormente transformalos a disparadores (SQL Script) directamente ejecutables en la plataforma seleccionada. La herramienta generadora crea disparadores para el SGBD ORACLE teniendo en cuenta las características específicas del modelo de ejecución de este. Hemos elegido el sistema ORACLE por sus problemáticos disparadores, donde, además de la no-terminación que puede producir en la ejecución de estos, existe el problema de Tabla-Mutantes que restringen el uso de estos mecanismos.

2. Por la variedad de modos de ejecución de los disparadores en ORACLE la transformación de un disparador SQL3 a ORACLE puede generar más de un disparador. Por eso, es necesario usar paquetes y estructuras de datos auxiliares. Estos paquetes y estructuras de datos auxiliares tienen que ser generados por la herramienta y se incorporan dentro del script SQL.

3. Evitar algunos problemas en la ejecución de disparadores como la no-terminación y el problema específico del SGBD ORACLE de tablas mutantes.

4. Permitir a los desarrolladores de la BD realizar fácilmente las tareas de mantenimiento del esquema conceptual, como por ejemplo re-definir las restricciones de integridad, o quitar o/y añadir elementos al esquema en cuestión. Tal que la herramienta tendrá la capacidad de recrear y reformar los disparadores antiguos del esquema en cuestión según las novedades introducidas por los diseñadores.

5. Asociar los disparadores generados del esquema en cuestión como operaciones de UML agregadas. Esta asociación tiene como objetivo principal permitir a los desarrolladores acceder al contexto de cada disparador para modificarlo o añadir nuevas especificaciones a su código.

6. La optimización de los disparadores en la cual se optimiza principalmente el número de disparadores generados mejorando la eficacia de la BD. La optimización se realiza en la fase de transformación de los disparadores SQL3 a los correspondientes del SGBD utilizada, dependiendo totalmente de las

características específicas de esto. En esta fase se acoplan los disparadores que comparten los mismos atributos (Tabla, Evento, Tiempo, Granularidad) en solo un disparador.

7. Permitir a los desarrolladores elegir múltiples opciones para controlar, todo el esquema o una parte determinada de él, y el tipo de las actualizaciones (INSERT, DELETE, UPDATE).

2. El interfaz

Como se ha mostrado en la figura 4.3, se puede acceder al interfaz de la herramienta generadora a través de la opción '*Enforce Cardinality Constraints*'. Esta opción muestra el interfaz de la herramienta que se muestra en la figura 4.4.

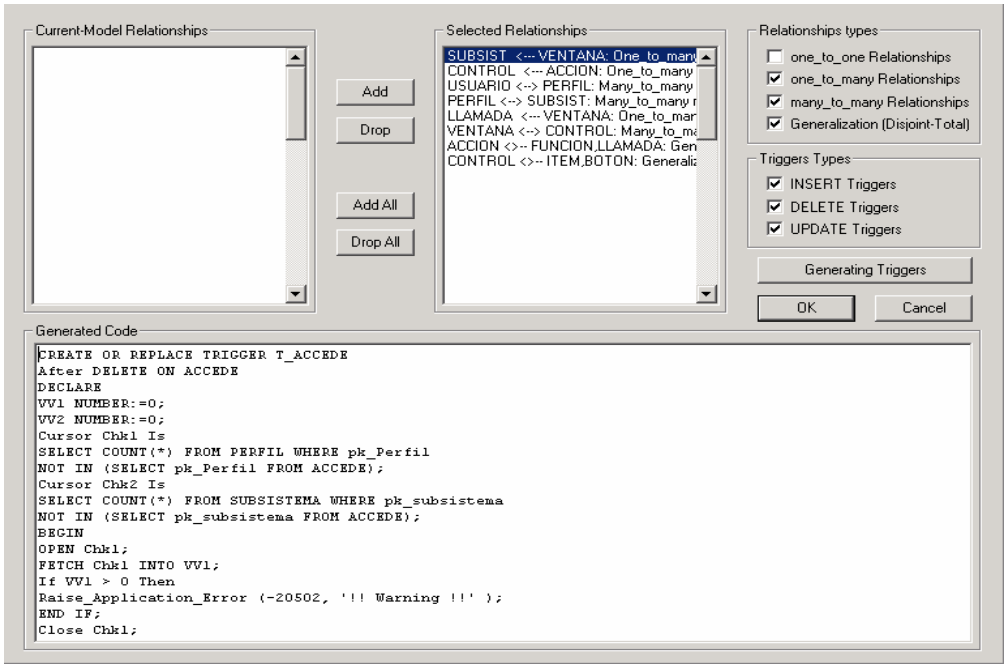


Figura 4.4. El interfaz de la herramienta generadora

La primera tarea de la herramienta es detectar todas las interrelaciones que existen en el esquema conceptual. Estas interrelaciones se presentan a los usuarios a través del interfaz en la zona '*Current Model Relationships*'. A partir de esto, el usuario según

sus necesidades puede elegir todas las interrelaciones '*Add All*' o parte de estas '*Add*'. Las interrelaciones elegidas para controlar sus restricciones aparecen en la zona '*Selected Relationships*'. Para distinguir entre estas interrelaciones, el interfaz utiliza símbolos para representarlas como, <-- para representar una agregación, <--> para representar una asociación, y <>-- para representar una generalización. La zona marcada por '*Relationships Types*' permite a los usuarios elegir las interrelaciones según sus tipos, como por ejemplo, todas las interrelaciones de tipo asociación. La zona marcada por '*Triggers Types*' permite a los usuarios generar disparadores sólo para los eventos especificados, como por ejemplo, disparadores para el borrado. Al pulsar el botón '*Generating Triggers*' el usuario puede obtener el código de los disparadores generados en la zona correspondiente, donde se puede guardar este código en un fichero de tipo SQL directamente ejecutable en el SGBD ORACLE.

Además de generar los disparadores la herramienta vuelve asociar estos disparadores como operaciones en el esquema. La figura 4.5 muestra una parte de un esquema con dos disparadores (T1_DISFRUTA y T2_DISFRUTA) asociados a la tabla Disfruta. El usuario puede acceder al código de los disparadores para realizar las modificaciones oportunas a través del interfaz que se muestra en la figura 4.6.

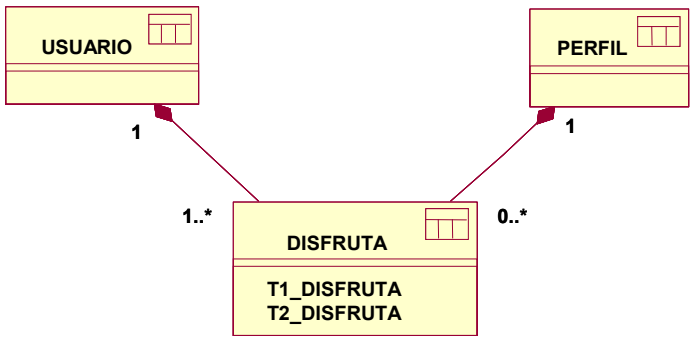


Figura 4.5. Visualización de los disparadores asociados a cada elemento del esquema

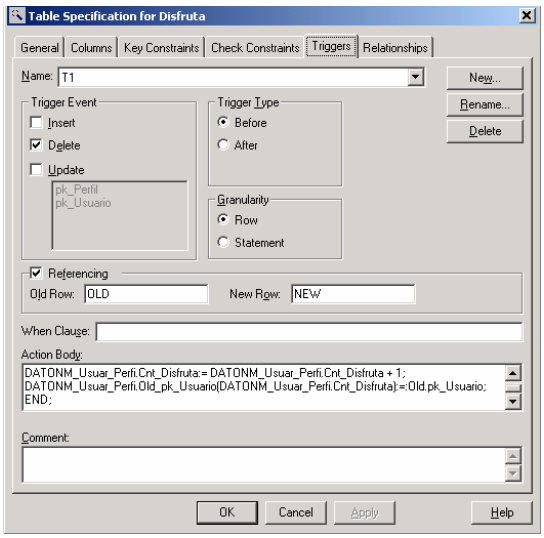


Figura 4.6. Interfaz para modificar el código de los disparadores

4.2.2 Módulo para la Visualización del Comportamiento de los Disparadores

Esta herramienta tiene como objetivo principal visualizar y validar el comportamiento activo de los disparadores para evitar su no-terminación cuando la BD se encuentre totalmente desarrollada y en funcionamiento. Por lo tanto, la herramienta realizará un análisis estático del comportamiento de los disparadores asociados a un esquema relacional dentro de un SGBD específico (Oracle 9i). También formará parte de un módulo *add-in* en RR. Se puede acceder a ella herramienta a través de la barra de herramienta TOOL y la opción "Validate Triggers Execution", como se muestra en la figura 4.3.

1. Los objetivos específicos

Para solucionar el problema de la invisibilidad en el orden de ejecución y el comportamiento de las reglas activas, proponemos diseñar e implementar una herramienta con los siguientes objetivos específicos:

(a) Utilizar los componentes del Diagrama de Secuencia de UML para modelar el

proceso de ejecución de las reglas, visualizando la activación en cascada de ellas. Con este diagrama se detectará el problema de la no terminación en la ejecución de los disparadores antes de implementarlos en un sistema gestor concreto. Para ello se utilizará el análisis sintáctico y estático de disparadores y el grafo de desencadenamiento (*Triggering Graph*).

(b) Mandar mensajes a los desarrolladores sobre el estado del comportamiento. Los tipos de mensajes serán informativos e indicarán al desarrollador si la ejecución de los disparadores termina o no con éxito.

2. Las notaciones del UML

El diagrama de secuencia de UML se considera muy útil para mostrar gráficamente las interacciones entre objetos, en un orden secuencial que describe, en cuestión de tiempo, las ocurrencias de esas interacciones. En el marco del UML de RR las notaciones del diagrama de secuencia que hemos utilizado para esta herramienta se muestran a continuación:

- *ScenarioDiagram*: Un escenario es un caso de uso que describe las ocurrencias secuenciales de los eventos durante la ejecución del sistema. El diagrama de escenario permite crear una representación visual de un escenario. La clase del diagrama del escenario de RR expone las propiedades y los métodos que permiten crear los diagramas y agregar o suprimir mensajes entre los objetos.

```
theScnDiagram As ScenarioDiagram
Set theScnDiagram = theCategory.AddScenarioDiagram
(DiagramName)
```

- *LifeLine*: Un diagrama de escenario contiene una o más líneas que representa la vida de los objetos que forman parte del escenario. Es una línea vertical discontinua y muestra el papel que juega el objeto a la largo del escenario que queremos modelar.

```
ObjInstance As ObjectInstance
Set    ObjInstance    =    theScnDiagram.AddInstance
(ClassName)
```

- *Message*: Los mensajes son métodos/operaciones que se requieren realizar. Un mensaje se representa en el diagrama de escenario a través de una flecha horizontal que sale de la línea de vida del objeto emisor, *theSender,* a la línea de vida del objeto receptor, *theReceiver*. Normalmente, estos mensajes llevan con ellos mismos las peticiones y las respuestas que interactúan entre los objetos del modelo. En el caso de que *theSender=theReceiver*, esto significa que el objeto *theSender* está mandando un mensaje a él mismo *MessageToSelf*. Cada mensaje está asociado con un número entero que muestra la posición relativa del mensaje en el diagrama. Por ejemplo, si el *theSequence = 3*, el mensaje es el tercer mensaje en el diagrama.

```
theSender, theReceiver As  ObjectInstance
theMessage As Message
theSequence As Integer
Set                    theMessage                    =
theScnDiagram.CreateMessage(MassageName,   theSender,
theReceiver, theSequence)
```

- *NoteView*: Las notas pueden contener cualquier tipo de información, hemos utilizado estas notas para avisar a los usuarios sobre los resultados de las verificaciones de escenarios. Por eso, hemos utilizado dos mensajes importantes a los usuarios, el primero es "***La terminación ha sido verificada***", se manda este mensaje cuando la comprobación del escenario es positiva. Mientras el segundo es "***Existe no-terminación en la ejecución***", este mensaje se manda a los usuarios cuando la comprobación del escenario detecta un estado de bucles en la ejecución, y le pide al usuario que intervenga para solucionar el problema.

```
AnItemView As RoseItemView
Set AnItemView = theScnDiagram.AddNoteView (theNote)
```

3. Representar el comportamiento

En general, el comportamiento de un disparador puede actuar de dos maneras. Primero actuar sobre su tabla asociada y segundo modificar otras tablas si es necesario según la semántica requerida. Por eso, para representar estos dos comportamientos en el diagrama de secuencia hemos utilizado dos tipos de mensajes de UML. El primer mensaje es *MessageToSelf* que sirve para representar el primer comportamiento del disparador, es decir, cada disparador se representa en el diagrama de escenario a través de un mensaje de un objeto (tabla) a él mismo. El otro comportamiento relacionado con las modificaciones de otras tablas se representa a través de utilizar un mensaje, *Message,* del cuerpo del disparador a la tabla modificada. Hemos utilizado este tipo de mensajes para representar específicamente las siguientes actuaciones:

- Las operaciones que se realizan por el usuario *Actor*; donde cada escenario comienza siempre por una de operación de actualización (INSERT, DELETE, UPDATE). Indicando sobre la flecha del mensaje el tipo de la operación realizada.
- Las operaciones de actualización incluidas dentro del cuerpo de un disparador y que actúan sobre otras tablas (*theSender* ≠ *theReceiver*). En este caso se indica sobre la flecha del mensaje el tipo de la operación realizada.
- Las acciones de integridad referencial (ON DELETE CASCADE y ON UPDATE CASCADE), donde cada una de estas acciones se representa como una operación realizada de la tabla referenciada a la tabla que referencia. En este caso, la flecha horizontal conecta la línea de vida del objeto referenciado con la línea de vida del objeto que referencia indicando sobre la flecha los símbolos (DC y UC) respectivamente para cada opción.

Para explicar bien la representación del comportamiento a través de la herramienta vamos a ver el esquema mostrado en la figura 4.7. El esquema tiene 8 disparadores agregados en sus tablas y la herramienta mostrará su comportamiento, demostrando que estos disparadores no generan ningún ciclo en la ejecución.

En este caso, lo que hace la herramienta visualizadora es realizar la verificación a través de una simulación de las posibles actualizaciones, donde cada tipo de actualización es mostrado como un escenario. Por eso, la herramienta genera 24 escenarios (8 Tablas * 3 tipos de actualización INSERT, DELETE, UPDATE) para verificar el esquema mostrado en figura 4.7.

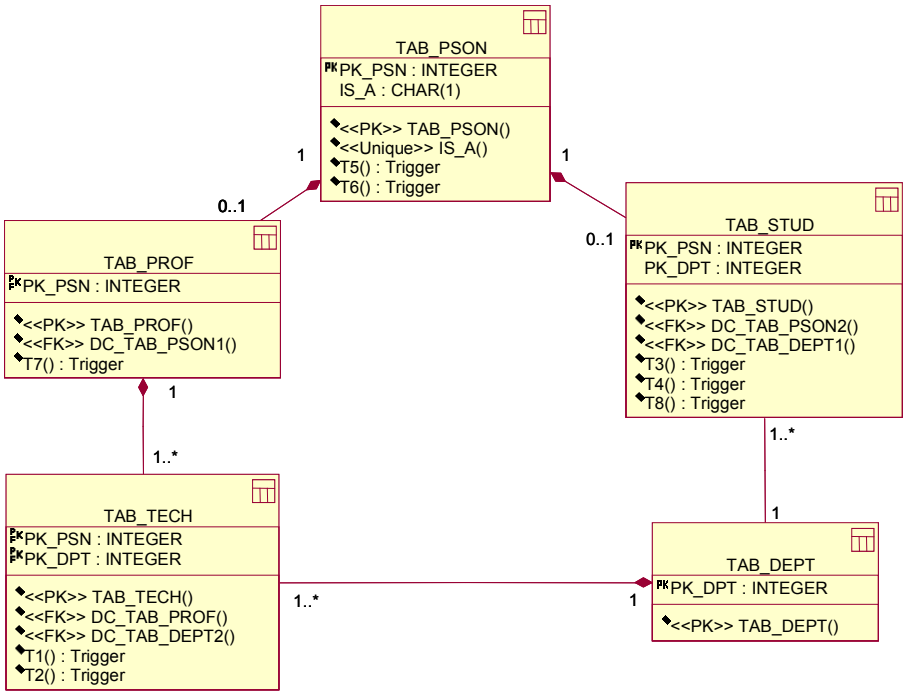


Figura 4.7. Un ejemplo de un esquema conceptual con sus disparadores

A continuación se muestran algunos escenarios como diagramas de secuencias para mostrar el comportamiento de estos disparadores.

Escenario #1:

En la figura 4.8, el actor *User* lanza una operación de borrado de la tabla TAB_TECH, como consecuencia de esta operación, se activan los disparadores T1, T2, asociados con TAB_TECH. Esta operación (1: DELETE) que aparece entre la línea de vida del actor y la línea de vida de TAB_TECH, indica el principio del escenario creado. Después de la primera operación, el tiempo de ejecución de estos disparadores depende de sus propiedades, es decir, que, la herramienta tiene en

cuenta el tiempo de ejecución y la granularidad de cada disparador, tal que se asigna al disparador T1 (B/R) el segundo orden, al ser de tipo BEFORE/ROW, mientras se asigna al disparador T2 (A/S) el tercer orden por ser de tipo AFTER/STATEMENT. El resultado final de este escenario se muestra en la figura 4.8, donde aparece un mensaje indicando al usuario que la terminación del actual escenario ha sido verificada correctamente.

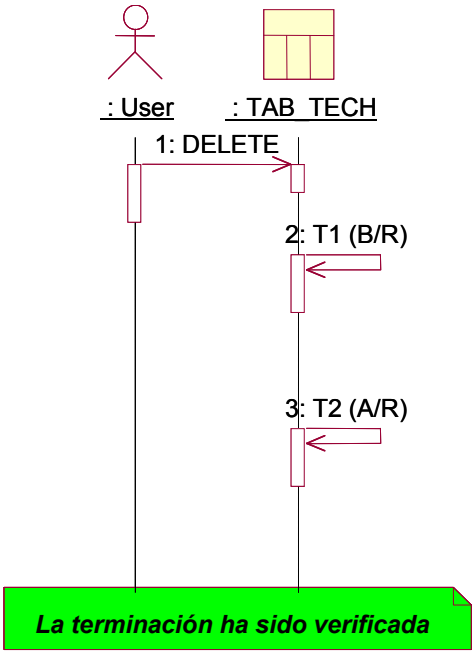


Figura 4.8. El ejemplo del escenario #1

Escenario #2:

En este escenario, figura 4.9 se muestra otro diagrama de secuencia más complicado ya que actúan todos los disparadores creados en el esquema. Además de esto, en este escenario interactúan otras tres operaciones por la secuencia del borrado en cascada. El esquema contiene una jerarquía con dos subtipos, la definición de esta jerarquía se muestra a continuación:

```
CREATE TABLE TAB_PSON (PK_PSN, ….);
CREATE TABLE TAB_DEPT (PK_DPT, ….);
CREATE TABLE TAB_PROF (PK_PSN, …..,
```

```
    CONSTRAINT CFK1 FOREIGN KEY (PK_PSN)
    REFERENCES TAB_PSON ON DELETE CASCADE);
CREATE TABLE TAB_STUD (PK_PSN, PK_DPT, …..,
    CONSTRAINT CFK2 FOREIGN KEY (PK_PSN)
    REFERENCES TAB_PSON ON DELETE CASCADE);
CREATE TABLE TAB_TECH (PK_PSN, PK_DPT,….     );
    CONSTRAINT CFK3 FOREIGN KEY (PK_PSN)
    REFERENCES TAB_PSON ON DELETE CASCADE);
    CONSTRAINT CFK4 FOREIGN KEY (PK_DPT)
    REFERENCES TAB_DEPT ON DELETE CASCADE);
```

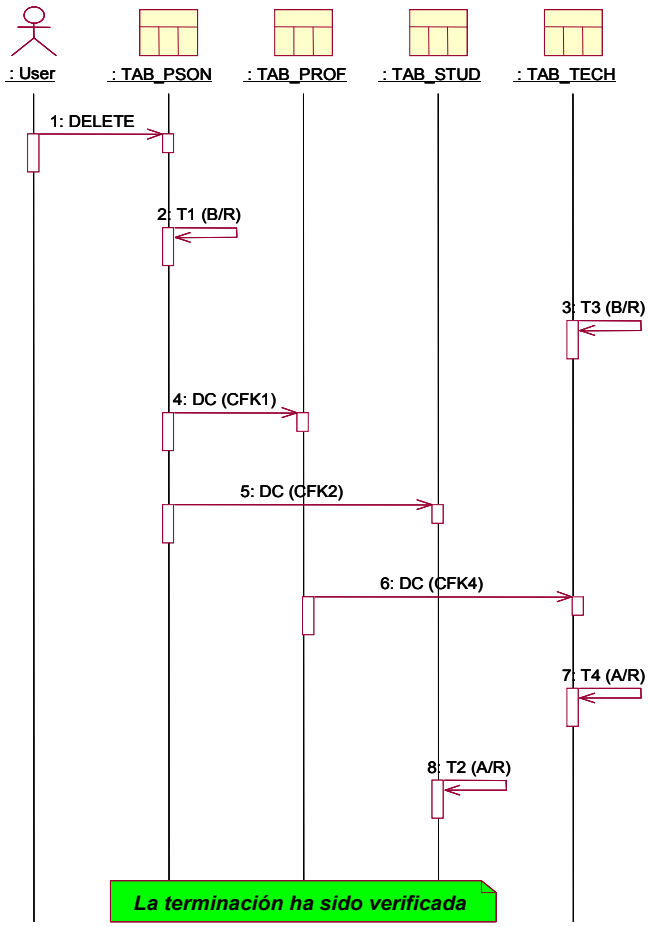


Figura 4.9. El ejemplo del escenario #2

Según esta definición existen cuatro acciones de integridad referencial (CFK1, CFK2, CFK3, CFK4) que generan cuatro operaciones como mensajes entre las tablas

referenciadas y las tablas que referencian. Ahora, cuando se borra una persona se borran en cascada todos los subtipos relacionados (estudiantes y profesores). Además de la operación principal (1: DELETE), las dos operaciones 4: DC (CFK1), 5: DC (CFK2) se representan en el diagrama de secuencia del escenario como flechas de la tabla referenciada a la que referencia. La otra acción de integridad referencial que aparece en el diagrama 6: DC (CFK4) es la consecuencia del borrado en cascada entre la tabla TAB_PROF y TAB_TECH.

