FUNDAMENTOS DE BASES DE DATOS

NOTAS DE REFERENCIA

• Martha Elena Millán •

Universidad del Valle

Programa Editorial

La teoría de bases de datos incluye los principios formales para definir y manipular datos estructurados e interrelacionados. Para definir los datos se utiliza un modelo de datos y para su manipulación un lenguaje. Diferentes modelos de datos se han propuesto buscando un mayor nivel expresivo para representar el mundo real. La potencia y limitaciones de cada modelo se pueden evaluar desde un punto de vista teórico y se evidencian desde un punto de vista práctico cuando se trata de implementarlos en aplicaciones tradicionales y modernas. Estas últimas generalmente requieren tipos de datos complejos. Los lenguajes de manipulación de datos tienen como propósito ofrecer facilidad, simplicidad y flexibilidad a la hora de utilizarlos para actualizar y recuperar información desde la base de datos. Los lenguajes de manipulación son, en su gran mayoría, declarativos, lo que reduce significativamente el tiempo de desarrollo y mantenimiento de las aplicaciones. El propósito de este material es ofrecer a profesores responsables de la asignatura Fundamentos de Bases de Datos, y a los estudiantes, una guía que cubra el contenido completo de la asignatura. La estructura de este material se apoya en el texto guía de la asignatura y no intenta remplazarlo. Los lenguajes de manipulación de datos tienen como propósito ofrecer facilidad, simplicidad y flexibilidad a la hora de utilizarlos para actualizar y recuperar información desde la base de datos. Los lenguajes de manipulación son, en su gran mayoría, declarativos, lo que reduce significativamente el tiempo de desarrollo y mantenimiento de las aplicaciones. El propósito de este material es ofrecer a profesores responsables de la asignatura Fundamentos de Bases de Datos, y a los estudiantes, una guía que cubra el contenido completo de la asignatura. La estructura de este material se apoya en el texto guía de la asignatura y no intenta remplazarlo.

Programa Editorial

FUNDAMENTOS DE BASES DE DATOS

NOTAS DE REFERENCIA

Colección Ingeniería

MARTHA ELENA MILLÁN

Profesora titular de la Escuela de Ingeniería de Sistemas y Computación de la Facultad de Ingeniería de la Universidad del Valle. Doctor en Informática, Universidad Politécnica de Madrid; ha estado trabajando en el área de las Bases de Datos y es miembro activo del Grupo de Estudios Doctorales en Informática. Actualmente se desempeña como profesora de Bases de Datos en los programas de pregrado en Ingeniería de Sistemas, y en postgrado en la Maestría en Ingeniería con énfasis en Ingeniería de Sistemas y Computación y en el Doctorado en Ingeniería con énfasis en Ciencias de la Computación.

FUNDAMENTOS DE BASES DE DATOS
NOTAS DE REFERENCIA

Martha Elena Millán

Colección Ingeniería

Millán, Martha Elena
Fundamentos de bases de datos / Martha Elena Millán. --
Santiago de Cali: Programa Editorial Universidad del Valle, 2012
154 p. ; 24 cm. -- (Ciencias Naturales y Exactas)
Incluye bibliografía.
1. Bases de datos - Fundamentos 2. Bases de datos - Enseñanza 3.
Diseño de bases de datos I. Tít. II. Serie.
005.74 cd 21 ed.
A1335388

CEP-Banco de la República-Biblioteca Luis Ángel Arango

Universidad del Valle
Programa Editorial

Título: *Fundamentos de Bases de Datos - Notas de referencia*
Autora: Martha Elena Millán
ISBN: 978-958-765-002-0
ISBN PDF: 978-958-765-487-5
DOI:
Colección: Ingeniería
Primera Edición Impresa Marzo 2012
Edición Digital Julio 2017

Rector de la Universidad del Valle: Édgar Varela Barrios
Vicerrector de Investigaciones: Javier Medina Vásquez
Director del Programa Editorial: Francisco Ramírez Potes

Diseño de carátula: Anna Echavarria. Elefante
Diagramación y corrección de estilo: G&G Editores

Universidad del Valle
Ciudad Universitaria, Meléndez
A.A. 025360
Cali, Colombia
Teléfonos: (57) (2) 321 2227 - 339 2470
E-mail: programa.editorial@correounivalle.edu.co

Cali, Colombia - Julio de 2017

CONTENIDO

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La teoría de bases de datos incluye los principios formales para definir y manipular datos estructurados e interrelacionados. Para definir los datos se utiliza un modelo de datos y para su manipulación un lenguaje. Diferentes modelos de datos se han propuesto buscando un mayor nivel expresivo para representar el mundo real.

La potencia y limitaciones de cada modelo se pueden evaluar desde un punto de vista teórico y se evidencian desde un punto de vista práctico cuando se trata de implementarlos en aplicaciones tradicionales y modernas. Estas últimas generalmente requieren tipos de datos complejos.

Los lenguajes de manipulación de datos tienen como propósito ofrecer facilidad, simplicidad y flexibilidad a la hora de utilizarlos para actualizar y recuperar información desde la base de datos. Los lenguajes de manipulación son, en su gran mayoría, declarativos, lo que reduce significativamente el tiempo de desarrollo y mantenimiento de las aplicaciones.

Dada la naturaleza declarativa de los lenguajes de consulta, el desempeño del sistema depende, fundamentalmente, del proceso de optimización, que garantiza la generación del mejor plan de ejecución para una consulta dada. El optimizador de consultas utiliza algoritmos especializados para evaluar e implementar las diferentes operaciones que permiten expresar las consultas. Reglas de transformación lógicas y físicas se aplican para producir el mejor plan de ejecución.

Un importante polo de investigación ha girado en torno a los sistemas de bases de datos. Modelos y lenguajes de manipulación de datos, optimización de consultas son, entre otros, temas permanentes de investigación. Diferentes modelos de bases de datos se han propuesto, basados en lógica, dando lugar a las denominadas bases de datos deductivas. La incorporación

de la variable tiempo ha generado investigaciones en el área de las bases de datos temporales. Temas de cooperación e interoperabilidad han dado lugar a modelos de bases de datos interoperables.

Estudiar los fundamentos de las bases de datos es parte fundamental en la formación de estudiantes en niveles de postgrado. En particular, la asignatura Fundamentos de Bases de Datos forma parte integral del núcleo de asignaturas de Fundamentación Avanzada del Programa de Doctorado y Maestría en Ingeniería, énfasis en Ingeniería de Sistemas y Computación.

El propósito de este material es ofrecer a profesores responsables de la asignatura Fundamentos de Bases de Datos y a los estudiantes, una guía que cubra el contenido completo de la asignatura. La estructura de este material se apoya en el texto guía de la asignatura[1] y no intenta reemplazarlo.

El documento está estructurado en siete capítulos. Cada uno incluye una introducción al capítulo, una bibliografía básica anotada para el capítulo y una lista de artículos recomendados, algunos de los cuales son de lectura obligatoria.

Los artículos incluidos en la bibliografía anotada son referencias de lectura obligada para estudiantes de maestría y doctorado. Estos artículos seminales, entre otros, permiten que el lector pueda acceder de manera directa a las propuestas teóricas, conceptuales y metodológicas que han orientado el desarrollo del área de bases de datos. Se espera que su lectura y análisis les ofrezca a los estudiantes de postgrado la posibilidad de tomar una posición crítica y apoyada en la literatura especializada, frente a las propuestas de solución a problemas de investigación abordados en el área de las bases de datos.

Los artículos de lectura obligatoria tienen como propósito profundizar en los temas presentados e introducir a los estudiantes en tareas de revisión de la literatura especializada.

En el Capítulo 1 se presentan los modelos de datos y se describe un sistema de gestión de bases de datos.

El Capítulo 2 está dedicado al modelo relacional. Inicialmente, se define de manera formal el modelo relacional y luego se describen dos enfoques para definirlo: nombrado y no-nombrado. En este mismo capítulo se presentan los lenguajes de consulta relacionales haciendo énfasis en consultas conjuntivas.

En el Capítulo 3 se introduce el tema de optimización de consultas como un problema de búsqueda. Se describe el proceso de compilación de una consulta y se discuten algunos aspectos prácticos relacionados con los lenguajes de consulta.