El escenario #2 muestra que cuando se borra una persona actúa la operación (1: DELETE) que activa el disparador 2: T1 (B/R) sobre la tabla TAB_PSON, luego el disparador 3: T3 (B/R) sobre la tabla TAB_TECH. T1 y T3 son disparadores sin consecuencia, es decir, que no tienen ninguna operación de modificación dentro de sus cuerpos. Cuando termina la ejecución de T1 y T3, se activa las operaciones de los borrados en cascada 4: DC (CFK1), 5: DC(CFK2), 6: DC(CFK4). Luego se activan otros dos nuevos disparadores 7: T4 (A/R) y 8: T2 (A/R). Cuando termina la ejecución de estos dos últimos no queda ninguna operación en la cola, por eso, el escenario se verifica correctamente y la terminación se confirma.

Escenario #3:

En este escenario se va a redefinir el disparador T4 añadiendo una sentencia de modificación según se muestra a continuación:

```
CREATE TRIGGER T4
AFTER DELETE FROM TAB_TECH
FOR EACH ROW
BEGIN ATOMIC
.....................
DELETE FROM TAB_PSON
.....................
END;
```

El nuevo escenario será similar al anterior escenario hasta llegar a la ejecución del disparador T4, donde se cambia el escenario detectando un caso de no-terminación, como se muestra en la figura 4.10. La operación (8: DELETE) representa la ejecución de la sentencia añadida. Como consecuencia de la ejecución de esta

sentencia aparece el disparador 9: T1 (B/R) otra vez en el diagrama. La aparición dos veces de un disparador en el diagrama forma un ciclo que repite la misma secuencia de operaciones formado un estado de no terminación.

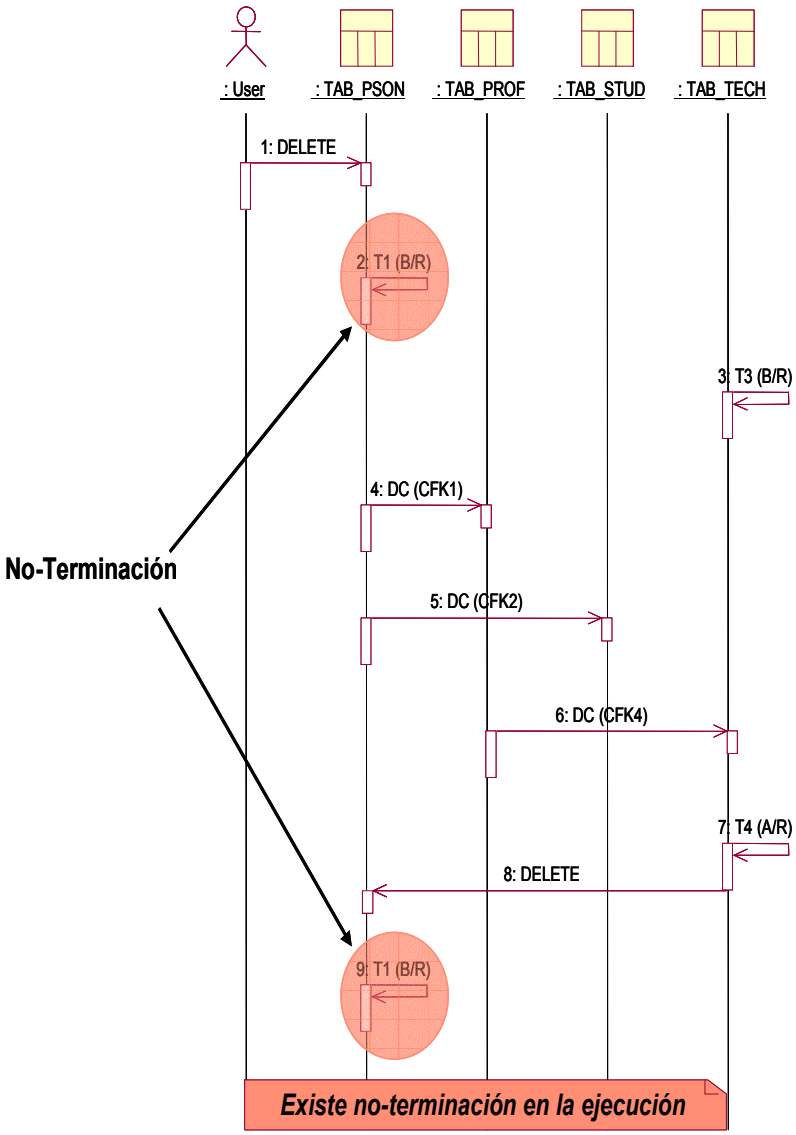


Figura 4.10. El ejemplo del escenario #3

Con la muestra de cómo actúa la herramienta visualizadora que tiene como objetivo principal visualizar el comportamiento activo y las interacciones entre los disparadores y las acciones de integridad referenciales.

Capítulo 5

5. Validación y Verificación de la Propuesta

5.1 Introducción

Para comprobar la propuesta de esta tesis doctoral nos basaremos en dos principios básicos dentro de la disciplina de las pruebas del software: la verificación y la validación. Por ello, en este capítulo se presentará cuatro experimentos que nos han permitido comprobar de forma empírica cuales son las repercusiones de nuestra contribución. Validar una propuesta supone probarla y verificar todas las fases y los elementos empleados para construir la misma y comprobar que efectivamente se ha solucionado los problemas asociados.

En el segundo capítulo se han abordado los problemas de la no-terminación para el conjunto de disparadores que controlan la semántica no contemplada en los mecanismos declarativos relacionales y también aquellos problemas derivados del SGBD en la implementación de los mismos. Por tanto, el objetivo principal en este capítulo es probar que el código generado por los disparadores cumplan las especificaciones para las que han sido definidas. En este capítulo se presentará algunas técnicas de pruebas del software para la verificación y la validación (V & V) de los disparadores con el propósito de comprobar que no existen errores, ni producen inconsistencia en los resultados y satisfacen los requisitos de los usuarios.

En general, la verificación es el proceso de evaluar un sistema o uno de sus componentes, durante o al final del proceso de desarrollo, para asegurar que el software implementa correctamente una función específica. Mientras, la validación significa la comprobación de que un programa puesto en ejecución resuelve las expectativas del cliente. En la verificación se intenta responder a la pregunta, ¿estamos construyendo el producto correctamente?. En la validación se intenta responder a la pregunta, ¿estamos construyendo el producto correcto? [Sommerville, 2005].

Existen dos técnicas para realizar el proceso de V & V. La primera técnica inspecciona el software producido, es decir, se basa en analizar y comprobar las representaciones del software tales como el documento de requisitos, los diagramas de diseño y el código fuente del software. Las inspecciones de software son técnicas estáticas que se pueden realizar a través de un repaso detallado y formal del software a través de grupos limitados de desarrolladores. Posteriormente a este análisis detallado, estos grupos se reúnen para estudiar y tomar decisiones sobre los resultados de las inspecciones. La segunda técnica (*Software Testing*) implica ejecutar el software en cuestión con datos de pruebas. Estas técnicas son denominadas dinámicas y son útiles para observar y examinar el comportamiento del producto software en su entorno operacional.

Aunque las técnicas estáticas pueden comprobar el cumplimiento del software frente a sus especificaciones, estas técnicas no pueden demostrar que el software es operacionalmente útil, tampoco se pueden utilizar para comprobar las propiedades asociadas con la ejecución del software, como por ejemplo, el rendimiento [Sommerville, 2005]. Esta es una de las razones por la que en nuestra propuesta se han aplicado técnicas dinámicas [Pressman, 2003]. A continuación se detallan las pruebas que se van a emplear en este trabajo:

- Pruebas funcionales o de caja negra consiste en estudiar la especificación de las funciones del software. Tienen como objetivo principal asegurar que el software examinado cumple la funcionalidad requerida a través de estudiar las entradas y

las salidas para derivar todos los casos posibles a probar. Hemos realizado dos técnicas de las pruebas funcionales:

- Técnica de partición equivalente tiene como objetivo identificar todas las clases de equivalencia y los casos de pruebas correspondientes a cada clase. Para que a continuación se cubran todos estos casos de pruebas tanto válidas como inválidas.
- Técnica de aceptación consiste comprobar que software construido a medida para un cliente tiene que satisfacer los requisitos definidos por el cliente. Normalmente, esta técnica se realiza con datos proporcionados por los usuarios porque son más parecidos a los datos reales. Puede que esta técnica revele errores y omisiones en la definición de los requisitos del sistema [Sommerville, 2005]. Esta técnica pueden clasificarse en alfa o beta según si es un usuario o varios los que realizan la prueba dentro del entorno de desarrollo. Nosotros aplicaremos la técnica de tipo alfa por ser suficiente para comprobar la validez de nuestra propuesta. Y vendrán a reforzar las pruebas realizadas con la técnica de partición equivalente.

- Prueba del camino básico es una prueba estructural que se deriva a partir del conocimiento de la estructura e implementación del software. Tiene como objetivo examinar los caminos o ramificaciones posibles del flujo de ejecución de una estructura de programa. Aplicado a nuestra propuesta se verificará que las estructuras de los disparadores generados por nuestro módulo visto en la sección 4.3.1 están correctamente construidas. Para realizar esta prueba tendremos que diseñar unos casos de pruebas que garanticen la ejecución, por lo menos una vez, de cada sentencia del software de la que se compone nuestro conjunto de disparadores.
- Pruebas de rendimiento: en general, este tipo de pruebas están diseñadas para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado [Pressman, 2003]. El objetivo principal de esta prueba es realizar una sobrecarga del sistema para provocar alguna manifestación de

defectos o errores que no son probables en un funcionamiento normal. En nuestro caso, el objetivo de utilizar estas pruebas es monitorizar el rendimiento de una BD cargada por disparadores cuando se realizan actualizaciones aleatorias masivas en sus elementos. Además, se realizará una comparación entre el rendimiento, dentro del mismo SGBD, de disparadores y procedimientos almacenados. Este tipo de comparación para monitorizar el rendimiento ha sido propuesto en [Paton, 1998] porque actualmente las limitaciones del lenguaje de los disparadores y las limitaciones en el modelo de ejecución de los SGBD relacionales, no permiten establecer un punto de referencia que permita a los desarrolladores determinar cuando los disparadores realmente perjudican al rendimiento del SGBD.

Para realizar la validación y la verificación de nuestra herramienta se ha desarrollado un escenario de pruebas. Este escenario contiene una BD que contempla aquellas restricciones de integridad más complejas de conservar durante la transformación a un esquema relacional, es decir, las restricciones que fueron estudiadas con profundidad en la sección 3.1.2. Aunque este escenario no contempla todas las posibilidades, cualquier otro escenario que contemple otra combinación no cambiará significativamente los resultados de nuestra experimentación ya que la semántica de los disparadores será muy parecida.

A continuación se presenta primero la BD del experimento que se va a utilizar para realizar las pruebas, para posteriormente mostrar los cuatro tipos de pruebas que se han aplicado para la verificación y la validación de nuestra propuesta. Por último, se presenta un análisis global y resumen de los resultados obtenidos.

5.2 Descripción del Escenario para Realizar las Pruebas

Para hacer las pruebas se ha implementado una BD experimental genérica. Se ha escogido esta BD como escenario de las pruebas porque en ella se contemplan todas las restricciones de integridad estudiadas en este trabajo. La BD del experimento contiene los siguientes tipos de interrelaciones (véase la figura 5.1);

- Dos interrelaciones binarias de tipo N:M, denominadas R2 y R4, que se transforman a tablas en el modelo relacional, aplicando las reglas de transformación.
- Una interrelación binaria R1 de tipo 1:N. La transformación de esta interrelación se realiza mediante la propagación de la clave primaria del tipo de entidad referenciada al tipo de entidad que referencia.
- Una jerarquía total y exclusiva R3 que se transforma optando por crear una tabla diferente para el supertipo y para cada uno de los subtipos.

Las especificaciones de las cardinalidades mínimas son también distintas en la BD de pruebas. Es decir, todas las interrelaciones contemplan las diferentes combinaciones en las restricciones cardinalidad. Aunque se han estudiado en este trabajo las cardinalidad máximas conocidas y las cardinalidades mínimas mayores que uno, no se han podido implementar porque la herramienta Racional Rose no permite especificar estas cardinalidades en el esquema conceptual.

Cualquier otro escenario que refleje otro domino no cambiará significativamente los resultados de nuestra experimentación ya que la semántica de los disparadores será muy parecida. Por ejemplo, una interrelación binaria de tipo N:M o 1:N es tratada de una manera similar en cualquier dominio ya que la única diferencia viene dada por la cardinalidad mínima, es decir, si la participación es obligatoria u opcional. En el caso de las jerarquías, la semántica más delicada viene recogida por la totalidad y la

exclusividad de la misma. Como ya vimos en la sección 3.1.2 se controlan de una manera similar en todos los dominios. Por eso, los disparadores generados para controlar cualquier elemento de nuestro escenario pueden ser reutilizados para controlar el mismo elemento en otro escenario, después de modificar los parámetros específicos que reflejan la semántica del elemento.

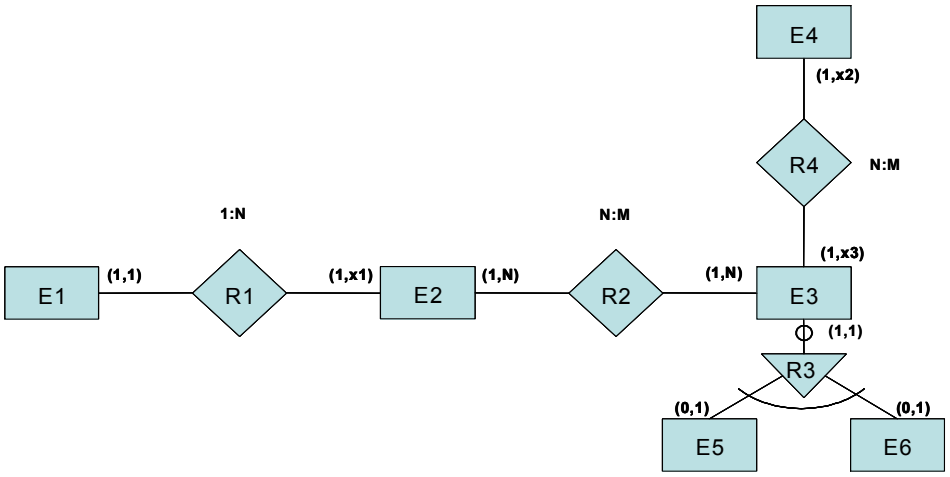


Figura 5.1. El modelo ER de la BD experimentada

Aplicando las reglas de transformación de la metodología de desarrollo de BD al esquema mostrado en la figura 5.1 se obtiene el esquema relacional presentado en la figura 5.2 [Teorey, 1999] [Elmasri y Navathe, 2004].

La transformación del esquema presentado en la figura 5.1 a un esquema soportado por el modelo de (Racional Rose Data Model) aparece en la figura 5.3. Cada interrelación binaria de tipo N:M se transforma en una tabla con dos interrelaciones de identificación que conectan las dos tablas que participan en esta interrelación. En el caso de la interrelación binaria de tipo 1:N, se transforma agregando la clave primaria de la entidad referenciada en la tabla de la entidad que referencia, creando una interrelación de no-identificación que relaciona las dos tablas. Y por último, la transformación de las jerarquías se representa con tres tablas, dos tablas para los subtipos relacionadas con la tabla del supertipo a través de dos interrelaciones en identificación.

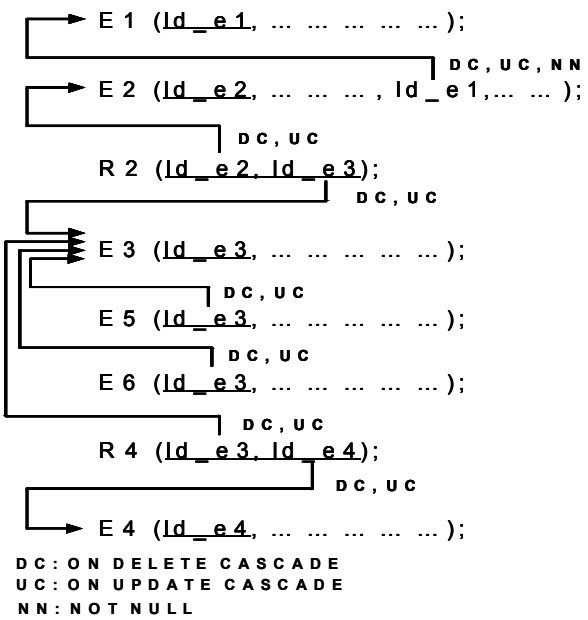


Figura 5.2. Transformación de la BD del experimento al modelo relacional.

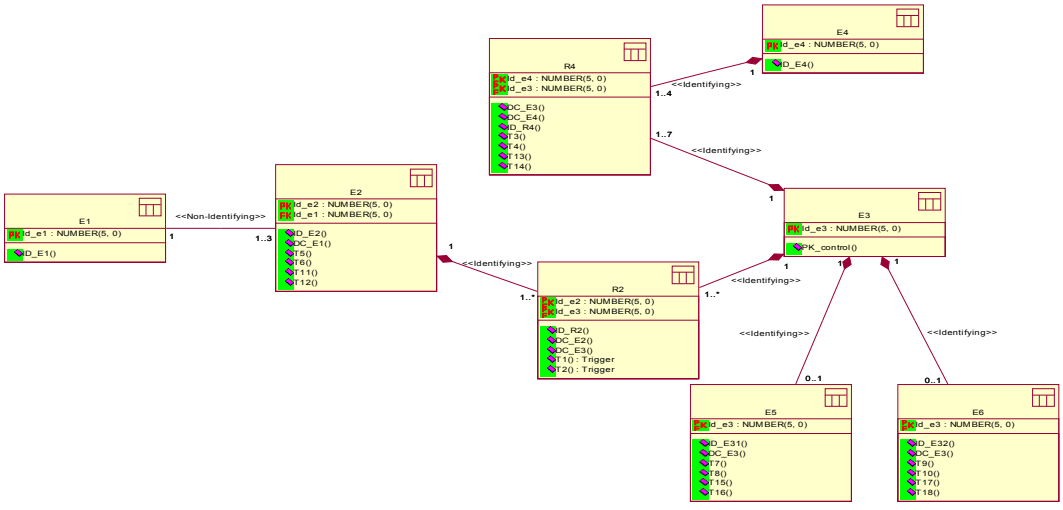


Figura 5.3. El esquema de la BD del experimento en Racional Rose

Para realizar la validación y la verificación de nuestra propuesta hemos sometido, el esquema de la BD de experimento mostrada en la figura 5.3, a nuestras herramientas para generar los disparadores necesarios para controlar las restricciones de integridad estudiadas en este trabajo. La tabla 5.1 muestra 18 disparadores generados y las tablas a las que están asociados. Estos disparadores forman 9 secuencias, una

secuencia puede contener uno ó más disparadores que se activan por una operación de actualización determinada. Por ejemplo, en la tabla E2 existen 4 disparadores que forman dos secuencias {T5, T6} y {T11, T12}. Los disparadores T5 y T6 se utilizan para controlar el borrado y la modificación de la clave ajena E1_e1. Con ellos se conserva la semántica de cardinalidad mínima que describe la participación del tipo de entidad E2 en la interrelación R1 (Cmin(E2→R1)). Los disparadores T11 y t12 controlan la inserción en la tabla E2 para que conserve la semántica de cardinalidad máxima del tipo de entidad E2 en R1 (Cmax(E2→R1)). Los disparadores T7, T8, T9, y T10 controlan la totalidad de la jerarquía mientras los disparadores T15, T6, T17, y T18 controlan la exclusividad de la jerarquía.

Nº	Tabla	Secuencias de Disparador	Evento	Tipo de pérdida de Semántica
1	E2	T5, T6	DELETE UPDATE	Cmin(E2→R1)
2	E2	T11, T12	INSERT	Cmax(E2→R1)
3	R2	T1, T2	DELETE UPDATE	Cmin(E2→R2) Cmin(E3→R2)
4	R4	T3, T4	DELETE UPDATE	Cmin(E3→R4) Cmin(E4→R4)
5	R4	T13, T4	INSERT	Cmax(E3→R4) Cmax(E4→R4)
6	E5	T7, T8	DELETE	Totalidad de E3→E5
7	E6	T9, T10	DELETE	Totalidad de E3→E6
8	E5	T15, T16	INSERT	Exclusividad de E3→E5
9	E6	T17, T18	INSERT	Exclusividad de E3→E5

Tabla 5.1. Las restricciones que vamos a controlar para el experimento.

A continuación se presentarán cuatro tipos de pruebas complementarias realizadas para la validación y la verificación de los disparadores generados por nuestra propuesta utilizando el SGBD ORACLE.