En el Capítulo 4 se definen las dependencias funcionales y las reglas de

---

1 Abiteboul S., Hull R., Vianu V. Foundations of Databases. Addison-Wesley Publishing Company, 1995.

inferencia para generar otras dependencias implicadas a partir de las primeras. Las dependencias de inclusión, de unión y multivaluadas también son tratadas en este capítulo. Adicionalmente, el capítulo incluye el tema de diseño de bases de datos a partir de dos principales enfoques: basado en dependencias y basado en un modelo de datos más rico semánticamente.

El Capítulo 5 está dedicado a los modelos de gestión de bases de datos de tercera generación. Se introducen las bases de datos de objetos complejos como antecedente de los sistemas de gestión de bases de datos orientadas a objetos. Se presentan también los fundamentos del modelo relacional extendido.

Finalmente, en el Capítulo 6 se introducen dos temas que no forman parte del núcleo de la asignatura de Fundamentos de Bases de Datos pero que son, en este momento, de obligado conocimiento. Teniendo en cuenta la pérdida de la relación 1-a-1 con el cliente, que los negocios electrónicos han generado y la necesidad de recuperarla, en alguna forma, los *data warehouse (dwh)* y las técnicas de *data mining* surgen como alternativa de solución. Debido a que estos temas son en sí mismos objeto de otras asignaturas, el nivel con que se tratan es solamente introductorio. El diseño de un dwh se introduce en este capítulo con el propósito de que el estudiante pueda identificar la diferencia en el modelo de diseño en relación con el utilizado para una base de datos relacional, a pesar de que su implementación generalmente se hace usando un SGBD relacional.

Se describe, en este capítulo, el estándar CRISP-DM, para el desarrollo de proyectos de minería de datos (*data mining*). Se presentan también dos de las tareas de *data mining* ampliamente conocidas: clasificación y asociación. En relación con el tema de *data warehousing*, se describe el modelo multidimensional, como estrategia de diseño de bodegas de datos o *data warehouse.*

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

Capítulo 1

INTRODUCCIÓN A LOS MODELOS DE DATOS Y A LOS SISTEMAS DE GESTIÓN DE BASES DE DATOS

Este capítulo introduce al estudiante en los modelos de datos y en los Sistemas de Gestión de Bases de Datos (SGBD). Tomando como referencia el texto guía, se presenta el concepto de modelo de datos como una forma para especificar: una estructura de datos particular, un conjunto de restricciones sobre esta estructura y mecanismos para manipular los datos.

Modelos de datos

Los modelos de datos están integrados por una serie de conceptos para describir datos, sus relaciones y restricciones [AHV95] [SKS96] y son útiles para representar, de manera abstracta, el mundo real. Su propósito, de acuerdo con [AHV95] es, además de facilitar la descripción de los datos y sus relaciones, permitir la representación de los datos y hacerlos entendibles. Por esta razón, los modelos de datos facilitan el diseño de bases de datos. Para especificar la estructura y las restricciones (i.e. árboles, grafos y relaciones) se usa un lenguaje de definición de datos (Data Definition Language - DDL, por sus siglas en inglés) y para especificar la manipulación de los datos se utiliza el lenguaje de manipulación de datos (Data Manipulation Language - DML, por sus siglas en inglés). Un DML ofrece mecanismos para recuperar datos de la base de datos vigente y para actualizar datos produciendo un nuevo estado de la base de datos.

Algunas de las utilidades que tiene un modelo de datos son, de acuerdo con [Codd80], facilitar la especificación de los tipos y la forma como los datos están organizados en una base de datos y servir de base para desarrollar metodologías de diseño de bd (base de datos) y lenguajes de alto nivel para consultar y manipular los datos.

Los siguientes tres componentes integran un modelo de datos [Codd80], [AH87], [MS80]:

- Un conjunto de tipos de estructuras de datos, componente estructural
- Un conjunto de reglas u operadores para manipular los datos, componente de manipulación de datos, y
- Un conjunto de reglas de integridad para asegurar estados consistentes de la bd, componente de especificación de integridad.