5.3 Pruebas Funcionales

Las pruebas funcionales, también llamadas cajas negras, examinan el software sin preocuparse del contenido y la estructura del mismo. También, se llaman pruebas de entrada/salida al considerar que el examinador tiene la responsabilidad de proporcionar solo los datos de entrada, y estudiar los resultados de salida, sin la necesidad de conocer los detalles del software. El objetivo principal de estas pruebas es garantizar que el software en cuestión cumple la funcionalidad especificada. En particular, la funcionalidad requerida en nuestro caso es garantizar que el conjunto de los disparadores generados por nuestra herramienta controlan las restricciones de integridad de la BD. Una prueba exhaustiva de caja negra es impracticable porque no se pueden ejecutar todas las posibilidades de funcionamiento y todas las combinaciones de entrada y salida, por eso, se deben buscar criterios que permitan elegir un subconjunto de casos cuya ejecución aporte una cierta confianza en detectar los posibles defectos del software [Piattini, et al., 1996]. En este contexto, es recomendable conseguir una cobertura alta de todos los casos de pruebas posibles del software.

En general, el diseño de pruebas de cajas negras se basa en la definición de clases de equivalencia. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para unas determinadas condiciones de entradas [Pressman, 2003]. Normalmente, las condiciones de entradas son los valores numéricos, un rango de valores, un valor booleano, etc. A continuación se presentan algunas reglas generales para identificar las clases de equivalencia:

- Si la condición de entrada requiere un cierto rango, debemos especificar tres clases de equivalencia: una válida y dos inválidas. Por ejemplo, para un rango (1, 2, 3,…100) $\Rightarrow$ clases de equivalencia son 0, 50, 101.

- Si la condición de entrada requiere un valor específico, debemos definir tres clases de equivalencia: una válida y dos inválidas. Por ejemplo, para un valor (9) $\Rightarrow$ clases de equivalencia son 8, 9, 10
- Si la condición de entrada requiere un valor de un miembro de un conjunto, debemos especificar dos clases de equivalencia: una válida y una inválida. Por ejemplo, para un conjunto de valores (‘‘, ‘talonario’, ‘movimiento’) $\Rightarrow$ clases de equivalencia son ‘talonario’, ‘xyz’
- Si la condición de entrada es booleana, debemos definir dos clases de equivalencia: Verdadero o Falso.

Una vez identificadas las clases de equivalencia se procede a identificar los casos de pruebas correspondientes asignando un valor único a cada clase. Se espera, al final de las pruebas, cubrir todas las clases de equivalencia tanto válidas como inválidas.

Hemos realizado las pruebas sobre la BD del experimento mostrada en la figura 5.3. El estado inicial de todas sus tuplas cumplen las condiciones de las restricciones de integridad. Para controlar estas restricciones hemos generado un conjunto de disparadores que actúan como vigilantes (véase tabla 5.1). La ejecución de estos disparadores no produce ningún ciclo. A continuación se presentan los pasos seguidos para realizar las pruebas asociadas a nuestra propuesta.

1. Identificación de las clases de equivalencia:

En este paso se identifican primero todas las condiciones de entradas para posteriormente agruparlas en clases de equivalencia necesarias. En función de las entradas esperadas sobre la BD del experimento podemos identificar dos condiciones:

- Las operaciones lanzadas para realizar un tipo de actualización sobre la BD del experimento. Estas operaciones proporcionan tres clases de equivalencia válidas: INSERT, DELETE, UPDATE.

- Los valores de las claves asociadas a dichas operaciones. En el contexto de nuestra propuesta, las entradas asociadas a las operaciones de actualización llevan consigo las claves primarias y las claves ajenas. Respecto a la BD del experimento estas claves son: (E1.Id_e1, E2.Id_e1, E2.Id_e2, R2.Id_e2, R2.Id_e3, E3.Id_e3, E5.Id_e3, E6.Id_e3, R4.Id_e3, R4.Id_e4, E4.Id_e4). Por lo tanto, cada una de estas entradas proporciona una clase de equivalencia.

Desde el punto de vista del SGBD, las condiciones de entradas siempre tienen que ser válidas, es decir, por ejemplo, si insertamos una tupla en una tabla, el SGBD comprueba primero la validez de la tupla insertada con los requisitos establecidos. Por eso, nunca se permite insertar un valor inválido en un campo. Además, cuando se quiere borrar una tupla de una tabla, el SGBD comprueba primero que dicha tupla existe en la tabla correspondiente, antes de realizar la operación. Pero, desde el punto de vista de nuestra propuesta y en función de la consistencia de los datos se puede identificar dos tipos de operaciones: válida e invalida. Una operación válida especifica que la BD acepta esta operación y es válida en función de la semántica esperada. Mientras, una operación inválida puede ser la misma de antes, pero con otros valores que la BD acepta, pero será inválida en función de la semántica esperada. Por ejemplo, insertar (x, y) en una tabla T puede resultar válido para la semántica mientras que insertar (x, z) en la misma tabla puede resultar inválido para la semántica de T. En las dos operaciones, la BD acepta los valores asociados, pero la diferencia es que la primera produce semántica válida mientras la segunda produce semántica inválida.

Por lo tanto, y según este análisis hemos considerado todas las condiciones de entrada para todas las operaciones de actualización, las cuales sólo se agrupan en una clase válida. Mientras las condiciones de entrada para claves asociadas pueden ser tanto válidas como inválidas, en función de la semántica esperada.

En la tabla 5.2 se han enumerado todas las clases de equivalencia especificadas para a continuación derivan los casos de pruebas necesarios para realizar las pruebas funcionales sobre la BD del experimento. La tabla muestra que no existen

operaciones de actualización inválidas porque tal como se explicó anteriormente el SGBD no permite lanzar una operación sino es sintácticamente válida.

2. Identificar los casos de pruebas

A partir de identificar las condiciones de entradas y las clases de equivalencia pasamos ahora a la identificación de los casos de pruebas, cada uno de estos casos cubre una clase de equivalencia. Como se muestra en la tabla 5.2, las clases de equivalencia válidas son similares a las clases inválidas, así que a continuación se presentan solo un tipo de estas clases agrupadas en función de las operaciones de actualización:

- DELETE (E1.Id_e1, E2.Id_e1, E2.Id_e2, R2.Id_e2, R2.Id_e3, E3.Id_e3, E5.Id_e3, E6.Id_e3, R4.Id_e3, R4.Id_e4, E4.Id_e4)
- INSERT (E1.Id_e1, E2.Id_e1, E2.Id_e2, R2.Id_e2, R2.Id_e3, E3.Id_e3, E5.Id_e3, E6.Id_e3, R4.Id_e3, R4.Id_e4, E4.Id_e4)
- UPDATE (E1.Id_e1, E2.Id_e1, E2.Id_e2, R2.Id_e2, R2.Id_e3, E3.Id_e3, E5.Id_e3, E6.Id_e3, R4.Id_e3, R4.Id_e4, E4.Id_e4)

Condiciones de Entrada	**Clases de equivalencia Válidas**	**Clases de equivalencia Inválidas**
Operaciones de actualización	DELETE **(1)** INSERT **(2)** UPDATE **(3)**	No hay
Claves de primarias y ajenas	E1.Id_e1 **(4),** E2.Id_e1 **(5),** E2.Id_e2 **(6),** R2.Id_e2 **(7),** R2.Id_e3 **(8),** E3.Id_e3 **(9),** E5.Id_e3 **(10),** E6.Id_e3 **(11),** R4.Id_e3 **(12),** R4.Id_e4 **(13),** E4.Id_e4 **(14)**	E1.Id_e1 **(15),** E2.Id_e1 **(16),** E2.Id_e2 **(17),** R2.Id_e2 **(18),** R2.Id_e3 **(19),** E3.Id_e3 **(20),** E5.Id_e3 **(21),** E6.Id_e3 **(22),** R4.Id_e3 **(23),** R4.Id_e4 **(24),** E4.Id_e4 **(25)**

Tabla 5.2. Las clases de equivalencias válidas e inválidas para las pruebas

3. Analizar los resultados de las pruebas

Hemos realizado las pruebas sólo sobre los casos de pruebas inválidos porque aportan más información a la hora de detectar los posibles defectos en la propuesta. Las tablas 5.3, 5.4, y 5.5 presentan los resultados obtenidos al realizar las pruebas.

En la tabla 5.3 se muestra un resumen de todos los casos inválidos probados para el borrado, las clases de equivalencia correspondientes, el tipo de pérdida de semántica producida por estos casos, y las acciones tomadas por nuestra propuesta para recuperar la semántica.

Caso	**Clases de equivalencia inválidas**		**Pérdida de semántica esperada**	**Acciones tomadas para recuperar la semántica**
1	(2) (15)	DELETE (E1.Id_e1)	Cmin(E3→R2)	ROLLBACK
2	(2) (16) (2) (17)	DELETE (E2.Id_e1, E2.Id_e2)	Cmin(E1→R1) Cmin(E3→R2)	ROLLBACK
3	(2) (18) (2) (19)	DELETE (R2.Id_e2, R2.Id_e3)	Cmin(E2→R1) Cmin(E3→R2)	ROLLBACK
4	(2) (20)	DELETE (E3.Id_e3)	Cmin(E2→R2) Cmin(E4→R4)	ROLLBACK
5	(2) (21)	DELETE (E5.Id_e3)	Totalidad (E3→E5)	DELETE (E3.Id_e3)
6	(2) (22)	DELETE (E6.Id_e3)	Totalidad (E3→E6)	DELETE (E3.Id_e3)
7	(2) (23) (2) (24)	DELETE (R4.Id_e3, R4.Id_e4)	Cmin(E3→R4) Cmin(E4→R4)	ROLLBACK
8	(2) (25)	DELETE (E4.Id_e4)	Cmin(E3→R4)	ROLLBACK

Tabla 5.3. Los casos de pruebas inválidos del borrado de la BD del experimento

Por ejemplo, el caso (1) prueba la operación DELETE (E1.Id_e1), esta operación es inválida para la semántica de la BD y producirá una pérdida de semántica respecto de

la cardinalidad mínima de la entidad E3 en la interrelación R2 (Cmin(E3→R2)). Borrar una tupla de la tabla E1 borraría en cascada todas las tuplas que referencian de la tabla E2 por la restricción de integridad referencial. El borrado de la tabla E2 producirá también un borrado en cascada de las tuplas que referencian de la tabla R2. Esto último borrado puede causar una pérdida de semántica con respecto a la cardinalidad mínima de E3 en la interrelación R2 (véase la figura 5.1). La acción tomada para cumplir la semántica por nuestra propuesta en este caso es rechazar todo la operación reiniciando el estado consistente de la BD antes del borrado.

Las clases de equivalencia (2)(18) y (2)(19) se han probado con un solo caso de prueba 3, porque borrar una tupla de la tabla R2 implica el borrado de todas las claves incluidas en la tupla borrada.

Otro ejemplo significativo de la tabla 5.3, es el mostrado en el caso 7 que prueba la operación DELETE (E5.Id_e3). Esta operación inválida causará una pérdida de semántica respecto a la totalidad de la jerarquía R3 (véase la figura 5.1), es decir, tener una ocurrencia del supertipo no asignada a uno de los subtipos. Así pues, para cumplir la semántica de la BD después del borrado de un subtipo, nuestra propuesta reacciona borrando el supertipo correspondiente.

Los casos de pruebas mostrados en la tabla 5.4 son similares a los casos anteriores pero para la operación de inserción. Destacamos algunos casos como por ejemplo, las clases de equivalencia (2)(16) y (2)(17) se han probado con solo un caso de prueba 10 porque insertar una tupla en la tabla E2 implica la inserción de las dos claves incluidas en la tupla insertada. Insertar una tupla en la tabla E2 puede producir una pérdida de semántica respecto a la cardinalidad máxima de E1.

No hemos realizado las pruebas de las clases de equivalencia (2)(18) y (2)(19) (véase tabla 5.2) que identifican la inserción en la tabla R3 (Id_e2, Id_e3) porque las cardinalidades máximas de las entidades que participan en ella; Cmax(E2→R3) y Cmax(E3→R3) son ilimitadas, por eso, no existe pérdida de semántica.

Caso	**Clases de equivalencia inválidas**		**Pérdida de semántica esperada**	**Acciones tomadas para recuperar la semántica**
9	(2) (15)	INSERT (E1.Id_e1)	Cmin(E2→R1)	*No hay reacción*
10	(2) (16) (2) (17)	INSERT (E2.Id_e1, E2.Id_e2	Cmax(E1→R1)	ROLLBACK
11	(2) (20)	INSERT (E3.Id_e3)	Totalidad (E3→E5) Totalidad (E3→E6)	*No hay reacción*
12	(2) (21)	INSERT (E5.Id_e3)	Exclusividad (E3→E6)	DELETE(E6.Id.e3)
13	(2) (22)	INSERT (E6.Id_e3)	Exclusividad (E3→E5)	DELETE(E5.Id.e3)
14	(2) (23) (2) (24)	INSERT (R4.Id_e3, R4.Id_e4)	Cmin(E3→R4) Cmin(E4→R4)	ROLLBACK
15	(2) (25)	INSERT (E4.Id_e4)	Cmin(E4→R4)	*No hay reacción*

Tabla 5.4. Los casos de pruebas inválidos para la inserción en la BD del experimento

Aunque hemos realizado una propuesta para tratar el tema de inserción segura en la BD (véase el apartado 3.4.2) esta propuesta no ha sido incluida en estas pruebas porque nuestra herramienta por ahora no genera los disparadores necesarios para controlar la inserción, por lo tanto, existe pérdida de semántica cuando se realizan los casos de pruebas 9, 11, y 15 de la tabla 5.4.

Los casos de la modificación mostrados en la tabla 5.5 son similares a los dos anteriores, sólo subrayamos que no hemos realizado las pruebas de las clases de equivalencia (3)(15), (3)(17), (3)(20), (3)(21), (3)(22), y (3)(25) (véase tabla 5.2) porque se trata de modificar la clave primaria y en este caso no se produce pérdida de semántica al estar controlado por la restricción de integridad referencial.

Como resumen podemos decir que las pruebas funcionales han reforzado nuestra propuesta de controlar la semántica utilizando disparadores porque las salidas obtenidas han sido las esperadas según las clases de equivalencia detectadas.

Los resultados obtenidos demuestran la vulnerabilidad de la BD respecto a su semántica a la hora de someterse a operaciones de actualización. Esta vulnerabilidad ha sido apoyada por nuestra propuesta de disparadores para dar un comportamiento capaz de evitar la pérdida de semántica. El 100% de las restricciones incluidas en nuestra propuesta evitan operaciones inválidas según la semántica esperada. Estos resultados han demostrado la validación del conjunto de disparadores generados por nuestras herramientas, al controlar completamente las restricciones de integridad. Cabe destacar que hemos validado los dos objetivos planteados para estas pruebas: el cumplimento del análisis de requisitos planteados en nuestra propuesta y la especificación adecuada de estos disparadores, es decir, la corrección de estos disparadores y la corrección de la interacción entre ellos y las acciones de integridad referencial.

Caso	**Clases de equivalencia inválidas**		**Pérdida de semántica esperada**	**Acciones tomadas para recuperar la semántica**
16	(3) (16)	UPDATE (E2.Id_e1)	Cmin(E1→R1) Cmax(E1→R1)	ROLLBACK
17	(3) (18)	UPDATE (R2.Id_e2)	Cmin(E2→R2)	ROLLBACK
18	(3) (19)	UPDATE (R2.Id_e3)	Cmin(E3→R2)	ROLLBACK
19	(3) (23)	UPDATE (R4.Id_e3)	Cmin(E3→R4) Cmax(E3→R4)	ROLLBACK
20	(3) (24)	UPDATE (R4.Id_e4)	Cmin(E4→R4) Cmax(E4→R4)	ROLLBACK

Tabla 5.5. Los casos de pruebas inválidos para la modificación en la BD del experimento

En otras propuestas de investigación no han utilizado pruebas funcionales para demostrar la validez de las aplicaciones activas de BD. Existen propuestas que comprueban el comportamiento de las reglas activas y los errores posibles basándose en depuradores y herramientas propias como en [Díaz, et al., 1993] y [Behrends,

1994]. En la propuesta de [Chan, et al., 1997] se realizaron pruebas de flujo o de camino básico para verificar una aplicación activa según mostraremos en la sección 5.3. Por lo tanto, nuestro trabajo en este contexto se considera una contribución más en el campo de los sistemas activos, aportando la manera de realizar pruebas funcionales sobre una BD cargada por un conjunto de disparadores.

Para completar el proceso de validación y verificación del conjunto de disparadores generados por la propuesta hemos realizado tres pruebas más, pruebas estructurales o el camino básico, que examinan todos los caminos de ejecución posibles en una secuencia de disparadores, pruebas de aceptación y pruebas del rendimiento, para examinar el comportamiento del SGBD utilizado para ejecutar la propuesta.

5.4 Pruebas del Camino Básico

Para la verificación de la propuesta se va a utilizar la prueba del camino básico [Pressman, 2003]. Es una prueba estructural que se deriva a partir del conocimiento de la estructura y la implementación del software. Tiene como objetivo principal examinar los caminos o ramificaciones posibles del flujo de la ejecución de una estructura de programa. En nuestro caso, esto significa verificar que los disparadores generados por nuestro módulo están construidos de forma correcta.

Cuando se genera un evento que activa una secuencia de disparadores, en este momento, la ejecución dependerá del camino tomado en el flujo de ejecución. Puede haber más de un camino o ramificación dentro del flujo asociado en la ejecución. En nuestro caso, las pruebas del camino básico tienen como objetivo garantizar que todos los caminos independientes en la ejecución han sido examinados por lo menos una vez, y comprobar todas las decisiones lógicas. Para ello, se han aplicado los pasos siguientes:

1. Determinar los casos de estudio

En este paso lo que se pretende es determinar todos los casos de estudio existentes en el experimento. Un caso de estudio es una secuencia, puede ser un disparador o un conjunto de ellos, activas simultáneamente por un evento de modificación. Los SGBD relacionales, en general, y en particular ORACLE, utilizan un proceso recursivo para procesar los disparadores, es decir, se suspende la ejecución del disparador T1 cuando se activa otro disparador T2 y se vuelve a procesar la ejecución de T1 cuando se termina la ejecución de T2. Esto significa, que las secuencias de código de cada disparador no interfieren entre ellas. El único problema que puede producirse es la no-terminación, tratado en la herramienta visualizadora vista en la sección 4.3.2. Por lo que la verificación se va a realizar por secuencia o

casos de estudio. La tabla 5.1 muestra 9 secuencias, cada secuencia contiene dos disparadores simultáneamente activados por el mismo evento, como {T1, T2}, {T3, T4}. La mayoría de los disparadores comparten las mismas estructuras por eso, se presenta un ejemplo que muestra como se han aplicado la técnica del camino básico a nuestra propuesta (véase el anexo I).

2. Numerar las sentencias del SQL

Para la construcción de los distintos caminos que pueden tener el flujo de ejecución de un determinado disparador, se numeran las sentencias SQL de cada secuencia según el orden de ejecución de sus disparadores. Para reducir el proceso de numeración se van a ignorar las sentencias de naturaleza declarativa, como por ejemplo, las sentencias que declaran el nombre de disparador, tipo de evento, granularidad, tiempo de activación, y la definición de los variables utilizados en el disparador. Se tendrán en cuenta, todas las sentencias que intervienen en las decisiones y los resultados. A continuación se muestra un ejemplo de la numeración de las secuencias asociadas a los disparadores {T1, T2} relacionados con la tabla R2. Esta secuencia puede ser activada por los siguientes eventos: el borrado en la tabla E2, el borrado en la tabla E3, el borrado en la tabla R2 y la actualización de clave ajena Id_e2 en R2.

```
    CREATE OR REPLACE TRIGGER T1
    BEFORE DELETE OR UPDATE ON R2
    REFERENCING NEW AS NEW OLD AS OLD
    FOR EACH ROW
1   BEGIN
2   DATONM_E2_E3.CNT_R2:= DATONM_E2_E3.CNT_R2 + 1;
3   DATONM_E2_E3.OLD_ID_E2(DATONM_E2_E3.CNT_R2):=:OLD.ID_E2;
4   DATONM_E2_E3.OLD_ID_E3(DATONM_E2_E3.CNT_R2):=:OLD.ID_E3;
5   END;
    CREATE OR REPLACE TRIGGER T2
    AFTER DELETE OR UPDATE ON R2
    DECLARE
    NTUPLES_ID_E2 NUMBER:=0;
```

```
    VAR_ID_E2 NUMBER:=0;
    NTUPLES_ID_E3 NUMBER:=0;
    VAR_ID_E3 NUMBER:=0;
 6  BEGIN
 7  FOR I IN 1 .. DATONM_E2_E3.CNT_R2 LOOP
 8  SELECT COUNT(*) INTO VAR_ID_E2 FROM E2 WHERE ID_E2 =
    DATONM_E2_E3.OLD_ID_E2(I);
 9  IF VAR_ID_E2 >0 THEN
10  SELECT COUNT(*) INTO NTUPLES_ID_E2 FROM R2
         WHERE ID_E2 = DATONM_E2_E3.OLD_ID_E2(I);
11  IF NTUPLES_ID_E2<1 THEN
12  RESET.DO ;
13  RAISE_APPLICATION_ERROR (-20502, '!! WARNING !! CANNOT
    DELETE FK = ' ||DATONM_E2_E3.OLD_ID_E2(I)||' FROM R2:
    BECAUSE A SEMANTIC LOSS WILL BE PRODUCED IN !! E2 !!' );
14  END IF;
15  END IF;
16  SELECT COUNT(*) INTO VAR_ID_E3 FROM E3 WHERE ID_E3 =
    DATONM_E2_E3.OLD_ID_E3(I);
17  IF VAR_ID_E3 >0 THEN
18  SELECT COUNT(*) INTO NTUPLES_ID_E3 FROM R2 WHERE ID_E3 =
    DATONM_E2_E3.OLD_ID_E3(I);
19  IF NTUPLES_ID_E3<1 THEN
20  RESET.DO ;
21  RAISE_APPLICATION_ERROR (-20502, '!! WARNING !!  CANNOT
    DELETE FK = '||DATONM_E2_E3.OLD_ID_E3(I)||' FROM R2
    BECAUSE A SEMANTIC LOSS WILL BE PRODUCED IN !! E3 !!' );
22  END IF;
23  END IF;
24  END LOOP;
25  DATONM_E2_E3.CNT_R2:=0;
26  END;
```

3. El diagrama del programa

El diagrama del programa es un grafo directo donde cada nodo es una sentencia y cada arco representa la dirección del flujo de la ejecución [Jorgensen, 2002]. Si i y j son dos nodos en el diagrama del programa, existe un arco del nodo i al nodo j, si y solo si, la sentencia correspondiente al nodo j se ejecuta inmediatamente después de la ejecución de la sentencia correspondiente al nodo i. En este diagrama podemos

representar solo las sentencias ejecutables y descartar otras sentencias como por ejemplo las declaraciones de variables, etc. En la figura 5.4 se muestra el diagrama de programa de la secuencia {T1, T2}. Los nodos del 1 al 6 cubren una secuencia de sentencias, los nodos de 7 al 24 cubren un bucle. Los nodos del 9 al 15 y del 17 al 23 cubren sentencias de control anidado y por último, los nodos 11 al 14 y del 19 al 22 cubren sentencias de control normal.