Por su parte, en [MS80] se define un modelo de bases de datos "… como un formalismo para expresar la estructura lógica de una bd y ofrecer las bases para manipularla". Los autores identifican, al igual que en [Codd80] cuatro componentes lógicas que integran un modelo de base de datos:

> *... un conjunto de elementos atómicos y de relaciones entre estos elementos, denominado espacio de datos, una especificación de restricciones aplicadas a las relaciones en el espacio de datos denominada restricciones de definición de tipo, un conjunto de operaciones para crear y destruir elementos y modificar las relaciones entre estos denominada operaciones de manipulación y un lenguaje de predicados para identificar de la base de datos elementos individuales por medio de sus propiedades lógicas.*

Los modelos de datos que hasta ahora se han propuesto se pueden clasificar en tres categorías [AHV95]:

- Modelos orientados por objetos,
- Modelos orientados por registros, y
- Modelos de datos físicos.

El modelo entidad-relación [Chen76], los modelos semánticos y el modelo orientado a objetos son, entre otros, modelos de datos orientados por objetos. En los modelos orientados por objetos no solamente se considera las entidades definidas mediante atributos (estado) sino que también se les asocia conducta. Los atributos de los objetos contienen valores. La conducta se expresa mediante mensajes a los que el objeto responde y métodos que implementan los mensajes. Los objetos se agrupan en clases.

Por su parte, los modelos semánticos [HMc81] [AH87] [HK88] [SKS96] intentan ofrecer estructuras de datos más ricas para representar el mundo real con más poder de abstracción. Sus componentes básicos permiten representar objetos, sus atributos y relaciones y ofrecen constructores de objetos complejos y relaciones ISA, entre otros [HK88]. Algunas de las ventajas que ofrecen son, de acuerdo con Hull y King [HK88], el incrementar la separación entre componentes conceptuales y físicas, reducir la sobrecarga semántica de los tipos de relación y ofrecer mecanismos de abstracción. Entre los modelos semánticos están E-R [Chen76], SDM [HMc78], IFO [AH87] y GMS [HK88].

Bajo el modelo entidad-relación (E-R) el mundo se percibe compuesto por entidades, que pertenecen a un conjunto entidad y por relaciones entre ellas, cada una de las cuales juega un rol en la relación. Mediante un par atributo-valor se ofrece información sobre entidades y relaciones. El modelo entidad-relación se detalla en [Chen76].

En los modelos orientados a registros, la bd se define en términos de registros de formato fijo que puede ser de diferentes tipos. El modelo relacional es un representante de esta clase de modelos.

En los modelos de datos físicos los datos se describen con un bajo nivel de abstracción, haciendo énfasis en la manera en que los datos están almacenados.

Sistemas de gestión de bases de datos (SGBD)

Un sistema de gestión de bases de datos (SGBD) es una capa de software necesaria para crear, manipular y recuperar datos desde una base de datos. De acuerdo con McLeod y Miles [MS80], un SGBD es una herramienta de propósito general útil para estructurar, almacenar y controlar los datos ofreciendo interfaces de acceso a la base de datos. Tareas fundamentales que desempeñan estos sistemas hacen referencia a la seguridad de acceso a los datos, al mantenimiento de la integridad de los datos, a mecanismos de recuperación debidos a fallos físicos y lógicos, al control de concurrencia en el momento de acceder a los datos y a la eficiencia del sistema evaluada, generalmente, en términos del tiempo de respuesta a las consultas de los usuarios.

Mediante el DDL y el DML, respectivamente, un usuario define una base de datos (tipos, estructura y restricciones) y puede recuperar, actualizar, insertar o borrar datos. Los usuarios no necesitan conocer detalles de almacenamiento de la base de datos, sólo requieren tener una vista abstracta de los datos.

Por esta razón la arquitectura de un SGBD, generalmente, se basa en la arquitectura de tres niveles (externo, conceptual e interno) ANSI-SPARC de la Figura 1.1, tomada de [AA93]. Se trata de separar la forma en que los usuarios ven los datos, de los detalles de almacenamiento físico de los mismos. Este principio de INDEPENDENCIA DE DATOS hace posible que el administrador de la bd cambie la estructura física de la bd (nivel interno) sin que la manera en la cual los diferentes usuarios ven los datos (nivel externo) se afecte.

El nivel interno describe la forma como los datos se almacenan en la base de datos (i.e. estructuras de datos, espacios de almacenamiento, índices, formato de registros). El nivel más bajo, el físico, trata con los mecanismos de almacenamiento físico que el sistema operativo utiliza (dispositivos físicos).

El nivel conceptual, representado en la arquitectura, corresponde a la descripción de los datos y de las relaciones entre éstos. A este nivel, la base de datos se ve como la integración de todas las vistas de los usuarios de la base de datos.

En el nivel externo se representa cada una las partes de la bd que es relevante para cada uno de los diferentes usuarios.

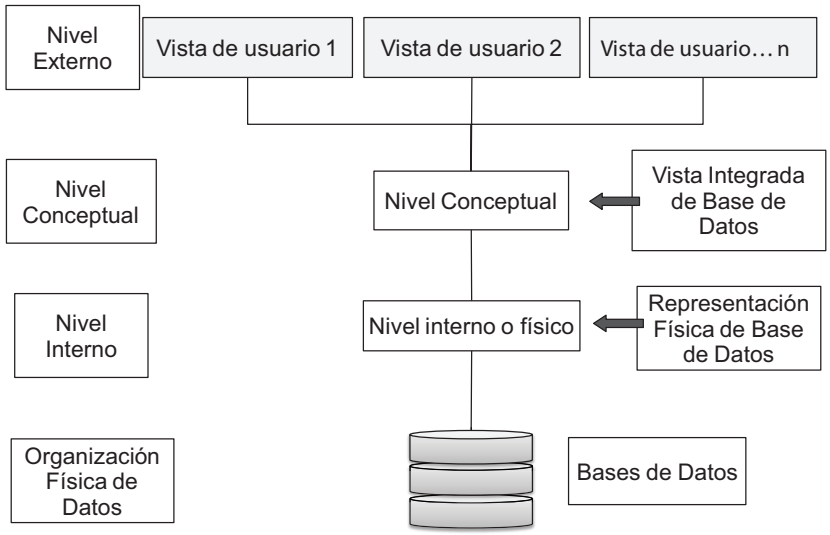

Figura 1.1 Arquitectura ANSI-SPARC

El ejemplo en la Figura 1.2, tomado de [AA93], muestra las diferencias entre los distintos niveles de la arquitectura.

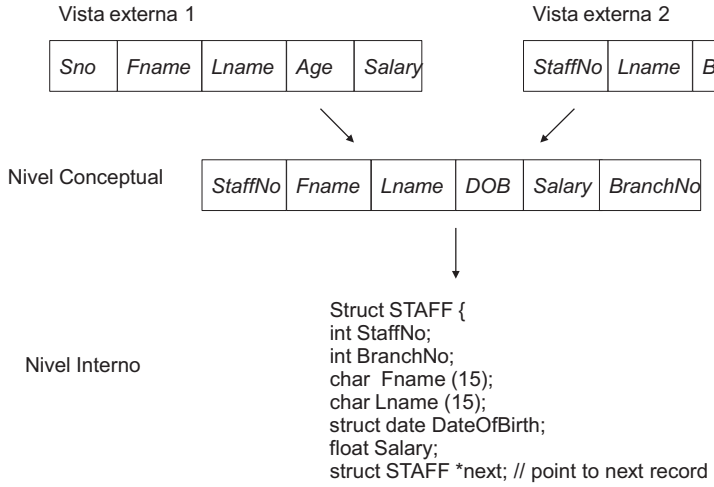

Figura 1.2 Diferencias entre niveles

Entre las funciones que ofrece al usuario un SGBD están la actualización, recuperación y almacenamiento de datos, el acceso al catálogo en el que se describen los datos almacenados, el soporte a transacciones, los servicios de control de concurrencia, recuperación y autorización, el soporte para comunicación de datos y servicios de integración y el soporte a la independencia de datos [Codd82].