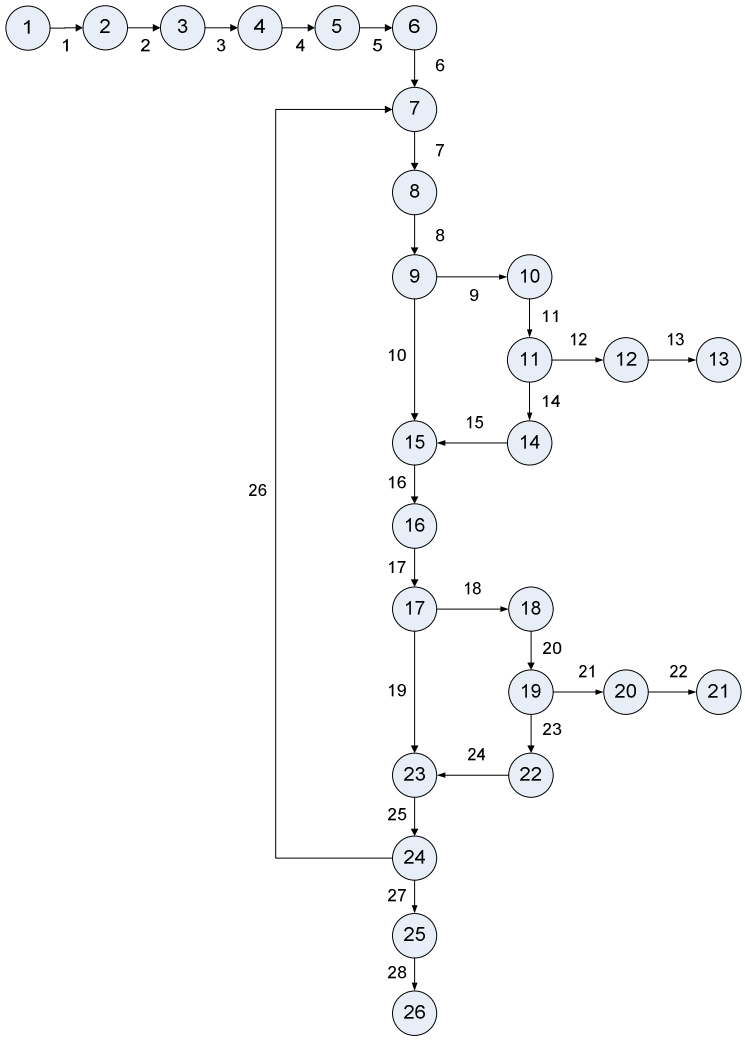


Figura 5.4. Diagrama de programa de la secuencia (T1, T2)

Para cubrir los bucles, nodos del 7 al 24, existen distintas aproximaciones que comprueban su funcionamiento y depende del tipo de estos. Los bucles se clasifican

en bucles simples, bucles anidados, bucles concatenados, y bucles no estructurales. En nuestro caso el único tipo de bucle que se usamos es el simple. Existen distintos tipos de pruebas que pueden aplicarse a este tipo de bucles [Pressman, 2003]. Por ejemplo, pasar por alto totalmente el bucle, o pasar una sola vez por el bucle, etc. En este trabajo, se ha optado por pasar una sola vez por el bucle, porque la mayoría de las actualizaciones que puede producirse en la BD son realizadas sobre sola una tupla [Huang, 1979].

El diagrama de programa mostrado en la figura 5.4 es un diagrama sencillo que contiene un número reducido de sentencias, por eso, no hemos utilizado el diagrama de *DD-Paths* (Decisión-Decisión) que se usa en los programas de cientos de sentencias, para condensar los diagramas de los mismos [Miller, 1977].

4. Complejidad ciclomática

Para saber cuántos caminos como mínimo debemos recorrer en el diagrama de programa tenemos que calcular el valor de la complejidad ciclomática (*cyclomatic complexity*) [McCabe, 1979]. La complejidad ciclomática del diagrama V(G) es el número de caminos independientes en el diagrama de programa, cada uno de estos caminos tiene por lo menos un nodo que no ha sido cubierto por cualquier otro camino. También, V(G) se considera el número mínimo de caminos que debemos examinar a la hora de realizar las pruebas. Para calcular el número ciclomático se puede aplicar la siguiente formula:

El número ciclomático de un diagrama se define como: $V(G) = e - n + 2$

Donde:
e ; es el número de arcos en G.
n ; es el número de nodos en G.

En el diagrama mostrado en la figura 5.4, el número ciclomático es: $V(G) = 28 - 26 + 2 = 4$

Esto significa que existen al menos 6 caminos independientes en el diagrama y por lo tanto, nos indica la medida de los caminos que debemos comprobar.

5. Examinar los caminos

A partir de esto, se generan datos iniciales consistentes correspondientes a cada camino. Dos tareas son realizadas en el transcurso de la prueba, la primera de ellas es monitorizar la ejecución del camino en cuestión, es decir, examinar a lo largo de la ejecución que se pasa por todos los nodos de los que se compone la sentencia objeto de estudio. La segunda de ellas, es la comprobación del estado final de la BD. La prueba es satisfactoria si el resultado final de estas dos tareas es sin errores de ejecución, ni inconsistencia en el estado de BD.

A continuación se muestran los resultados de los cuatro caminos básicos más significativos para probar la secuencia de disparadores {T1, T2} utilizada para controlar el borrado de la tabla R2:

P1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 14, 15, 16, 17, 18, 20, 23, 24, 26, 7, 8, 10, 16, 17, 19, 25, 27, 28}

Resultados esperados:

(a) Se introducen dos valores OLD.Id_e2 y OLD.Id_e3 de las claves asociadas a la tupla *t* de la tabla R2.

(b) Se verifica que el borrado de OLD.Id_e2 de la tabla R2 no afecta la semántica de cardinalidad mínima de la tabla E2. ⇒ ***Restricción verificada***.

(c) Se verifica que el borrado de OLD.Id_e3 de la tabla R2 no afecta la semántica de cardinalidad mínima de la tabla E3. ⇒ ***Restricción verificada***.

(d) Se comprueba que no existen más tuplas para borrar.

(e) **Se acepta el borrado** de la tupla *t*.

P2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13}

Resultados esperados:

(a) Se introducen dos valores OLD.Id_e2 y OLD.Id_e3 de las claves asociadas a la tupla *t* de la tabla R2.

(b) Se verifica que el borrado de :OLD.Id_e2 de la tabla R2 no afecta la semántica de cardinalidad mínima de la tabla E2. ⇒ ***Restricción no verificada***.

(c) **Se rechaza el borrado** de la tupla *t*.

P3 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 14, 15, 16, 17, 18, 20, 21}

(a) Se introducen dos valores OLD.Id_e2 y OLD.Id_e3 de las claves asociadas a la tupla *t* de la tabla R2.

(b) Se verifica que el borrado de OLD.Id_e2 de la tabla R2 no afecta la semántica de cardinalidad mínima de la tabla E2. ⇒ ***Restricción verificada***.

(c) Se verifica que el borrado de OLD.Id_e3 de la tabla R2 no afecta la semántica de cardinalidad mínima de la tabla E3. ⇒ ***Restricción no verificada***.

(d) **Se rechaza el borrado** de la tupla *t*.

P3 = {1, 2, 3, 4, 5, 6, 7, 8, 10, 16, 17, 19, 25, 27, 28}

Resultados esperados: camino imposible porque los valores de la tupla borrada (OLD.Id_e2, OLD.Id_e3) no existen en la tabla R2.

Los resultados obtenidos del camino básico para los disparadores {T1, T2} no han detectado errores en la ejecución, y el estado de la BD al final de la ejecución ha sido consistente. Aunque en esta sección sólo se ha presentado el diseño y el análisis de las pruebas para las secuencias de disparadores asociados a la interrelación R2, se han realizado los mismos pasos para el resto de secuencias.

En resumen, podemos concluir que aunque la verificación de la herramienta se ha realizado sobre un ejemplo de BD concreta no es un resultado parcial. Es decir, el código de los disparadores siempre tendrá la misma estructura pues no depende del dominio sino del tipo de restricción de integridad que controle. Por lo que, en un dominio distinto que posea algunas de las restricciones contempladas en el experimento también puede asegurarse que la herramienta generará disparadores correctos que llevarán a la BD a un estado consistente.

5.5 Pruebas de Aceptación

Las pruebas de aceptación son pruebas funcionales que se realizan por el cliente. Se comprueba el software construido a medida para un cliente para validar la realización de los requisitos predefinidos. Estas pruebas no se realizan durante el desarrollo del software, sino después de cumplir todas las fases relacionadas con el desarrollo del mismo. Normalmente, este tipo de pruebas se realiza con datos proporcionados por los usuarios porque son más parecidos a los datos reales. Puede que estas pruebas revelen errores y omisiones en la definición de los requisitos del sistema [Sommerville, 2005]. Las pruebas de aceptación se dividen en alfa o beta dependiendo de si participan uno o varios usuarios en la prueba. En nuestro caso se utilizarán las pruebas alfa de aceptación al no considerar relevante la información que puedan aportar más usuarios ya que el objetivo principal de esta prueba es demostrar que los disparadores generados por la herramienta controlan, junto con el script de creación de la BD, toda la semántica especificada en el esquema conceptual.

Para demostrar que la BD creada por la herramienta es consistente con el esquema conceptual se lanzarán distintas operaciones de modificación. En nuestro experimento las tareas del usuario constan en realizar una operación sobre la BD y validar el resultado. Por eso, se ha simulado un usuario que realiza operaciones de modificación aleatorias. Crear un usuario automático nos ha permitido lanzar casi 2500 operaciones de actualización (INSERT, DELETE, UPDATE) sobre todas las tablas de la BD del experimento y validar la semántica después de cada operación en un tiempo muy reducido.

Para ello se han realizado dos pruebas, la primera lanzará los eventos para modificar la base de datos sin disparadores. La segunda prueba lanzará los mismos eventos para modificar la base de datos con disparadores. Es decir, ambas BD de datos son

generadas por el mismo script aunque la primera de ellas no completa la semántica a través de disparadores.

Los resultados de la primera prueba se muestran en la tabla 5.6, los cuales indican que después de las operaciones de modificación a la que ha sido sometida la BD solo 974 tuplas permanecen en la BD frente a unas 1360 tuplas de la segunda prueba. Lo que significa que la propuesta de los disparadores ha evitado la desaparición de casi 386 tuplas por razones de conservación de las restricciones de integridad de la BD.

Tablas	**Tuplas antes de pruebas**	**Las tuplas restantes después de las dos pruebas**	
		Sin Trigger	**Sin Trigger**
E1	200	125	125
E2	400	165	250
E3	100	53	54
E4	100	65	67
E5	50	18	27
E6	50	14	27
R2	4000	470	712
R4	400	64	98
Total	**5300**	**974**	**1360**

Tabla 5.6. Los porcentajes del cumplimento de la semántica después las pruebas de aceptación

En la tabla 5.7 se muestran los resultados de la exactitud del funcionamiento de los disparadores, donde se ve claramente la diferencia entre los porcentajes de las tuplas que cumplen los requisitos de la semántica una vez sometida la BD a las operaciones de actualización en cada una de las pruebas.

Como podemos observar en la tabla, la segunda prueba cumple con el objetivo principal de nuestra herramienta, generar un script de una BD relacional junto con

los disparadores necesarios para recoger toda la semántica especificada en un esquema conceptual. Por lo que podemos concluir que los disparadores generados por la herramienta son válidos.

Además, la tabla muestra un hecho muy importante y el cual motivó la realización de este trabajo de tesis doctoral: el desarrollo de una BD relacional que sólo consta de las restricciones declarativas proporcionadas por el modelo relacional no son suficientes para reflejar la semántica contenida en un esquema conceptual y las pérdidas semánticas son significativamente grandes según se muestra en la tabla 5.7.

Nº	**Tipo de pérdida de Semántica de la Tabla 5.1**	**% de las tuplas que cumplen la semántica**	
		Sin Trigger	**Con Trigger**
1	Cmin(E2→R1)	8.8%	100%
2	Cmax(E2→R1)	6.2%	100%
3	Cmin(E2→R2)	4,6%	100%
	Cmin(E3→R2)	3.2%	100%
4	Cmin(E3→R4)	48.1%	100%
	Cmin(E4→R4)	29.8%	100%
5	Cmax(E3→R4)	9.3%	100%
	Cmax(E4→R4)	6.9%	100%
6	Totalidad de E3→E5	38.8%	100%
8	Totalidad de E3→E6		
7	Exclusividad de E3→E5	4.4%	100%
9	Exclusividad de E3→E5	2.6%	100%

Tabla 5.7. Los porcentajes del cumplimento de la semántica después de cada prueba.

5.6 Pruebas de Rendimiento

En general, las pruebas de rendimiento están diseñadas para probar la eficiencia del software en tiempo de ejecución dentro del contexto de un sistema integrado [Pressman, 2003]. El objetivo principal de estas pruebas es realizar una carga masiva sobre la BD del experimento y comprobar la eficiencia de nuestra propuesta.

Para tener una visión global sobre la eficiencia de los disparadores necesitamos tener un conjunto de referencias que nos indican si su rendimiento es bueno o sobrepasa los límites definidos por esas referencias. Ya que según [Geppert et al., 1995] la definición de una referencia (*Benchmark*) para la medición del rendimiento es un requisito imprescindible dentro de los sistemas activos. El problema actualmente es que no existe una referencia para medir el rendimiento de los sistemas activos relacionales debido a las limitaciones del lenguaje de los disparadores y a las limitaciones del modelo de ejecución [Paton, 1998]. Por el contrario, si se han elaborado algunas referencias para sistemas orientados a objetos, como por ejemplo, BEAST [Geppert et al., 1998], etc. Estas referencias, en general, se centran en aspectos como la detección de eventos, la gestión de reglas, y el modelo de ejecución, y estos aspectos no pueden ser tratados en la misma medida dentro de los sistemas relacionales.

Los objetivos específicos de nuestras pruebas de rendimiento son: primero monitorizar los defectos o errores probables que no se han visto en las pruebas anteriores, y segundo, realizar una comparación entre el rendimiento dentro del mismo SGBD, con distintas propuestas para realizar las mismas tareas en nuestro caso, el control de restricciones de integridad. La primera propuesta utiliza los disparadores generados por nuestra herramienta, y la segunda propuesta utiliza procedimientos almacenados.

Para llevar a cabo nuestras pruebas de rendimiento se han tomado referencias para medir ciertos parámetros como el tiempo de respuesta de ejecución (*Elapsed Time*).

A continuación se presenta el diseño del experimento que se ha utilizado para comprobar el rendimiento de los disparadores según nuestra propuesta:

1. Diseño del experimento:

Para realizar las pruebas de rendimiento se ha desarrollado lo siguiente:

- Se ha transformado la BD del experimento al modelo relacional tal y como se muestra en la figura 5.2.
- Se ha generado los disparadores necesarios para controlar las restricciones de integridad mostradas en la tabla 5.1.
- Se han implementado manualmente los procedimientos almacenados necesarios para controlar las mismas restricciones que aparecen en la tabla 5.1.
- Se han insertado más de 5000 tuplas distribuidas en las tablas de la BD del experimento.
- Posteriormente, se han creado las transacciones de las operaciones de actualización aleatorias necesarias para ejecutar las dos propuestas. Estas transacciones son iguales y se ejecutan una vez con los disparadores y la otra con los procedimientos almacenados.

A continuación pasamos a definir los parámetros que se han tenido en cuenta.

2. Parámetros a medir:

Esta evaluación de parámetros consistirá básicamente en probar, con las distintas tablas y distintas operaciones de actualización aleatorias, cómo afecta la inclusión de disparadores en el funcionamiento de la BD comparándolos con los parámetros de referencia, los cuales vendrán dados por la implementación de procedimientos almacenados. El SGBD utilizado para realizar las pruebas ha sido ORACLE9i.

El parámetro que tendremos en cuenta para comprobar el rendimiento será el tiempo de respuesta de ejecución (*Elapsed Time*) de un conjunto de actualizaciones en un entorno multiusuario [Boral y DeWitt, 1984]. El tiempo de la respuesta es el tiempo

necesario para el proceso de una sentencia SQL (*S*), este proceso consiste en tres pasos: *Parse, Execute y Fetch*. El primer paso es analizar *S*, según el plan de ejecución. Este paso incluye las comprobaciones de la autorización de seguridad, la existencia de tablas, columnas. En el segundo paso, se ejecuta *S*, para la sentencia de recuperación de datos este paso identifica las tuplas seleccionadas. En el tercer paso, se recuperan las tuplas seleccionadas por una consulta. Este paso se realiza sólo para la sentencia (SELECT).

Para obtener este parámetro se ha utilizado las dos herramientas básicas SQL TRACE y TKPROF de ORACLE que nos proporcionan los resultados que permiten diagnosticar el rendimiento de nuestro experimento. SQL TRACE proporciona información sobre el rendimiento de cada sentencia SQL individual, como *Parse*, *Execute*, *Fetch counts*, *CPU time*, *Elapsed times*, etc. Al ejecutar el programa TKPROF se convierte el archivo obtenido de SQL TRACE en un archivo legible que contiene estadísticas sobre los recursos utilizados para la ejecución de cada sentencia SQL.

Para determinar el tiempo de respuesta, los experimentos se analizan de la manera siguiente: se considera que U es el número de usuarios del experimento y N es el número de iteraciones para ejecutar cada usuario. T_{ij} representa el tiempo de respuesta medido desde el inicio hasta el final de la ejecución. El tiempo medio de respuesta para cada usuario U_i será calculado dividiendo la suma de los tiempos de dicho usuario por el número de iteración.

El tiempo de la respuesta del usuario: $U_i = \frac{\sum_{j=1}^{N} T_{ij}}{N}$

3. Implementación del experimento:

En nuestro trabajo la implementación de los experimentos se ha realizado empleando dos aproximaciones con el mismo objetivo. Este objetivo es controlar las restricciones de integridad presentadas en la tabla 5.1. La primera aproximación se

ha realizado mediante disparadores asociados a las tablas del esquema relacional, mientras la segunda aproximación ha utilizado los procedimientos almacenados.

Para asegurar que la diferencia en el rendimiento entre estas propuestas viene dada por la naturaleza de la ejecución de cada una, se ha utilizado las mismas sentencias en ambas, como por ejemplo, CURSOR, IF THEN, etc. Además de esto, para calcular exactamente el tiempo de respuesta se han utilizado transacciones que no transmiten resultados al usuario.

Tanto la codificación de los disparadores como la de los procedimientos almacenados utilizan una política pesimista en el control de las restricciones. Esta política actúa sobre la BD de la siguiente forma: cada vez que se realiza una operación de actualización se comprueba si las restricciones se cumplen. Si se produce cualquier violación de las restricciones la operación de actualización se rechaza (*rollback*) y el estado inicial de la BD se restaura.

4. Resultados:

En la figura 5.5 se puede ver los resultados de rendimiento de las dos propuestas cuando se ejecutan distintas transacciones que realizan operaciones de actualización aleatorias de 100, 200,.., 1000 tuplas. Cada transacción en cada propuesta se ha ejecutado por un solo usuario cinco veces (U=1, N=5). El resultado ha demostrado que la diferencia entre el rendimiento de las dos propuestas varía entre 3%-4% a favor de los procedimientos almacenados, es decir, que no existe una diferencia significativa entre las dos propuestas al ejecutarlas por un usuario (U=1). Nosotros consideramos que esta diferencia es causa de la utilización de dos disparadores para hacer la tarea de un procedimiento almacenado. Se han utilizado dos disparadores para evitar el problema de las tablas mutantes en ORACLE (véase el apartado 3.2).

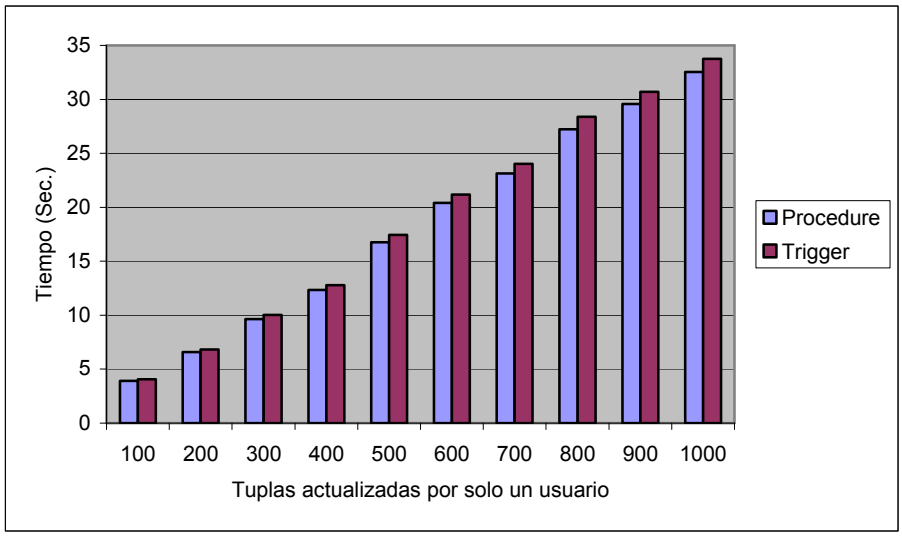


Figura 5.5. El rendimiento de las dos propuestas con un usuario

El siguiente experimento varía el número de usuarios (U = 2,3,4,..., 10) aunque mantiene el número de repeticiones (N=5). Cada uno de estos usuarios realiza el mismo número de actualizaciones aleatorias sobre la BD experimental. En las figuras 5.6 y 5.7 se muestran los resultados del rendimiento para la propuesta de disparadores y la de procedimientos almacenados, donde, U = (2,3,4,…,10).

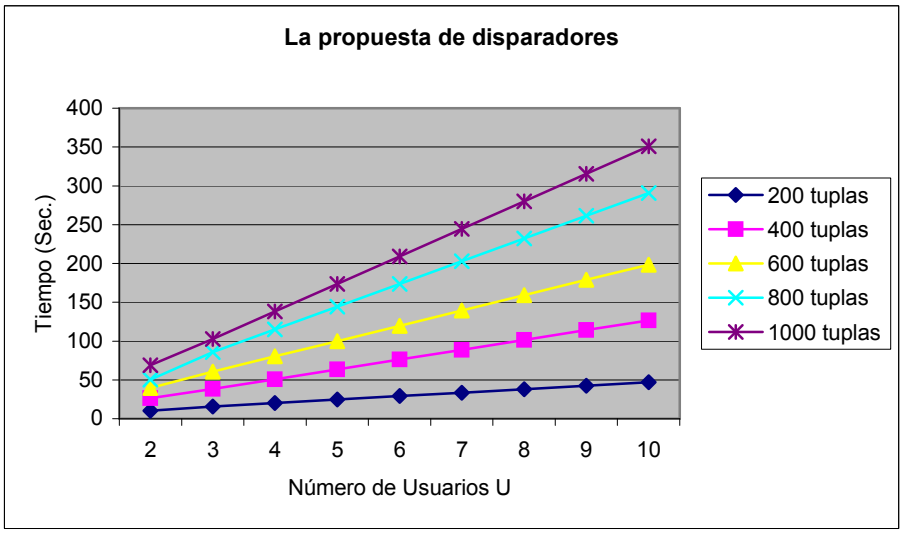


Figura 5.6. El rendimiento de la propuesta de disparadores con distintos usuarios

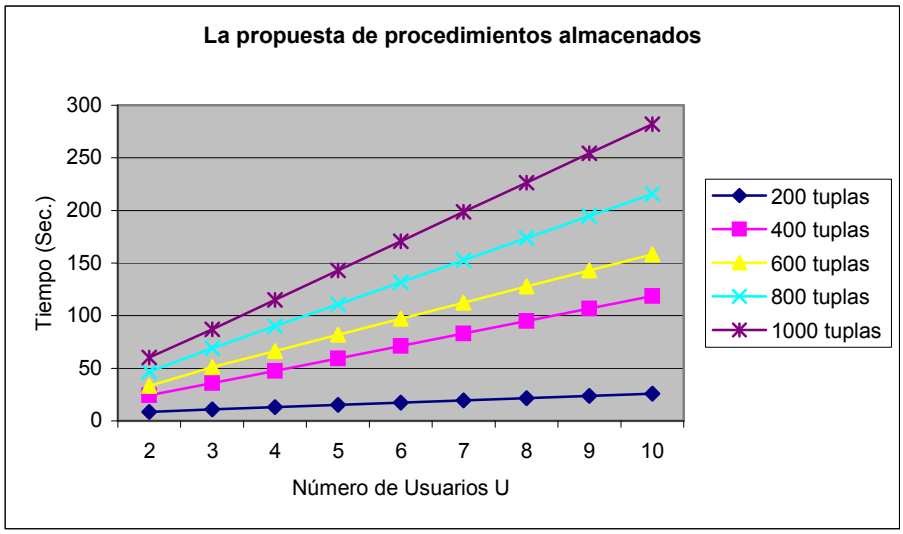


Figura 5.7. El rendimiento de la propuesta de procedimientos con distintos usuarios

En la figura 5.8 se muestra la comparación del rendimiento entre los resultados obtenidos por las dos propuestas cuando trabajan 10 usuarios a la vez. En ella puede observarse una diferencia pequeña a favor de los procedimientos almacenados. Diferencia que viene motivada por la ejecución de dos disparadores para realizar la misma tarea que un solo procedimiento almacenado.

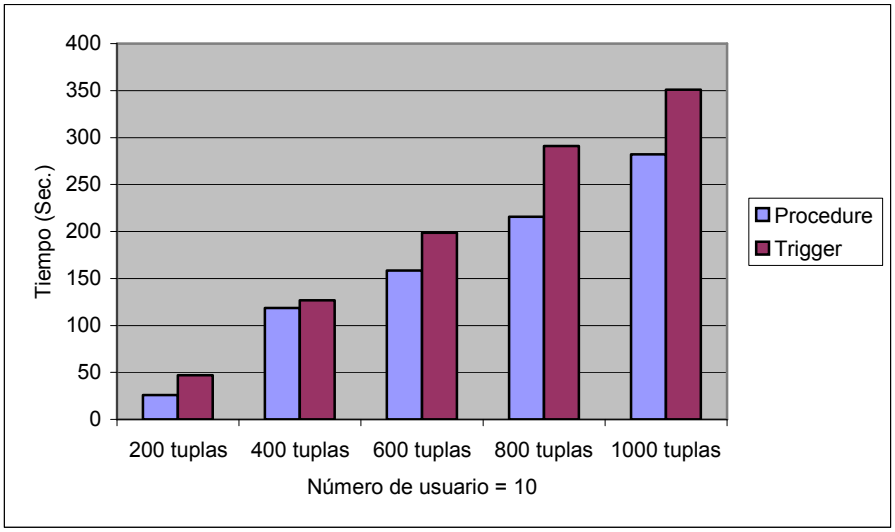


Figura 5.8. Comparación entre los resultados del rendimiento de las dos propuestas

Podemos concluir que el establecimiento de referencias para medir el rendimiento de una BD relacional con reglas activas es muy importante para su validación. Al no tener referencias para este tipo de sistemas se han definido parámetros de rendimiento a través de los procedimientos almacenados. Para ello se han elaborado un conjunto de procedimientos almacenados con la misma función y con las mismas sentencias SQL que los disparadores generados en nuestra propuesta.

Como resumen, se puede decir que como cualquier otro tipo de sistema de software, los sistemas activos tienen que proporcionar sus funcionalidades de una manera eficiente. Por eso, existen estudios acerca de cómo evaluar el rendimiento de estos sistemas y en general, se han focalizado al análisis de las ganancias y las pérdidas obtenidas frente a los sistemas pasivos. De estos estudios se ha demostrado que el uso de mecanismos proporcionados por los SGBD activo resulta un arma de doble filo. Es decir, que la utilización de estos mecanismos puede resultar lenta porque el sistema de reglas activas es un mecanismo de propósito general que introduce una sobrecarga en el SGBD y por tanto, la comparación con un sistema pasivo no va a resultar significativo. Sin embargo, existen dos razones que demuestran que la utilización de las reglas activas puede mejorar el rendimiento de las aplicaciones [Simon y Kotz-Dittrich, 1995] [Paton, 1998]. La primera razón viene dada por la centralización de semántica de la aplicación en estas reglas, lo que implica que se puedan aplicar buenas y mejores técnicas para la optimización, se puede evitar la redundancia de la verificación, y los cambios en el entorno se realizan fácilmente. La segunda razón, viene dada por considerar que las reglas activas son instrumentos de afinación (*Tuning Instrument*) eficaces para que la aplicación sea rápida. Véase el caso de sustituir la transacción tradicional (*polling transaction*) por disparadores [Simon y Kotz-Dittrich, 1995].

La conclusión final puede señalar la necesidad de implementar BD activas eficaces y llevar a cabo esta tarea utilizando herramientas de desarrollo que faciliten la implementación de aplicaciones activas. El rendimiento de estas BD juega un papel crucial y debe tenerse presente cuando los sistemas se diseñan y se llevan a cabo.

5.7 Análisis Global de los Resultados

En general, todavía no existe una validación formal (*Mathematically-based*) de las reglas activas porque la mayoría de los sistemas activos de BD relacionales no proporcionan el soporte para una descripción formal de los requisitos de la regla. Los requisitos de la regla y la semántica de la ejecución de estas, en la mayoría de los casos están descritos de una manera informal [Vaduva, 1998]. Sin embargo, existen algunos aspectos del comportamiento activo que han sido definidos formalmente en el marco de las BD deductivas [Fernandes, 1997], o en BDOO [Fraternali y Tanca, 1995]. En este contexto los métodos informales pueden ser utilizados, como por ejemplo, la inspección en el código de las reglas para comprobar si la especificación de la regla corresponde a los requisitos.

La técnica que se ha utilizado para la validación consta de la aplicación de varias pruebas de software (*Software Testing*). Las pruebas de software que se han utilizado son dinámicas y pueden validar y verificar si los disparadores desarrollados en las herramientas desarrolladas para nuestra propuesta no generan ningún error, ni producen inconsistencia en los resultados y satisfacen los requisitos de usuarios.

Para realizar las pruebas de verificación y validación de nuestra propuesta se ha diseñado un experimento centrado en una BD como ejemplo. A partir de esta BD se ha construido el esquema conceptual del mismo, el cual contiene todas las restricciones de integridad problemáticas para poder aplicar la propuesta presentada en este trabajo de tesis doctoral.

Las pruebas realizadas y los resultados de la realización de las mismas se muestran a continuación:

- Las pruebas funcionales de partición equivalente han sido utilizadas para la validación de la propuesta. El objetivo principal es probar que el software obtenido por nuestra herramienta realiza las tareas definidas por el usuario. Es

decir, realizan la función para la que han sido diseñados. Para ello, se han efectuado pruebas que constan de realizar casos de pruebas sobre la BD creada. Los resultados obtenidos han demostrado que el 100% de las restricciones incluidas en nuestra propuesta han sido reforzadas ante la violación de una operación inválida. Es decir, los resultados han demostrado que los disparadores generados por las herramientas propuestas controlan completamente las restricciones de integridad estudiadas, como las restricciones de cardinalidad de las interrelaciones binarias y las participaciones de los supertipos y subtipos en las jerarquías. Por lo tanto, hemos validado con estas pruebas, la corrección de los disparadores y la corrección de la interacción entre ellos y las acciones de integridad referencial.

- Las pruebas del camino básico han sido utilizadas para la verificación de nuestra propuesta. El objetivo principal es dar una garantía de que los disparadores generados por nuestras herramientas son correctos. Para ello se han examinado 12 casos de prueba o secuencias de disparadores que contienen 38 caminos independientes. Se ha monitorizado el flujo de ejecución y el estado final de la BD después de cada prueba. El resultado final indica que no existen ningún error de ejecución ni se han producido estados inconsistentes en la DB. Por lo que los disparadores generados son correctos.
- Las pruebas de aceptación han sido utilizadas para la validación de la propuesta, tiene como objetivo principal reforzar los resultados obtenidos en las pruebas de partición equivalente y probar que el software obtenido por nuestras herramientas realiza las tareas definidas por el usuario. Se han realizado estas pruebas lanzando casi 2500 operaciones de actualización aleatorias sobre las tablas de la BD del experimento. Los resultados finales de las pruebas de aceptación han demostrado que la semántica de todas las tuplas restantes en la base de datos cumple 100% las restricciones definidas en la tabla 5.1, lo que significa cumplir totalmente los requisitos de los usuarios. Los errores que han sido registrados en estas pruebas son errores programados, es decir, los errores producidos por la

pérdida de semántica. Ningún otro error ha sido observado en todas estas pruebas de aceptación lo que valida nuestros disparadores y también valida la herramienta que los genera.

- Las pruebas del rendimiento han sido realizadas como otro tipo de pruebas para la validación de la propuesta. El objetivo principal de estas pruebas es analizar a través de una carga masiva de datos cómo afecta la ejecución de los disparadores al rendimiento de la BD. Para realizar el análisis, se han tenido que definir un conjunto de parámetros que sirvan de referencia. Estos parámetros están basados en la implementación de procedimientos almacenados con la misma funcionalidad y secuencias SQL que los disparadores generados por la herramienta. Los resultados de estas pruebas indican una diferencia pequeña a favor de los procedimientos almacenados. Esta diferencia viene motivada por la limitación del sistema activo de ORACLE que nos ha obligado a utilizar dos disparadores para realizar la tarea de un procedimiento almacenado, evitando el problema de tablas mutantes.

Capítulo 6

6. Conclusiones y Líneas Futuras

6.1 Conclusiones

Las conclusiones de este trabajo de tesis doctoral se derivan de la consecución de los objetivos planteados inicialmente. En el marco de una metodología de desarrollo de BD los objetivos se enfocaban en distintas fases que pasamos a enumerar a continuación.

En las primeras fases de la metodología para desarrollar una BD, nos hemos encontrado que los modelos conceptuales que se aplican son cada vez más potentes para presentar la semántica y esto implica que son capaces de recoger con mayor precisión las especificaciones del dominio. El problema surge cuando queremos mantener esta semántica en la siguiente fase de la metodología, la fase lógica, en la que para transformar el esquema conceptual a un esquema relacional se aplican un conjunto de reglas. Estas reglas son, en general, muy básicas pues solo especifican la transformación de los elementos más simples y sencillos del modelo conceptual. Todo ello implica una pérdida de semántica significativa que lleva al diseñador a controlar y comprobar las restricciones de integridad fuera de la BD.

En este trabajo de tesis doctoral se presenta la utilización de mecanismos activos para preservar la semántica dentro de la BD. Para ello se han definido un conjunto de pasos que transforman las restricciones de integridad en un esquema conceptual sin

perder la semántica que estas reflejan. Se ha considerado que una restricción de integridad se define como una regla activa con tres componentes:

- Una operación de actualización (inserción, borrado, o modificación) cuya ejecución ha de dar lugar a la comprobación del cumplimiento de la restricción.
- Una condición que debe cumplirse, la cual es, en general, una proposición lógica, definida sobre uno o varios elementos del esquema, que puede tomar uno de los valores de verdad (cierto o falso).
- Una acción que debe llevarse a cabo dependiendo del resultado de evaluar la condición.

Además de estos componentes estáticos, en este trabajo se ha considerado derivar los componentes dinámicos (granularidad de la regla y el tiempo de la activación) de la especificación de la restricción. En resumen, la aportación principal de este trabajo de tesis doctoral, dentro de la fase conceptual de la metodología de desarrollo de BD, se enfoca en enriquecer las reglas de transformación del modelo conceptual al modelo relacional con tecnología activa, para controlar la semántica del UD en general, y en particular, las restricciones de integridad que pueden ser establecidas en un esquema conceptual y que no tienen una transformación directa al modelo relacional.

La aplicación de la propuesta a las restricciones más conflictivas en la fase conceptual y estudiar su efecto en el rendimiento de la BD es otro de los objetivos propuestos que se ha cubierto en este trabajo. En lo concerniente a la mejora del rendimiento de la BD, se ha detectado que la implementación de mecanismos para conservar todas las restricciones de integridad, en particular, la participación obligatoria en las interrelaciones binarias puede llevar consigo un impacto sobre el rendimiento del SGBD. La relación entre la semántica recogida de la BD, la complejidad de implementación utilizada para conservar la semántica y el rendimiento del SGBD indica que al reforzar la semántica de la BD a través de aumentar la complejidad de la implementación, se reduce el rendimiento del SGBD. Por ello, en este trabajo se ha propuesto una aproximación para mejorar el

rendimiento de la BD basado en definir un conjunto de métricas y un umbral de cumplimiento para establecer qué restricciones de participación son las que realmente necesitan verificarse antes.

Otro de los objetivos es la implementación de esta propuesta en una herramienta CASE. Este objetivo venía motivado por dos factores. El primero de ellos es la poca utilización de mecanismos activos en las BDs relacionales aún sabiendo de su potencial para expresar restricciones. En general, este rechazo por parte de los diseñadores y desarrolladores de BD se debe a que el modelo de ejecución es poco claro, y la sintaxis y comportamiento depende del SGBD. Otro problema al que se deben de enfrentar es la interacción de los mecanismos activos y las acciones de integridad referencial, que puede producir ejecuciones en cascada no controladas. También, a veces, el uso intensivo de estos mecanismos puede producir un impacto sobre el rendimiento de la BD.

El segundo factor viene dado por el estudio de herramientas CASE comerciales que pone de manifiesto las carencias en la cobertura de todas las fases de la metodología de desarrollo de BD, así como una pobre implementación del conjunto de reglas de transformación entre las fases de la metodología, provocando una BD con muy poca semántica asociada. La mayoría de las herramientas CASE comerciales soportan la definición del esquema de la BD global, admitiendo representaciones gráficas del modelo conceptual, según las metodologías de diseño de BD que soportan. Así, nuestro objetivo principal es el desarrollo de herramientas (herramienta generadora de disparadores y herramienta visualizadora del comportamiento de estos) que integren las reglas propuestas de transformación incluyendo los mecanismos activos y además proporcionen un método de visualización de estos mecanismos para entender cuál es su comportamiento.

Estas herramientas han sido integradas en la herramienta comercial Rational Rose por su flexibilidad para agregar módulos *add-in* fácilmente. Desde el esquema conceptual creado en Rational Rose con sus restricciones establecidas, las herramientas serán capaces de realizar las tareas que se muestran a continuación:

- La herramienta generadora tiene como objetivos específicos la transformación automática de las restricciones de integridad especificadas en el esquema conceptual a cláusulas del cálculo relacional (TRC) según la metodología propuesta y, posteriormente, se transforman a disparadores del SQL3. Para conseguir una implementación de la BD, la herramienta generará un script con la semántica conceptual ejecutable dentro del SGBD elegido.
- La herramienta visualizadora tiene como objetivo principal solucionar el problema de la invisibilidad del orden de la ejecución y el comportamiento de las reglas activas. Esta herramienta utiliza los componentes del Diagrama de Secuencia de UML para modular el proceso de ejecución de las reglas, visualizando la activación en cascada de ellas. Para detectar el problema de la no terminación se ha utilizado el análisis sintáctico y estático de disparadores y el grafo de desencadenamiento (*Triggering Graph*) para mostrar el comportamiento. La herramienta manda mensajes a los desarrolladores sobre el estado de la verificación.

Para validar estas herramientas se ha realizado un experimento centrado en un escenario concreto. Este escenario recoge la semántica de un dominio determinado. Se han realizado distintas pruebas de software sobre la BD de este experimento.

A continuación se enumeran los resultados más destacados obtenidos de la consecución de estas pruebas:

- Para consolidar esta propuesta se han utilizado dos tipos de técnicas de pruebas funcionales, la técnica de partición equivalente y la técnica de aceptación.
    - Para la técnica de partición equivalente se han identificado las clases de equivalencia y sus casos de pruebas correspondientes con el objetivo de cubrir todos estos casos. En total se ha realizado para la BD del experimento 20 casos de prueba cubriendo todas las restricciones definidas en el ER. Los resultados obtenidos han demostrado que el 100% de las restricciones incluidas en nuestra propuesta han sido reforzadas ante la violación de una operación inválida. Esto significa que los disparadores generados por las

herramientas propuestas controlan completamente las restricciones de integridad estudiadas en este trabajo.

- Para la técnica de aceptación se ha sometido a la BD, implementada con disparadores generados para controlar la semántica, a más de 2500 operaciones de actualización aleatorias. Los resultados han demostrado que el 100% de la semántica de las restricciones de integridad definidas en el esquema ER han sido controladas y protegidas.

- Respecto a las pruebas del camino básico, se han utilizado para garantizar que los disparadores generados por nuestras herramientas son correctos. Se han examinado nueve secuencias de disparadores, donde una secuencia de disparadores puede consistir en un disparador o más y que son activados por el mismo evento, cubriendo todos los caminos definidos en estos disparadores a través de generar datos iniciales y lanzar un evento. Los resultados han demostrado que no existe ningún error de ejecución de estos caminos ni se han producido estados inconsistentes en la DB después de la ejecución. Por lo que indica que los disparadores generados por nuestras herramientas, están correctamente construidos para cumplir con los requisitos impuestos por UD.

- Respecto a las pruebas del rendimiento, se han realizado para saber cómo afecta la ejecución de los disparadores al rendimiento de la BD. Los parámetros utilizados para realizar estas pruebas se basen en la implementación de procedimientos almacenados con la misma funcionalidad y secuencias SQL que los disparadores generados por la herramienta. El valor de estos parámetros nos dará un punto de referencia para poder comparar con los valores de estos parámetros aplicados a los disparadores. La experimentación realizada indica que no existen diferencias significativas y, por lo tanto, podemos concluir que aunque la BD sufre pérdida en su rendimiento este es aceptable.