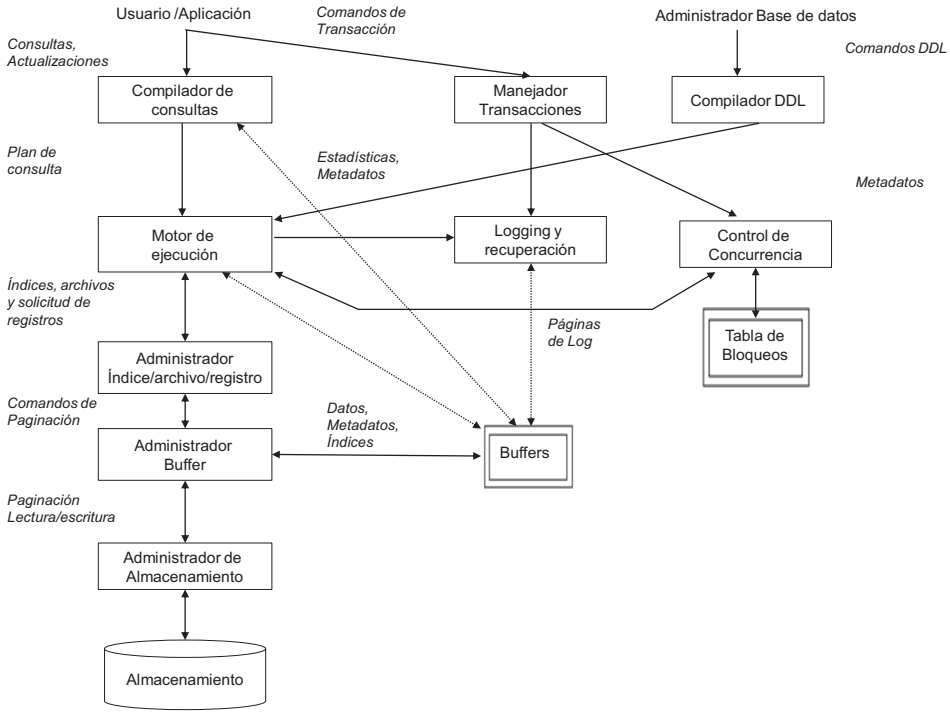

Figura 1.3 Componentes de un SGBD

Los principales componentes que integran un SGBD se presentan en la Figura 1.3, tomada de [AA93]. Cada componente tiene una función específica dentro del sistema. El procesador de consultas (*Query Processor*) transforma las consultas en instrucciones de bajo nivel para enviarlas al gestor de base de datos (*Database Manager*). El gestor de base de datos gestiona consultas de usuario con respecto a los esquemas conceptuales. Cuando la consulta se acepta, el gestor de almacenamiento (*File Manager*) debe ejecutarla. Este último gestiona espacio y asignación de almacenamiento en disco. Adicionalmente, mantiene índices. El procesador para el lenguaje de manipulación de datos (*Data Manipulation Language –DML– Processor*) transforma expresiones de manipulación de datos que aparecen en los programas de aplicación en invocaciones de funciones estándar en el lenguaje anfitrión. De esta manera, debe interactuar con el procesador de consultas.

El compilador del lenguaje de definición de datos (*Data Definition -DDL-Language*) toma una expresión de definición de datos y la convierte en un conjunto de tablas con metadatos, que se almacena en el catálogo. El gestor del catálogo (*Catalog Manager*) gestiona el acceso y mantenimiento del sistema de catálogo, al que acceden la mayoría de los componentes del SGBD. Las funciones de cada uno de estos componentes se detallan en [AA93].

Referencias

[AH87] Abiteboul S., Hull R. IFO: A Formal Semantic Database Model. *ACM Transaction on Database Systems* (12)4: 525-565, 1987.

[Codd80] E. F. Codd. Data Models in Database Management. *ACM SIGMOD Record* (11)2:112-114, 1981.

[Codd82] The 1981 Turing Award Lecture: Relational database: A practical foundation for productivity. *Communication of the ACM* 25(2): 109-117, 1982.

[DBFTG86] Thomas Burns, Elizabeth N. Fong, David Jefferson, Richard Knox, Leo Mark, Christopher Reedy, Louis Reich, Nick Roussopoulos, Walter Truszkowski. Reference Model for DBMS Standarization, Database Architecture Framework Task Group (DAFTG*). SIGMOD Record* (15)1: 19-58, 1986.

[HMc78] Hammer M., McLeod D. The semantic data model: a modeling mechanism for database applications. Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data *Austin, Texas* pp: 26 - 36, 1978.