Los resultados obtenidos por estas pruebas han demostrado que los disparadores generados acometen su funcionalidad, es decir, cumplir el objetivo establecido desde el principio: enriquecer las reglas de transformación conservando totalmente las restricciones de integridad de la BD.

Somos conscientes que nuestra propuesta tiene sus ventajas e inconvenientes. Las ventajas son principalmente consecuencia del tratamiento dentro de la BD de la semántica del UD que se modela. Con ello, evitamos información redundante en las distintas aplicaciones que acceden a la BD, descargándolas de esta responsabilidad. Esto repercute en una mejor adaptabilidad del sistema ya que si cambia alguna de las restricciones de integridad asociadas al UD, solo modificamos la BD y no en todas las aplicaciones que acceden a ella. Otra ventaja es que si modelamos un sistema crítico con limitaciones importantes en las restricciones de integridad, la comprobación de la BD nos asegurará que los datos almacenados cumplen todos los requisitos de del sistema y por tanto, no tendremos información corrupta.

Los inconvenientes resultan de varios factores, además de los problemas asociados con la utilización de la tecnología activa: usar mecanismos activos puede afectar al rendimiento de la BD. Algunos de estos factores pueden reducirse con la utilización de herramientas de ayuda al desarrollo de BD relacionales con tecnología activa, según se ha mostrado en esta propuesta.

En resumen, para reducir estos inconvenientes se enumeran a continuación las soluciones adoptadas dentro de este trabajo de tesis doctoral:

- Aunque la implementación de los sistemas activos es significativamente más complicada que en los sistemas tradicionales, la utilización de herramientas de ayuda para desarrollar estos sistemas resulta útil. Herramientas nuevas que cubren algunos huecos de las herramientas comerciales pueden servir para motivar a los desarrolladores a una mayor utilización de los sistemas activos.
- Para mejorar el rendimiento del SGBD se puede utilizar la propuesta para relajar las restricciones de integridad con el fin de reducir la complejidad de implementación.

6.2 Difusión de Resultados

La evolución de este trabajo de tesis doctoral se ve reflejada en un conjunto de publicaciones que han ayudado a mejorar la propuesta y ampliarla.

El trabajo presentado en [Al-Jumaily et al., 2002] aporta una solución para la pérdida de semántica en la transformación de interrelaciones binarias a un modelo relacional. La solución propuesta se basaba en la aplicación de reglas ECA (evento-condición-acción) con la particularidad de introducir un procedimiento de petición de datos para que la inserción contemplara las restricciones de cardinalidad. Además, se creó un algoritmo de transformación para este tipo de interrelaciones permitiendo que las instancias de una interrelación contuvieran información inaplicable o desconocida.

Algunos estudios acerca de las carencias de las herramientas CASE y la aportación de ciertos algoritmos, para el módulo de transformación de esquemas ER a esquemas relacionales, que podrían añadirse a las mismas, se muestra en [Al-Jumaily et al., (a) 2003].

En [Al-Jumaily et al., (b) 2003] se presenta una propuesta que pretende contribuir al campo del desarrollo de la tecnología activa, concretamente a las bases de datos activas, realizando una mejora del modelo de ejecución de los disparadores del estándar SQL3 que evite problemas potenciales durante su ejecución. El objetivo concreto es reforzar cualquier tipo de restricción de integridad al realizar su transformación al modelo relacional. Para completar la aportación, se implementa esta técnica dentro de una herramienta CASE denominada PANDORA[6], a través de un generador de disparadores. Por último y para validar dicha propuesta, se realiza un estudio de rendimiento de la BD con la incorporación de los disparadores que controlan la semántica de las cardinalidades. Este estudio se desarrolla en el SGBD

[6] Proyecto CICYT, referencia TIC1999-0215

ORACLE 9i y se diagnostica su rendimiento a través de las herramientas proporcionadas por dicho SGBD.

El trabajo [Al-Jumaily et al., (a) 2004] estudia el problema del impacto sobre el rendimiento del SGBD al implementar mecanismos para conservar todas las restricciones de integridad en BD. En el trabajo se definen un conjunto de métricas que puntúan el grado de cumplimiento de las restricciones de integridad en cada tabla de la BD. A través de estas métricas (cuantificadores difusos relativos) los desarrolladores pueden tomar decisiones sobre qué restricciones tiene más importancia a la hora de ser controladas.

En [Al-Jumaily et al., (b) 2004] se muestra y explica la implementación llevada a cabo para la herramienta generadora de disparadores. Y en [Al-Jumaily et al., 2006] se expone la implementación de la herramienta visualizadora del comportamiento de los disparadores utilizando diagramas de secuencia del UML.

6.3 Líneas Futuras

Una vez expuesto el trabajo desarrollado en la presente tesis doctoral se van a enumerar a continuación algunas de las posibles vías de estudio y desarrollo. A continuación se enumeran las líneas futuras de esta tesis doctoral:

- Una de las líneas de investigación actuales en Ingeniería del Software se dirige hacia la generación automática de códigos dentro del desarrollo de aplicaciones. Aunque esta línea parece ser no demasiado innovadora, la introducción de desarrollos dirigidos por modelos (MDD: *Model Driven Development*) [OMG, 2006] han proporcionado una nueva tendencia y una forma de resolver este problema para dominios cada vez más complejos. MDD plantea el uso de modelos como aproximaciones para cubrir el ciclo de vida del desarrollo del software. La arquitectura dirigida por modelos (MDA: *Model Driven Architecture*) se compone básicamente de tres capas de abstracción: el modelo independiente de computación (CIM: *Computation Independent Model*) que específica las reglas de negocio sin tener en cuenta aspectos computacionales. El modelo independiente de la plataforma (PIM: *Platform Independent Model*) que representa la lógica del negocio a alto nivel, sin considerar los aspectos de la tecnología utilizada para la implementación. Por último, el modelo específico de la plataforma (PSM: *Platform Specific Model*) que específica en más detalle la tecnología utilizada para implementar el sistema. La contribución principal de MDD es la aportación de una solución para la heterogeneidad de sistemas con diferentes plataformas de implementación y la interoperabilidad entre ellos. En particular, la aplicación de MDD en el desarrollo de BDs es una tarea muy ambiciosa que necesita herramientas de desarrollo más eficaces para la creación, no solo de los esquemas lógicos, sino también proporcionar facilidades para contemplar todos los aspectos necesarios para cumplir la semántica del UD.

Por lo tanto, nuestra propuesta aporta una solución dentro de MDD aplicada al desarrollo de BDs relacionales y una de las líneas futuras en las que se enfoca este trabajo de tesis doctoral se dirige a la ampliación de facilidades de la herramienta implementada. Más concretamente, se trabajará en los siguientes aspectos:

- La herramienta de generación de disparadores solo considera aquellas restricciones reflejadas en el esquema conceptual. Para poder representar todas las especificaciones del UD se hace necesario ampliar la herramienta para permitir definir otro tipo de restricciones. Por ejemplo, las restricciones dinámicas. Uno de los trabajos futuros se enfoca al desarrollo de un módulo para la definición de restricciones de integridad que no pueden representarse en un esquema conceptual. El módulo añadido a la herramienta generadora contendrá un interfaz interactivo con los usuarios. A través de este interfaz, se permitirá a los usuarios introducir las restricciones de integridad de una manera natural, con el objetivo de convertir estas restricciones a cláusulas del cálculo relacional de tupla, y posteriormente aplicar nuestra propuesta para generar los disparadores necesarios para controlar las mismas. Este módulo juega un papel muy importante en automatizar la formalización de las reglas de negocio cubriendo el aspecto conceptual CIM del MDD.
- Todavía no existen desarrollos para abordar el problema de la inserción de nuevos datos en la BD que cumplan todas las especificaciones del UD lo que produce una pérdida de semántica significativa. En particular, este problema se ve incrementado cuando existe obligatoriedad en la participación de las entidades asociadas a una interrelación ya que provocan inserciones en cascada. Aunque en este trabajo de tesis doctoral se ha presentado una solución teórica para este problema, en un trabajo futuro se pretende añadir esta propuesta a un módulo de la herramienta generadora el cual desarrolle el código necesario para implementar nuestra solución.
- En general, la elaboración de herramientas interactivas con los usuarios deben de incorporar, para su validez, algunas métricas para medir el nivel de la

realización de los objetivos planteados para construir estas herramientas. Por lo tanto, con la ampliación del dominio de la propuesta considerando más restricciones y más funcionalidades dentro de las herramientas propuestas, nos planteamos escoger un escenario real y realizar un conjunto de experimentos para medir el grado de satisfacción de los diseñadores con la utilización y con los resultados obtenidos por estas herramientas.

- Otra de las líneas futuras se dirige hacia la fase conceptual, concretamente, la propuesta presentada puede ayudar a profundizar más en la inclusión de propiedades deseables que debe cumplir un esquema conceptual según la perspectiva de la calidad del esquema [Piattini and Díaz, 2000]. La calidad de un esquema conceptual implica intentar obtener un esquema completo, correcto, y sencillo independiente del SGBD donde vaya a ser implementado [Moody, 1998]. En la consecución de estas propiedades van orientadas algunas de las líneas futuras de este trabajo. Se trate de estudiar la completitud del esquema conceptual que requiere incluir todos los aspectos relevantes tanto estáticos como dinámicos del UD. Lo que implica que el lenguaje de modelado conceptual debe permitir la descripción de todos los aspectos más relevantes del UD a modelar. Por lo tanto, pretendemos en este contexto realizar de forma automática una validación sintáctica de las restricciones de integridad, representadas en un esquema conceptual, el objetivo principal es mejorar la transformación del esquema conceptual al esquema relacional evitando la redundancia y la contradicción de estas restricciones.

- El mayor problema de los sistemas activos es validar y verificar el conjunto de reglas y el comportamiento activos de estas. La mayoría de los estudios realizados están dirigidos a satisfacer estas características. Por lo tanto, como una línea futura dirigida al marco de la validación y la verificación de los sistemas activos y el comportamiento activo nos gustaría ampliar el estudio para incluir los siguientes aspectos:

  - Actualmente UML [UML, 2006] no proporciona una extensión para la modelización del comportamiento activo de los sistemas activos, es decir, no

se pueden introducir los disparadores en el diagrama de clases de UML como una operación más. Pero, según lo que se ha explicado a la largo de este trabajo de tesis doctoral acerca del comportamiento de los disparadores que tiene que ser verificado. Por lo tanto, un de los trabajos futuros que se pretende realizar es crear una extensión de UML para modelar gráficamente los disparadores que facilite a los diseñadores de BDs implementar y verificar el comportamiento de aplicaciones activas con menos esfuerzo. A través de un interfaz gráfico de acuerdo a esta extensión, los desarrolladores pueden asociar disparadores a sus correspondientes clases y verificar simultáneamente, el comportamiento de estos disparadores utilizando los diagramas de secuencias del UML.

- En el campo de los sistemas activos relacionales no hemos encontrado un depurador comercial que permite la comprobación del comportamiento activo de los disparadores en tiempo de la ejecución (*run-time)*. Por lo tanto, creemos que sería interesante estudiar la posibilidad de construir depuradores de los sistemas activos relacionales. Estos depuradores pueden mejorar el análisis estático realizado en este trabajo, considerando la influencia de las condiciones de los disparadores en el comportamiento durante el tiempo de ejecución. Mostrando a su vez, las interacciones entre estos disparadores y los distintos objetos de la BD de una manera dinámica.
- Ampliar el estudio de la verificación y validación de los disparadores para detectar inconsistencias y anomalías en su comportamiento. Los problemas asociados con la ejecución de los disparadores necesitan más investigación para determinar otros efectos secundarios indeseados, que pueden venir dados por las interacciones de estos disparadores con el resto de las aplicaciones. Incluiremos dentro de este estudio distintos entornos como por ejemplo, BDs distribuidas ó BDs vía Web, y también, distintas arquitecturas, como la de tres capas, y haremos las comparaciones necesarias para su validación y verificación.

- La implementación de aplicaciones de gran cantidad de disparadores puede llevar consigo un nivel de complejidad muy alto y un impacto significativo sobre el rendimiento del SGBD. Como hemos explicado en este trabajo, existe una relación entre la semántica recogida en BD, la complejidad de implementación utilizada para conservar la semántica y el rendimiento del SGBD. En nuestra propuesta dentro de este contexto se ha presentado un conjunto de métricas para relajar las restricciones de cardinalidad de participación obligatoria. Por eso, creemos necesario definir más métricas que puedan ayudar a los administradores de la BD a relajar el control de otras restricciones de integridad, con el objetivo de ahorrar recursos y aumentar el rendimiento de la BD.

Bibliografía

[Aiken et al., 1995]. A. Aiken, J.M Hellerstein., J. Widom. "Static analysis techniques for Predicting the Behavior of Active Database Rules"; ACM Transactions on Database Systems, 20(1), March 1995.

[Al-Jumaily et al., (a) 2003]. H.T. Al-Jumaily, D. Cuadra, P. Martínez. "PANDORA CASE TOOL: Generating Triggers For Cardinality Constraints In RDBMS". In the IADIS International Conference, Portugal, 2003.

[Al-Jumaily et al., (a) 2004]. H.T. Al-Jumaily, D. Cuadra, P. Martínez. "Applying a fuzzy approach to relaxing cardinality constraints". In the 15th International Conference on Database and Expert Systems Applications, Zaragoza, Spain, 2004.

[Al-Jumaily et al., (b) 2003]. H.T. Al-Jumaily, D. Cuadra, P. Martínez. "Incorporando técnicas activas para la conservación de semántica en la transformación de esquema", VIII Jornadas de Ingeniería del Software y Bases de Datos 12-14 Noviembre, Alicante 2003.

[Al-Jumaily et al., (b) 2004]. H.T. Al-Jumaily, D. Cuadra, P. Martínez. "Plugging Active Mechanisms To Control Dynamic Aspects Derived From The Multiplicity Constraint In UML". In the 7th International Conference on the Unified Modelling Language, Portugal, 2004.

[Al-Jumaily et al., 2002]. H.T. Al-Jumaily, D. Cuadra, P. Martínez. "An Execution Model For Preserving Cardinality Constraints in The Relational Model". 4th Int. Conf. On Enterprise Information Systems. Spain 2002.

[Al-Jumaily et al., 2006]. H.T. Al-Jumaily, C. Pablo, D. Cuadra, P. Martínez. "Using UML's Sequence Diagrams as Termination Analyzer for Triggers-Based Executing". In the 23rd British National Conference on Databases, Queen's University Belfast, Northern Ireland, 18-20 July, 2006.

[Amghar et al., 2000]. Y. Amghar, M. Meziane, A. Flory. "Using Business Rules within a Design Process of Active Database", LISI – InstiNational des Scienes Appliquees, 2000.

[Bailey y Ramamohanarao, 1995]. J. A. Bailey, K. Ramamohanarao. "Issues in Active Databases", Proc.of 6th Australian Database Conf. Australian, 1995.

[Balaban y Shoval, 2002]. M. Balaban and P. Shoval. "MEER- An EER model enhanced with structure methods". Information Systems 27, pp 245-275, 2002.

[Baralis et al., 1995]. E. Baralis, S.Ceri, S.Paraboschi. “Improved Rule Analysis by Means of Triggering and Activation Graphs"; Proc. of 2nd Int. Workshop on Rules in Database Systems, Athens, Greece, September 1995.

[Baralis et al., 1996]. E. Baralis, S. Ceri, S. Paraboschi. “Modularization Techniques for Active Rule Design"; ACM Transactions on Database Systems, 21(1), March 1996.

[Baralis et al., 1998]. E. Baralis, S. Ceri, S. Paraboschi. “Compile-Time and Runtime Analysis of Active Behaviors". IEEE Transactions on Knowledge and Data Engineering, 1998.

[Baralis y Widom, 2000]. E. Baralis and J. Widom. “Better Static Rule Analysis for Active Database Systems”, ACM Transactions on Database Systems, 2000.

[Batín et al. 1994]. C. Batini, S. Ceri, S. Navathe. “Diseño Conceptual de Bases de Datos: Un enfoque de entidades interrelaciones”.Addison-Wesley / Díaz de Santos. 1994.

[Batra y Antony, 1994]. D. Batra, S. R. Antony. “Novice Errors in Conceptual Database Design", European Journal of Information Systems", Vol. 3, 1994.

[Batra y Zanakis, 1994]. D. Batra and H. Zanakis. “A Conceptual Database Design Approach Based on Rules and Heuristica", European Journal of Information Systems", Vol. 3, Nº 3, 228-239, 1994.

[Behrends, 1994]. H. Behrends. “Simulation-based Debugging of Active Databases” Proc. of the 4th Intl. Workshop on Research Issues in Data Engineering: Active Database Systems, Houston, February 1994.

[Benazet et al., 1995]. E. Benazet, H. Guehl, M. Bouzeghoub. “VITAL: A Visual Tool for Analysis of Rules Behaviour in Active Databases". Proc. of 2nd Int. Workshop on Rules in Database Systems, Athens, Sept. 1995.

[Berghe, 2000]. T.V. Berghe. “A Methodological Framework for Active Application Development”. Institut d’Administration et de Gestion, 2000.

[Bertino et al., 1997]. E. Bertino, B. Catania, S. Bressan. “Integrity constraint checking in Chimera” proc. Of the 2nd Int. Workshop on Constraints Database Systems, Delphi, Greece. 1997.

[Bloor, 2001]. Bloor Research. “Databases an evaluation & comparison", 2001. Accesible en: www-3.ibm.com/software/data/pubs/pdfs/bloor.pdf

[Booch, 1994]. G. Booch. “Object Oriented Analysis and Design with Application", Second Edition Benjamin Cummings, 1994.

[Booch, 2005]. G. Booch. “The unified modeling language user guide", Second Edition Addison-Wesle, 2005.

[Boral y DeWitt, 1984]. H. Boral, D.J. DeWitt. “A Methodology For Database System Performance Evaluation", In Proc. of the 1984 SIGMOD Int. Conf., Boston, June 1984.

[Bouzeghoud y Metais, 1991]. E. Bouzeghoud and E. Metais. “Semantic modeling and object modeling: two complementary paradigms”. Entity-Relationship Approach (ER’91) pp. 325-348, North-Holland, Amsterdam, 1991.

[Bruce, 1992]. T. Bruce. “Designing Quality Databases with IDEFIX Information Models", New York: Dorset House, 1992.

[CA, 2006]. Computer Associates International Inc. Erwin Data Modeller, v. 4.1.2522. www.ca.com.

[Ceri et al., 1994]. S. Ceri, P. Fraternali, S. Paraboschi, L. Tanca. “Automatic Generation of Production Rules for Integrity Maintenance", ACM Transaction on Database Systems, Vol. 19, No. 3, September 1994.

[Ceri et al., 1995]. S. Ceri, E. Baralis, P. Fraternali, S. Paraboschi. “Design of Active Rule Applications: Issues and Approaches", In Proc. Of the Int'1 conf. Deductive and Object-oriented Database. 1995.

[Ceri et al., 1997]. S. Ceri, P. Fraternali, S. Paraboschi. “The IDEA Tool Set", Proceedings of the 13th Int. Conf. on Data Engineering, 1997.

[Ceri et al., 2000]. S. Ceri, R. J. Cochrane, J. Widom. "Practical Applications of Triggers and Constraints: Successes and Lingering Issues". Proceedings of the 26th VLDB Int. Conf., Cairo, Egypt, 2000.

[Ceri y Fraternalli, 1997]. S. Ceri, P. Fraternalli. “Designing database applications with objects and rules: the IDEA Methodology”. Addsion-Wesley, 1997.

[Ceri y Manthey, 1994]. S. Ceri, R. Manthey. “Chimera: A Model and Language for Active DOOD Systems", Proc. of the 2nd Int. Conf. East/West Database Workshop, Klagenfurt, Austria, 25-28 September 1994.

[Ceri y Widom, 1990]. S. Ceri., J. Widom. “Deriving Production Rules for Constraint Maintenance”, IBM Almaden Reserch Center, Proc. VLDB Conf. 1990.

[Chan, et al., 1997]. H.W.R. Chan, S.W. Dietrich, S.D. Urban. “On Control Flow Testing of Active Rules in a Declarative Object-Oriented Framework”. Proc. of 3rd Intl. Workshop on Rules in DatabaseSystems, RIDS 97, Skovde, Sweden, June 1997.

[Chen, 1976]. P. Chen. “The Entity-Relationship Model – Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, N. 1. 1976.

[Cochrane et al., 1996]. R. Cochrane, H. Pirahesh, N. Mattos. "Integrating Triggers and Declarative Constraints in SQL Database Systems". Proc. 22nd VLDB Int. Conf., India 1996.

[Coleman et al., 1994]. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. "Object Oriented Development: The Fusion Method". Prentice-Hall International, Englewood Cliffs, New Jersey, 1994.

[Cook y Daniels, 1994]. S. Cook, J. Daniels. “Designing Object Systems: Object-Oriented Modelling with Syntropy". Prentice- Hall International, New York 1994.

[Couchot, 2001]. A. Couchot. “Improving Active Rules Termination Analysis by Graphs Splitting", Fifth East-European Conf. on Advances in Databases and Information Systems. Vilnius, Lithuania. September 25-28, 2001.

[Coupaye et al., 1994]. T. Coupaye, C. Collet, and T. Svensen. “NAOS: Efficient and Modular Reactive Capabilities in an Object-Oriented Database System”, Proc. of the 20th VLDB Conf., Santiago, Chile, September 1994.

[Coupaye y Collet, 1998]. T. Coupaye, C. Collet. “Semantics Based Implementation of Flexible Execution Models for Active Database Systems", 14ièmes Journées Bases de Données Avancées (BDA'98), Tunisia, October 1998.

[Cuadra et al., 2002]. D. Cuadra, C. Nieto, E. Castro, P. Martínez, M. Velasco. "Preserving relationship cardinality constraints in relational schemata". Database Integrity: Challenges and Solutions, Ed: Idea Group Publishing, 2002.

[Cuadra, 1999] D. Cuadra, C. Nieto, P. Martínez, A. Miguel, “Control de Restricciones de cardinalidad en una Metodología de Desarrollo de Bases de Datos Relacionales”, Novática, 1999.

[Cuadra, 2003]. D. Cuadra. “Aproximación Formal a las Restricciones de Cardinalidad en un Marco Metodológico de Desarrollo de Bases de Datos Relacionales”, Tesis Doctoral, Departamento De Informática, Universidad Carlos III De Madrid, 2003.

[Date y Darwen, 1997] C.J. Date, H. Darwen. “A Guide to the SQL STANDARD: A user's guide to the standard database language SQL”. Addison-Wesley, Fourth Edition. 1997.

[Date, 1990] C.J. Date. “A Contribution to the Study of Database Integrity”, In Relational Database: Writings 1985-1989. Addison-Wesley, Pg. 185-215. 1990.

[Dayal, et al., 1996]. U. Dayal, A. P. Buchmann, S. Chakravarthy. “The HiPAC Project” in “Active Database Systems; Triggers and Rules for Advanced Database Processing", Morgan Kaufmann Publishers, pp. 177-205. 1996

[DB2, 2005]. IBM DB2 Universal Database Enterprise Server Edition V8.2. www-306.ibm.com/software/. 2005.

[De Miguel et al., 1999]. A. Miguel, M. Piattini, E. Marcos. “Diseño de Bases de Datos Relacionales”. RAMA Ed, 1999.

[Debray y Hickey, 2000]. S. Debray, T. Hickey. “Constraint-Based Termination Analysis for Cyclic Active Database Rules”. Proc. DOOD'2000: 6th. Int. Conf. on Rules and Objects in Databases, July 2000.

[Decker, 2002]. H. Decker. “Translating Advanced Integrity Checking Technology to SQL”, in Database Integrity: Challenges & Solutions. IDEA Group Publishing, 2002.

[Díaz y Jaime, 1997]. O. Díaz, A. Jaime. "EXACT: An Approach to active object-oriented databases", The VLDB Journal 6: 282-295. 1997.

[Díaz, 1996]. O. Díaz. "The operational semantics of user-defined relationships in object oriented database systems", Data & Knowledge Engineering 16, 1995.

[Díaz, 1998]. O. Díaz. "EXACT: An Extensible Approach to Coping with Heterogeneous Rule Execution Models". In N.W. Paton. "Active Rules in Database Systems", Springer-Verlag, New York, 1998.

[Díaz, et al., 1993]. O. Díaz, A. Jaime, N. Paton. "DEAR a DEbugger for Active Rules in an object oriented context". In M. Williams N. Paton editor Rules In Database Systems pages 180-193 LNCS Springer Verlag 1993.

[Dittrich et al., 1996]. K.R. Dittrich, S. Gatziu, A. Geppert; "The Active Database Management System Manifesto: A Rulebase of ADBMS Features"; A joint Report by ACT-NET Consortium. ACM Sigmod Record 25 (3), 1996.

[Eisenberg y Melton, 2004]. A. Eisenberg and J. Melton. "SQL: 2003 has been published**",** ACM SIGMOD Record, Volume 33, Issue 1, March 2004.

[Elizondo, 1998]. A. J. Elizondo. "Reglas activas: soporte y manejo en las bases de datos orientadas a objetos", tesis doctoral Universidad de País Vasco, 1998.

[Elmasri y Navathe, 2004]. R. Elmasri and S. Navathe. "Fundamentals of Database Systems", Third Edition, Addison-Wesley, 2004.

[Fahrner y Vossen, 1995]. C. Fahrner and G. Vossen. "A survey of database design transformations based on the Entity-Relationship model". Data & Knowledge Engineering. 15, 213-250. 1995.

[Faria y Vidal, 1999]. J.P. Faria and R.M. Vidal. "Data-Driven Active Rules for the Maintenance of Derived Data and Integrity Constraints in User Interfaces to Databases", XIV Simpósio Brasileiro De Banco De Dados, BRASIL Outubro de 1999.

[Ferg, 1991]. S. Ferg. "Cardinality Concepts in Entity-Relationship modeling", in Proc. 10th Int. Conf. On Entiy-Relationship Approach, San Mateo, pp. 1-30, Oct. 1991.

[Fernandes, 1997]. A.A. Fernandes, M. H. Williams, N.W. Paton. "A Logic-Based Integration of Active and Deductive Databases", New Generation Computing 15(2), 1997.

[Fraternali y Tanca, 1995]. P. Fraternali and L. Tanca. "A Structured Approach for the Definition of the Semantics of Active Database Systems", ACM Transactions on Database Systems, 20(4), December 1995.

[Gatziu y Dittrich, 1998]. S. Gatziu and K.R. Dittrich. "SAMOS"; In N.W. Paton. "Active Rules in Database Systems", Springer-Verlag, New York, 1998.

[Gehani, 1991]. N.H. Gehani and H.V. Jagadish. "Ode as an Active Database: Constraints and Triggers". Proceedings of the 17th VLDB Conf., September, 1991.

[Geppert et al., 1995]. A. Geppert, S. Gatziu, K. R. Dittrich. “A Designer’s Benchmark for Active Database Management Systems: OO7 Meets the BEAST”, In Proc. 2nd Workshop on Rules in Databases (RIDS), Athens, Greece, September 1995.

[Geppert et al., 1998]. A. Geppert, M. Bernatsson, D. Lieuwen, C. Roncancio. “Performance of Object-Oriented Active Database Systems Using the BEAST Benchmark", Theory and Practice on object Systems, Vol 4(3), 1998.

[Geppert y Dittrich, 1995] A. Geppert, K.R. Dittrich. “Specification and Implementation of Consistency Constraints in Object-Oriented Database Systems”: In proc. of the Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), GI-Fachtagung, Dresden, 1995. Informatik Aktuell, Springer 1995. Pg. 322-337.

[Gertz y Lipeck, 1993]. M. Gertz and U.W. Lipeck. “Deriving Integrity Maintaining Triggers from Transaction Graphs” in proc. 9th ICDE, IEEE Computer Society Oress, 1993.

[Gertz, 1994]. M. Gertz. “Specifying Reactive Integrity Control for Active Databases”, Proc. Of RIDE’94, Houston, Texas, 1994.

[Guerrini y Montrsi, 1997]. G. Guerrini and D. Montrsi. “Design and Implementation of Chimera Active Rule Language", Data & Knowledge Engineering 24, 1997.

[Hanson, 1996]. E.N. Hanson. “The Design and Implementations of Ariel Active Database Rule System", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No.1, February 1996.

[Horowitz, 1994]. B. M. Horowitz. “Immediate States as a Source of Non-deterministic Behavior in Triggers", In Proc. 4th Int. Workshop on Research Issues in Data Engineering: Active Database Systems, 1994.

[Huang, 1979]. J.C. Huang. “Detection of dataflow anomaly through program instrumentation”, IEEE Transactions on Software Engineering, 1979.

[Iglesias et al., 2002]. A. Iglesias, D. Cuadra, E. Castro, P. Martinez. “Integrating Intelligent Methodological and Tutoring assistance in a CASE platform: the PANDORA experience", IS2002 Conf. Cork, Irlanda, Junio, 2002.

[Ishakbeyoglu y Ozsoyoglu, 1998]. N.S. Ishakbeyoglu and Z.M. Ozsoyoglu. “Maintenance of implication integrity constraints under updates to constraints”, The VLDB Journal 7: 67–78, 1998.

[ISO, 1998]. ISO 14598-1. “Information Technology – Evaluation of Software Products – General Guide”. ISO, 1998.

[Jones y Song, 1996]. T.H. Jones and Y. Song. “Analysis of Binary/Ternary Cardinality Combinations in Entity-Relationship Modeling", Data & Knowledge Engineering Vol. 19, No. 1 , 39-64. 1996.

[Jones, Song, 1996] T.H. Jones, II-Y Song. “Analysis of Binary/Ternary Cardinality Combinations in Entity-Relationship Modeling”, Data & Knowledge Engineering Vol. 19, No. 1, 39-64. 1996.

[Jorgensen, 2002]. P.C. Jorgensen. “Software Testing; A Craftsman’s Approach”, 2nd Edition, CRC Press, 2002.

[Kim y Chakravarthy, 1995]. S.K. Kim and S. Chakravarthy. “A Confluent Rule Execution Model for Active Databases”. Tech. Report, October 1995.

[Kulkarni, 1998]. K. Kulkarni, N. Mattos, R. Cochrane. “Active Database Features in SQL3". In “Active Rules in Database Systems”, Springer-Verlag, New York, 1998. .

[Lazarevic y Misic, 1991]. B. Lazarevic and V. Misic. “Extending the entity-relationship model to capture dynamic behaviour”. European Journal Information Systems 1 (2) pp. 95-106. 1991.

[Lee y Ling, 1999]. S.Y. Lee and T.W. Ling. “Unrolling Cycle to Decide Trigger Termination", Proc. 25th VLDB Conf. Scotland, 1999.

[Mannila y Räihä, 1992] H. Mannila and K.J. Räihä. “The Design of Relational Databases“, Addison-Wesley, 1992.

[Markowitz, 1990]. V.M. Markowitz. “Referential Integrity Revisited: An Object-Oriented Perspective", Proc. 16th, VLDB conf. Brisbane. Austraila 1990.

[McCabe, 1979]. T.J. McCabe. “A Complexity metric”, IEEE Transactions on Software Engineering, Decembre, 1979.

[Melton y Simon, 2002]. J. Melton and A.R. Simon. “SQL: 1999 Understanding Relational Language Components", Morgan Kaufmann Publishers, 2002.

[Microsoft, 2005]. Microsoft Corporation. SQL Server 2005 www.microsoft.com/SQL/

[Miguel y Piattini, 1999]. A. Miguel y M. Piattini. “Fundamentos y Modelos de Bases de Datos", RAMA Ed, 1999.

[Miller, 1977]. E.F. Miller. “Tutorial: Program Testing Techniques”, COMPSAC ’77 IEEE Computer Society, 1977.

[Mokrane et al., 2000]. B. Mokrane, K. Zoubida, M. Elisabeth. “CASE Tools: Computer Support for Conceptual Modeling / Advanced database technology and design", Editorial Artech House, 2000.

[Montesi y Torlone, 2002]. D. Montesi, R. Torlone. “Analysis and optimization of active databases", Data & Knowledge Engineering 40, 2002.

[Moody, 1998] D.L. Moody. “Metrics for evaluating the quality of entity relationship models”. Conceptual Modeling - ER'98: Proc. 17th International Conference on Conceptual Modeling, pp: 211-225, Nov. 1998.

[Motro, 1989]. A. Motro. “Integrity = Validity + Completeness". ACM Transactions on Database Systems, 14(4): 480-502, December 1989.

[Ocelot, 2003]. Ocelot Computer Services Inc. The Ocelot SQL DBMS www.ocelot.ca

[Olivé, 2003] A. Olivé. “Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages”. Conceptual Modeling - ER 2003, 22nd Int. Conf. on Conceptual Modeling, Chicago, IL, USA, October 13-16, 2003

[OMG, 2006] OMG: Object Management Group, “Model Driven Architecture” http://www.omg.org/mda

[Oracle, 2005]. Oracle Corporation. Oracle Database 9i. www.oracle.com

[Pacheco, 1997]. M. A. Pacheco. “Dynamic integrity constraints definition and enforcement in databases: a classification framework”, Proc. of the First Working Conf. on Integrity and Internal Control in Information Systems, Zürich, Switzerland, Dec. 1997.

[Pastor, 2004]. Mª.Á. Pastor. “Análisis de restricciones en el nivel conceptual: generación automática de transacciones seguras”, Tesis doctoral del departamento de sistemas informáticos y computación, universidad politécnica de valencia, 2004.

[Paton et al., 1994]. N. Paton, O. Diaz, H. William, J. Campin, A. Dinn, and A. Jaime. "Dimensions of Active behaviour", In Proc. 1st Int. Wshp. On Rules In Database Systems, 1994.

[Paton y Díaz, 1999]. N. W. Paton and O. Díaz. “Active Database Systems", ACM Computing Surveys, Vol.31, No.1, 1999.

[Paton, 1989]. N. Paton. “ADAM: An Object-Oriented Database System implemented in Prolog", Proc. of the British National Conf. on Databases, Cambridge University Press, 1989.

[Paton, 1998]. N.W. Paton. “Active Rules in Database Systems”, Springer-Verlag, New York, 1998.

[Philippe et al., 1999]. A. Philippe, D. Marc, R. Philippe. “Using Active Database Mechanisms to Build Co-operative Application", Society for design and Process Science, 1999.

[Piattini and Díaz, 2000] M. Piattini and O. Díaz. “Advanced Database Technology and Design”. Ed. Artech House, Boston-London, 2000.

[Piattini, et al., 1996]. M. Piallini, J.A. Calvo-Manzano, J. Cervera, L. Fernández. “Análisis y diseño detallado de Aplicaciones Informáticas de Gestión”, RAMA Ed, 1996.

[PMI, 2005]. Project Management Institute, http://www.pmi.org

[Pressman, 2003] R.S. Pressman. “Ingeniería del Software un Enfoque Practico”, McGraw-Hill Interamerican, 5ª ed, 2005.

[Rational, 2003]. Rational Rose. Rational Enterprise Edition V 2003.06.00.436.000. www-306.ibm.com/software/rational/

[Rochfeld y Tardieu, 1983]. A. Rochfeld and H. Tardieu. “MERISE: An information system design and development methodology", Information & Management, 6, 1983.

[Rumbaugh, 1991]. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. “Object-Oriented Modelling and Design", Prentice-Hall, 1991.

[Rumbaugh, 2005]. J. Rumbaugh. “The unified modeling language reference manual ", Addison-Wesle, 2005.

[SEI, 2006] Software Engineering Institute, http://www.sei.cmu.edu

[Segev y Zhao, 1994]. A. Segev and J.L. Zhao. “Rule Management in Expert Database Systems", Management Science Vol. 40, No. 6, June 1994.

[Simon y Kotz-Dittrich, 1995]. E. Simon and A. Kotz-Dittrich. “Promises and Realities of Active Database Systems", In Proc of the 21st VLDB Conf. Zúrich, Switzerland, 1995.

[Sommerville, 2005]. I. Sommerville. “Ingeniería del Software”, Pearson Addison Wesle, 2005.

[Song et al., 1995]. Y. Song, M. Evans, E.K. Park. “A Comparative Analysis of Entity-Relationship Diagrams", Journal of Computer & Software Engineering, 1995.

[Stonebraker y Kemnitz, 1991]. M. Stonebraker, G. Kemnitz. "The POSTGRES next-generation database management system". Communications of the ACM, October 1991.

[Storey, 1993]. V.C. Storey. "Understanding Semantic Relationships", VLDB Journal, 2, 455-488, Fred J. Maryanski, Editor, 1993.

[Summit, 1996]. Summit Software Company. “Basic Script 2.25 Language Reference”, 1996.

[Tardieu et al., 1983]. H. Tardieu, A. Rochfeld, R. Coletti. “La Méthode MERISE. Tome 1: Principles et Outils. Les Editions d’Organisation“, Paris 1983.

[Teorey et al., 1986]. T.J. Teorey, D. Yang, J. Fry. “A Logical Design Methodology for Relation Databases Using the Extended Entity-Relationship Model". Computer Surveys, Vol. 18. No. 2. 1986.

[Teorey, 1999]. T.J. Teorey. “Database Modeling & Design” third edition, Morgan Kaufmann Series in data management systems, 1999.

[Türker y Gertz, 2000].. C. Türker, M. Gertz. “Semantic Integrity Support in SQL-99 and Commercial (Object) Relational Database Management Systems”. UC Davis Computer Science Technical Report CSE-2000-11, University of California, October 2000. Accesible en: http://www.db.cs.ucdavis.edu/papers/TG00.pdf

[UML, 2006] Object Management Group, Unified Modeling Language http://www.uml.org/

[Urban y Delcambre, 1990]. S.D. Urban and L.M. Delcambre. “Integrity Analysis: a design process for specifying operations on objects", IEEE Transaction on Knowledge and Data Engineering, 1990.

[Urrutia, et al., 2003]. A. Urrutia, J. Galindo, M. Piattini. “Restricciones de Participación y Tipo de Correspondencia Difusa en un Modelo Conceptual". Revista electrónica ingeniería informática, número 9, año 2003, Agosto 2003.

[Vaduva, 1998]. A. Vaduva. “Rule Development for Active Database Systems", PhD thesis, University of Zurich, 1998.

[Widom y Ceri, 1996]. J. Widom, S. Ceri. “Active Database Systems; Triggers and Rules for Advanced Database Processing", Morgan Kaufmann Publishers, 1996.

[Widom y Finkelstein, 1990]. J. Widom, S.J. Finkelstein. “Set-Oriented Production Rules in Relational Database Systems", Proc. ACM-SIGMOD Int. Conf. 1990.

[Widom, et al., 1991]. J. Widom, R. J. Cochrane, and B. G. Lindsay. "Implementing set-oriented production rules as an extension to Starburst". Proc. 7th Int. Conf. on VLDB, September 1991.

[Winter, 1998]. R. Winter. “Design and implementation of derivation rules in information systems", Data & Knowledge Engineering 26, 1998.

Anexo I

Pruebas del camino básico

En este anexo se presentan las pruebas de camino básico que hemos realizado para la verificación de los disparadores generados por las herramientas propuestas. Las pruebas se han realizado sobre 9 secuencias de disparadores que contienen 18 disparadores, cada una de estas secuencias controla una restricción determinada (véase la tabla 5.1).

A continuación se presentan las pruebas de algunas secuencias para verificar el código de los mismos.

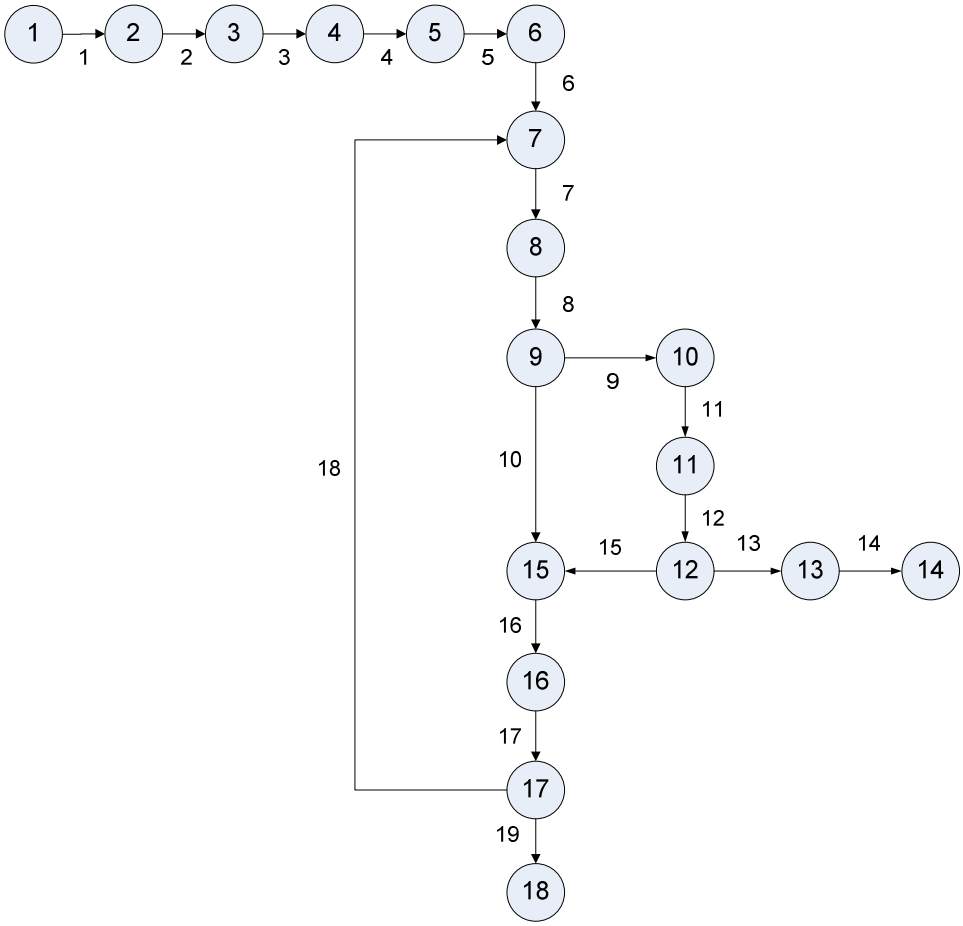
Los disparadores {T5, T6}

Se activan cuando se produce una operación de actualización (DELETE o UPDATE) sobre la tabla E2 (véase figura 5.3). Estos disparadores controlan las cardinalidades mínima y máxima de la entidad E2.