[HMc81] Hammer M., McLeod D. Database Description with SDM: A Semantic Database Model. *ACM Transaction on Database Systems* (6)3: 351-386, 1981.

[HK88] Hull R., King R. Semantic Database Modeling: Survey, Applications and Research Issues. *ACM Computing Surveys* (19)3: 202-260, 1987.

[MS80] McLeod Dennis, Smith Miles John. Abstractions in Databases. *SIGMOD Record* 11(2): 19-25, 1981.

[SKS96] Silberschatz A., Korth H., Sudarshan S. Data Models. *ACM Computing Surveys* (28)1: 105-108, 1996.

Bibliografía básica anotada

[Codd80] E. F. Codd. Data Models in Database Management. *ACM SIGMOD Record* (11)2:112-114, 1981.

El autor define un modelo de datos en términos de tres componentes: estructuras, operadores y reglas de integridad. Propone, también, varios usos que se le puede dar a un modelo de datos y discute algunos de los conceptos que han generado, en alguna medida, malentendidos en la comunidad de bases de datos.

[Codd82] The 1981 Turing Award Lecture: Relational database: A practical foundation for productivity. *Communication of the ACM* 25(2): 109-117, 1981.

A partir de tres objetivos fundamentales: independencia de datos, comunicabilidad y procesamiento de conjuntos se define un sistema de gestión de bases de datos relacional. Se ofrecen argumentos para sustentar la afirmación de que los SGBD mejoran la productividad tanto de los usuarios como de los programadores. Se ilustran distintos tipos de sistemas relacionales.

[HMc81] Hammer M., McLeod D. Database Description with SDM: A Semantic Database Model. *ACM Transaction on Database Systems* (6)3: 351-386, 1981.

Los autores se proponen diseñar un modelo de base de datos de alto nivel que le permita al diseñador incorporar de manera natural y directa más semántica al esquema de base de datos. Se describe SDM, un formalismo para estructurar y describir una base de datos de manera que el esquema pueda "capturar más significado". Se pretende, principalmente, que SDM sirva como un mecanismo de especificación formal para describir el significado de la base de datos.

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

Capítulo 2

MODELO DE DATOS RELACIONAL[1]

Este capítulo introduce el modelo relacional. Inicialmente, se define de manera formal el modelo relacional. Luego, tomando como referencia [AHV95], [Codd70], se describen dos enfoques para definir el modelo relacional. El primer enfoque depende de si el nombre o el orden de los atributos es importante. El segundo enfoque depende de cómo se ven las relaciones y las instancias de la base de datos. Posteriormente, se muestran los lenguajes de consulta relacionales, haciendo énfasis en un tipo frecuente de consulta: las conjuntivas. En esta misma sección se presentan tres paradigmas de consulta.

Modelo relacional: Definición

Intuitivamente, las relaciones se asocian con tablas nombradas cuyas columnas representan atributos que también pueden tener asociado un nombre. Las filas de las tablas son tuplas. Los valores que toman las tuplas se extraen de conjuntos de constantes llamados dominios. Todas las tablas constituyen la estructura de la base de datos que se representa en un esquema de base de datos (nivel intensional) y su contenido en una instancia de base de datos (nivel extensional).

La propuesta inicial del modelo de datos relacional [Codd70] caracteriza las relaciones como la estructura fundamental para describir y organizar los datos y el álgebra relacional para manipularla. El modelo relacional se define en [Codd79], integrado a partir de tres elementos:

- Un conjunto de relaciones que varía en el tiempo,
- Reglas de inserción-actualización-eliminación, y
- Un álgebra relacional.

1 Capítulo 3 del texto guía.

Preliminares y notación

Sean:

- ***att***, un conjunto contable infinito de atributos con un orden total $\leq_{att}$ sobre ***att***,
- ***dom***, un conjunto contable infinito, disjunto con respecto a ***att***, llamado dominio, y
- ***relname***, un conjunto contable infinito de nombres de relaciones disjunto de los otros conjuntos,
- ***var***, un conjunto infinito de variables que toman valores sobre elementos de ***dom***.