Numerar las sentencias del SQL

```
    CREATE OR REPLACE TRIGGER T5
    BEFORE DELETE OR UPDATE ON E2
    REFERENCING NEW AS NEW OLD AS OLD
    FOR EACH ROW
 1  BEGIN
 2  DATO1N_E2_E1.CNT_E2:= DATO1N_E2_E1.CNT_E2 + 1;
 3  DATO1N_E2_E1.OLD_ID_E1(DATO1N_E2_E1.CNT_E2):=:OLD.ID_E1;
 4  DATO1N_E2_E1.NEW_ID_E1(DATO1N_E2_E1.CNT_E2):=:NEW.ID_E1;
 5  END;
    CREATE OR REPLACE TRIGGER T6
    AFTER DELETE OR UPDATE ON E2
    DECLARE
    VAR_ID_E1 NUMBER:=0;
    NTUPLES1_ID_E1 NUMBER:=0;
    NTUPLES2_ID_E1 NUMBER:=0;
 6  BEGIN
 7  FOR I IN 1 .. DATO1N_E2_E1.CNT_E2 LOOP
 8  SELECT COUNT(*) INTO VAR_ID_E1 FROM E1
                WHERE ID_E1 = DATO1N_E2_E1.OLD_ID_E1(I);
 9  IF VAR_ID_E1 >0 THEN
10  SELECT COUNT(*) INTO NTUPLES1_ID_E1 FROM E2
          WHERE ID_E1 = DATO1N_E2_E1.OLD_ID_E1(I);
11  SELECT COUNT(*) INTO NTUPLES2_ID_E1 FROM E2
          WHERE ID_E1 = DATO1N_E2_E1.NEW_ID_E1(I);
12  IF NTUPLES1_ID_E1<1 OR NTUPLES2_ID_E1>3 THEN
13  RESET.DO ;
14  RAISE_APPLICATION_ERROR  (-20502,  '!!  WARNING  !!    CANNOT
    DELETE FK = '||DATO1N_E2_E1.OLD_ID_E1(I)||' FROM E2
    : BECAUSE A SEMANTIC LOSS WILL BE PRODUCED IN !! E1 !!' );
15  END IF;
16  END IF;
17  END LOOP;
18  END;
```

El diagrama del programa



Complejidad ciclomática

Ciclomático del diagrama V(G) = 19 – 18 + 2 = 3

Esto significa que existen al menos 3 caminos independientes en el diagrama y por lo tanto, nos indica la medida de los caminos que debemos comprobar.

Examinar los caminos

A continuación se muestran los resultados de los tres caminos básicos más significativos para probar la secuencia de disparadores {T5, T6}:

P1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18, 7, 8, 9, 15, 16, 17, 18}

Resultados esperados:

(f) Se introduce el valor OLD.Id_e1 de las claves asociadas a la tupla *t* de la tabla E2.

(g) Se verifica que el borrado de OLD.Id_e1 de la tabla E2 no afecta la semántica de cardinalidad mínima de la tabla E1. ⇒ ***Restricción verificada***.

(h) Se comprueba que no existen más tuplas para borrar.

(i) **Se acepta el borrado** de la tupla *t*.

(j) Se repiten los pasos de (a) – (d) en el caso de la modificación de los valores OLD.Id_e1 y New.Id_e1.

P2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

Resultados esperados:

(d) Se introduce el valor OLD.Id_e1 de la clave asociada a la tupla *t* de la tabla E2.

(e) Se verifica que el borrado de :OLD.Id_e1 de la tabla E2 no afecta la semántica de cardinalidad mínima de la tabla E2. ⇒ ***Restricción no verificada***.

(f) **Se rechaza el borrado** de la tupla *t*.

(g) Se repiten los pasos de (a) – (e) en el caso de la modificación de los valores OLD.Id_e1 y New.Id_e1.

P3 = {1, 2, 3, 4, 5, 6, 7, 8, 10, 16, 17, 19}

Resultados esperados: camino imposible porque los valores de la tupla borrada OLD.Id_e1 no existe en la tabla E2.

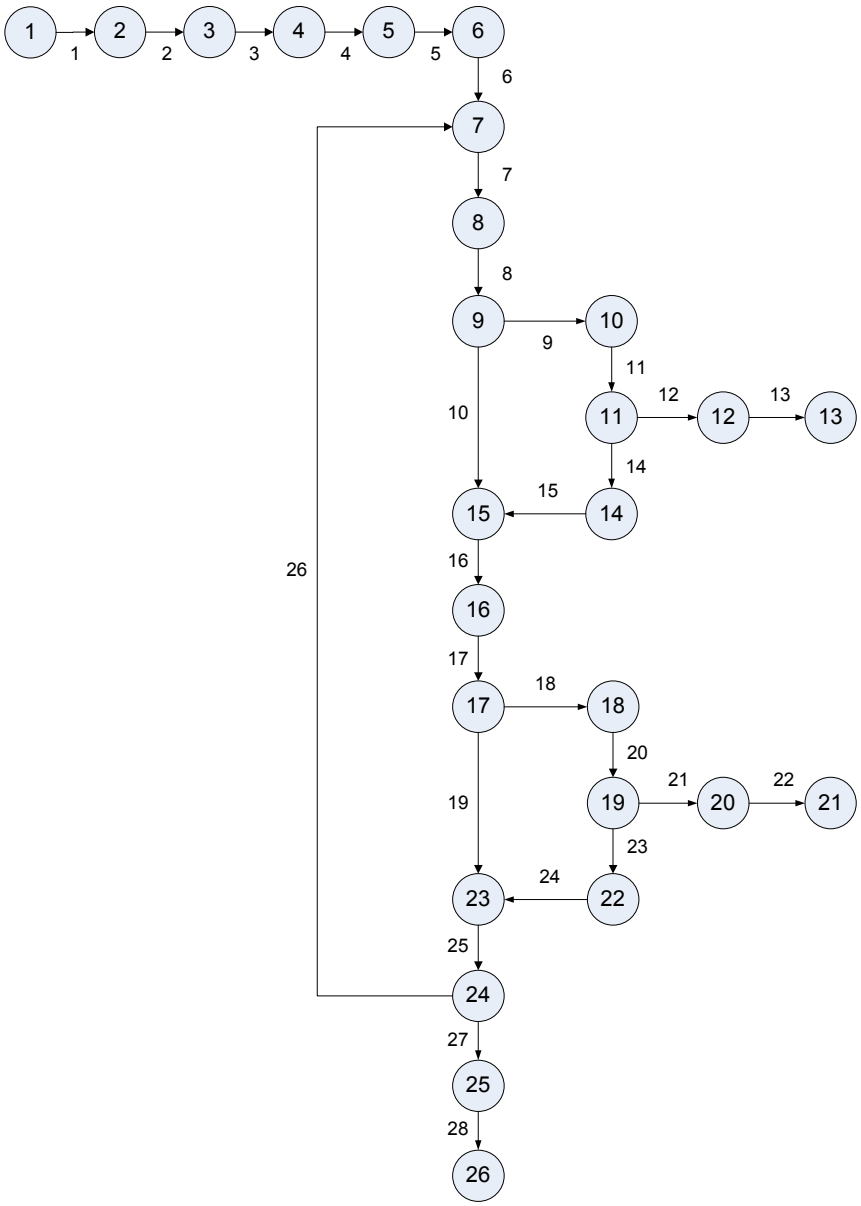
Los disparadores {T13, T14}

Se activan cuando se produce una operación de inserción sobre la tabla R4 (véase figura 5.3). Estos disparadores controlan las cardinalidades mínima y máxima de las entidades E3 y E4.

Numerar las sentencias del SQL

```
    CREATE OR REPLACE TRIGGER T13
    BEFORE INSERT ON R4
    REFERENCING NEW AS NEW OLD AS OLD
    FOR EACH ROW
 1  BEGIN
 2  DATONM_E4_E3.CNT_R4:= DATONM_E4_E3.CNT_R4 + 1;
 3  DATONM_E4_E3.OLD_ID_E4(DATONM_E4_E3.CNT_R4):=:NEW.ID_E4;
 4  DATONM_E4_E3.OLD_ID_E3(DATONM_E4_E3.CNT_R4):=:NEW.ID_E3;
 5  END;
    CREATE OR REPLACE TRIGGER T14
    AFTER INSERT ON R4
    DECLARE
    NTUPLES_ID_E4 NUMBER:=0;
    VAR_ID_E4 NUMBER:=0;
    NTUPLES_ID_E3 NUMBER:=0;
    VAR_ID_E3 NUMBER:=0;
 6  BEGIN
 7  FOR I IN 1 .. DATONM_E4_E3.CNT_R4 LOOP
 8  SELECT COUNT(*) INTO VAR_ID_E4 FROM E4
          WHERE ID_E4 = DATONM_E4_E3.OLD_ID_E4(I);
 9  IF VAR_ID_E4 >0 THEN
10  SELECT COUNT(*) INTO NTUPLES_ID_E4 FROM R4
          WHERE ID_E4 = DATONM_E4_E3.OLD_ID_E4(I);
11  IF NTUPLES_ID_E4>4 THEN
12  RESET.DO ;
13  RAISE_APPLICATION_ERROR  (-20502,  '!!  WARNING  !!    CANNOT
    DELETE FK = ' ||DATONM_E4_E3.OLD_ID_E4(I)||' FROM R4
    : BECAUSE A SEMANTIC LOSS WILL BE PRODUCED IN !! E4 !!' );
14  END IF;
15  END IF;
16  SELECT COUNT(*) INTO VAR_ID_E3 FROM E3
          WHERE ID_E3 = DATONM_E4_E3.OLD_ID_E3(I);
17  IF VAR_ID_E3 >0 THEN
18  SELECT COUNT(*) INTO NTUPLES_ID_E3 FROM R4
          WHERE ID_E3 = DATONM_E4_E3.OLD_ID_E3(I);
19  IF NTUPLES_ID_E3>7 THEN
20  RESET.DO ;
21  RAISE_APPLICATION_ERROR  (-20502,  '!!  WARNING  !!    CANNOT
    DELETE FK = '||DATONM_E4_E3.OLD_ID_E3(I)||' FROM R4
    : BECAUSE A SEMANTIC LOSS WILL BE PRODUCED IN !! E3 !!' );
22  END IF;
23  END IF;
24  END LOOP;
25  DATONM_E4_E3.CNT_R4:=0;
26  END;
```

El diagrama del programa



Complejidad ciclomática

Ciclomático del diagrama V(G) = 28 – 26 + 2 = 4

Esto significa que existen al menos 4 caminos independientes en el diagrama y por lo tanto, nos indica la medida de los caminos que debemos comprobar.

Examinar los caminos

A continuación se muestran los resultados de los cuatro caminos básicos más significativos para probar la secuencia de disparadores {T13, T14}:

P1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 14, 15, 16, 17, 18, 20, 23, 24, 26, 7, 8, 10, 16, 17, 19, 25, 27, 28}

Resultados esperados:

(a) Se introducen dos valores NEW.Id_e3 y NEW.Id_e4 de las claves asociadas a la tupla *t* de la tabla R4.

(b) Se verifica que la inserción de NEW.Id_e3 de la tabla R4 no afecta la semántica de cardinalidad mínima de la tabla E3. ⇒ ***Restricción verificada***.

(c) Se verifica que la inserción de NEW.Id_e4 de la tabla R4 no afecta la semántica de cardinalidad mínima de la tabla E4. ⇒ ***Restricción verificada***.

(d) Se comprueba que no existen más tuplas para insertar.

(e) **Se acepta la inserción** de la tupla *t*.

P2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13}

Resultados esperados:

(a) Se introducen dos valores NEW.Id_e3 y NEW.Id_e4 de las claves asociadas a la tupla *t* de la tabla R4.

(b) Se verifica que la inserción de : NEW.Id_e3 de la tabla R4 no afecta la semántica de cardinalidad mínima de la tabla E3. ⇒ ***Restricción no verificada***.

(c) **Se rechaza la inserción** de la tupla *t*.

P3 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 14, 15, 16, 17, 18, 20, 21}

(e) Se introducen dos valores NEW.Id_e3 y NEW.Id_e4 de las claves asociadas a la tupla *t* de la tabla R4.

(f) Se verifica que la inserción de NEW.Id_e3 de la tabla R4 no afecta la semántica de cardinalidad mínima de la tabla E3. ⇒ ***Restricción verificada***.

(g) Se verifica que la inserción de NEW.Id_e4 de la tabla R4 no afecta la semántica de cardinalidad mínima de la tabla E4. ⇒ ***Restricción no verificada***.

(h) **Se rechaza la inserción** de la tupla *t*.

P4 = {1, 2, 3, 4, 5, 6, 7, 8, 10, 16, 17, 19, 25, 27, 28}

Resultados esperados: camino imposible porque los valores de la tupla insertada (NEW.Id_e3, NEW.Id_e4) no existen en la tabla R4.

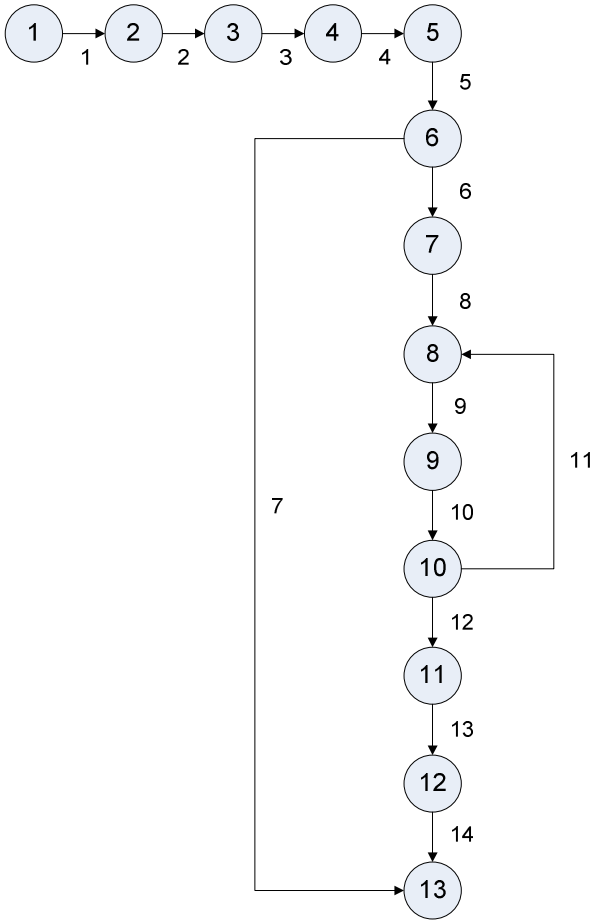
Los disparadores {T7, T8}

Se activan cuando se produce una operación de borrado sobre la tabla E5 (véase figura 5.3). Estos disparadores controlan la totalidad de la jerarquía R3 borrando el supertipo cuando se borra el subtipo correspondiente.

Numerar las sentencias del SQL

```
     CREATE OR REPLACE TRIGGER T7
     BEFORE DELETE ON E5
     REFERENCING NEW AS NEW OLD AS OLD
     FOR EACH ROW
1    BEGIN
2    DATOHY_E3.CNT_E3:= DATOHY_E3.CNT_E3 + 1;
3    DATOHY_E3.OLD_E5_ID_E3(DATOHY_E3.CNT_E3):=:OLD.ID_E3;
4    END;
     CREATE OR REPLACE TRIGGER T8
     AFTER DELETE ON E5
5    BEGIN
6    IF DATOHY_E3.CNT_E5 = 1 THEN RETURN; END IF;
7    DATOHY_E3.CNT_E5 := 1;
8    FOR I IN 1 .. DATOHY_E3.CNT_E3 LOOP
9    DELETE FROM E3 WHERE ID_E3
           = DATOHY_E3.OLD_E5_ID_E3(I);
10   END LOOP;
11   DATOHY_E3.CNT_E5 := 0;
12   DATOHY_E3.CNT_E3 := 0;
13   END;
```

El diagrama del programa



Complejidad ciclomática

Ciclomático del diagrama V(G) = 14 – 13 + 2 = 3

Esto significa que existen al menos 3 caminos independientes en el diagrama y por lo tanto, nos indica la medida de los caminos que debemos comprobar.

Examinar los caminos

A continuación se muestran los resultados de los cuatro caminos básicos más significativos para probar la secuencia de disparadores {T7, T8}:

P1 = {1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 13, 14}

Resultados esperados:

(a) Se introduce el valor OLD_e3 de la clave asociada a la tupla *t* de la tabla E5

(b) Se verifica el estado del disparador (DATOHY_E3.CNT_E5 = 0) significa que el disparador no ha sido activado antes. ⇒ ***Disparo permitido***.

(c) **Se acepta el borrado y se borra en cascada el supertipo** de la tupla *t*.

P2 = {1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 9, 10, 12, 13, 14}

Resultados esperados:

(a) Se introduce el valor OLD_e3 de la clave asociada a la tupla *t* de la tabla E5

(b) Se verifica el estado del disparador (DATOHY_E3.CNT_E5 = 0) significa que el disparador no ha sido activado antes. ⇒ ***Disparo permitido***.

(c) **Se acepta el borrado y se borra en cascada el supertipo** de la tupla *t*.

(d) Se repite los pasos de (a) – (c) para más de una tupla.

P3 = {1, 2, 3, 4, 5, 7, 13}

(a) Se introduce el valor OLD_e3 de la clave asociada a la tupla *t* de la tabla E5

(b) Se verifica el estado del disparador (DATOHY_E3.CNT_E5 = 1) significa que el disparador ha sido activado antes. ⇒ ***Disparo no permitido***.

(c) Salir no hacer nada.

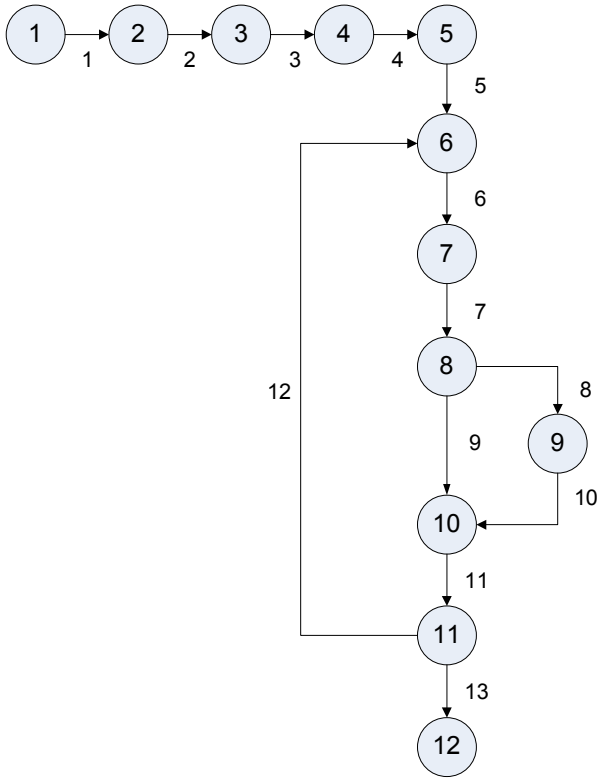
Los disparadores {T17, T18}

Se activa cuando se produce una operación de inserción sobre la tabla E6 (véase figura 5.3). Cada vez que se inserta en E6 la acción de T18 tiene que comprobar que no existe la clave primaria insertada en la otra tabla E5. Si está duplicada el disparador tiene que borrarla para que conserve la exclusividad de la interrelación.

Numerar las sentencias del SQL

```
     CREATE OR REPLACE TRIGGER T17
     BEFORE INSERT ON E6
     REFERENCING NEW AS NEW OLD AS OLD
     FOR EACH ROW
1    BEGIN
2    DATOHY_E3.CNT_E3:= DATOHY_E3.CNT_E3 + 1;
3    DATOHY_E3.NEW_E6_ID_E3(DATOHY_E3.CNT_E3):=:NEW.ID_E3;
4    END;
     CREATE OR REPLACE TRIGGER T18
     AFTER INSERT ON E6
     DECLARE
     VAR_ID_E3 E3.ID_E3%TYPE;
5    BEGIN
6    FOR I IN 1 .. DATOHY_E3.CNT_E3 LOOP
7    SELECT ID_E3 INTO VAR_ID_E3 FROM E6
           WHERE ID_E3= DATOHY_E3.NEW_E6_ID_E3(I);
8    IF VAR_ID_E3>0 THEN
9    DELETE FROM E5 WHERE ID_E3
           = DATOHY_E3.NEW_E6_ID_E3(I);
10   END IF;
11   END LOOP;
12   END;
```

El diagrama del programa



Complejidad ciclomática

Ciclomático del diagrama V(G) = 13 – 12 + 2 = 3

Esto significa que existen al menos 3 caminos independientes en el diagrama y por lo tanto, nos indica la medida de los caminos que debemos comprobar.

Examinar los caminos

A continuación se muestran los resultados de los cuatro caminos básicos más significativos para probar la secuencia de disparadores {T17, T18}:

P1 = {1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13}

Resultados esperados:

(a) Se introduce el valor NEW.Id_e3 de la clave asociada a la tupla *t* de la tabla E6.

(b) Se verifica que la inserción de NEW.Id_e3 no viola la semántica de exclusividad de la jerarquía R3. ⇒ ***Restricción violada***.

(c) **Se borra NEW.Id_e3 de la tabla E5**.

P1 = {1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 6, 7, 8, 10, 13}

Resultados esperados:

(a) Se introduce el valor NEW.Id_e3 de la clave asociada a la tupla *t* de la tabla E6.

(b) Se verifica que la inserción de NEW.Id_e3 no viola la semántica de exclusividad de la jerarquía R3. ⇒ ***Restricción violada***.

(c) **Se borra NEW.Id_e3 de la tabla E5**.

(d) Se repite los pasos de (a) – (c) para más de una tupla.

P3 = {1, 2, 3, 4, 5, 7, 9, 11, 13}

(a) Se introduce el valor NEW.Id_e3 de la clave asociada a la tupla *t* de la tabla E6.

(b) Se verifica que la inserción de NEW.Id_e3 no viola la semántica de exclusividad de la jerarquía R3. ⇒ ***Restricción no violada***.

(c) **Se acepta la inserción** de la tupla *t*.