Como notación estándar se utilizan:

- las primeras letras del alfabeto en mayúsculas o con subíndices (A, B, A_1, A' ,…) para denotar los atributos individuales,
- las letras finales del alfabeto (X, X_1,Y, Y',…) para denotar los conjuntos de atributos,
- las letras mayúsculas R, S para denotar nombres de relaciones,
- el mismo nombre del esquema en letra minúscula (i.e. r_k es una instancia de relación definida sobre el esquema R_k) para denotar las instancias de relaciones, y
- letras en negrita ***R***, $\boldsymbol{R_1}$ para denotar los esquemas de base de datos.

Se define una relación R a partir de n conjuntos $S_1, S_2, ..., S_n$ consistiendo de tuplas de aridad n en donde cada elemento i toma valores en S_i. En la relación R cada fila representa una n-tupla distinta. El orden de las filas no es importante pero sí el de las columnas, cuando el orden de los atributos es relevante. Sin embargo, a las columnas se les puede asignar una etiqueta que corresponde a su respectivo dominio. Las diferentes relaciones o tablas, variantes en el tiempo, integran la base de datos.

La estructura de una tabla (relación) se define asignándole un nombre y un conjunto de atributos que corresponde al tipo ($sort$) de la relación, donde $sort$ es la función,

$$\text{sort} : \textbf{relname} \rightarrow \mathrm{P}^{fin}(\textbf{att})$$

y P^{fin} es el conjunto potencia finito de ***att***.

El tipo de una relación se denota por $sort(R)$. La aridad de la relación R es aridad (R)=$|sort(R)|$.

Un esquema de relación es un nombre de relación R. Un esquema de base de datos es un conjunto finito no vacío ***R*** de nombres de relación, es decir ***R*** = $\{R_1[U_1], R_2[U_2], \ldots, R_n[U_n]\}$, donde cada $R_i[U_i]$ representa $sort(R) = U$.

Sea u una tupla definida sobre U. El valor de u para un atributo A en U es $u(A)$. Cuando $V \subseteq U, u[V]$ representa a la tupla v de V tal que $v(A) = u(A)$ para cada $A \in V$.

Enfoques para el modelo relacional

Enfoques nombrado y no-nombrado

De acuerdo con [AHV95], [Codd70], [Codd79], y dependiendo de si se considera importante el nombre de los atributos asignados a las columnas de una relación o su orden, el modelo relacional se puede formular desde dos enfoques: uno nombrado y otro no-nombrado.

Bajo el primer enfoque, el nombrado, los atributos forman parte, explícitamente, del esquema de la base de datos. Una tupla *u* se define a partir de un esquema de relación $R[U]$ y se escribe, usualmente, utilizando como sintaxis $\langle A_1 : a_1, A_2 : a_2, ..., A_k : a_k \rangle$ donde, generalmente, el orden de los atributos es $\leq_{att}$.

En el segundo, el no-nombrado, sólo se tiene en cuenta la aridad del esquema de una relación. Una tupla es una n-tupla ordenada ($n \geq 0$) de constantes. La i-ésima coordenada de una tupla u es $u(i)$.

Enfoques convencional y basado en programación lógica

Por otra parte, y dependiendo de la forma como se ven las relaciones y las instancias de una base de datos se presentan dos perspectivas [AHV95]: la convencional y la basada en programación lógica.

Bajo la perspectiva convencional, una instancia de relación definida sobre un esquema $R[U]$ es un conjunto finito I de tuplas de tipo U. Una instancia de base de datos de un esquema de base de datos $\mathbf{R}$ es una transformación $\mathbf{I}$ con dominio R tal que $\mathbf{I}(R)$ es una relación sobre R para cada $R \in \mathbf{R}$.

Desde la perspectiva de programación lógica las tuplas son ordenadas y se ven como hechos de una relación. Un hecho definido sobre R y una relación de aridad n, es una expresión de la forma $R(a_1, a_2, ..., a_n)$ donde $a_i \in \mathbf{dom}$, $i \in [1, n]$. Una relación definida sobre R es un conjunto finito de hechos definidos sobre R. Cuando $\mathbf{R}$ es un esquema de base de datos, una instancia de base de datos es un conjunto finito $\mathbf{I}$ que es la unión de instancias de relación definidas sobre R para $R \in \mathbf{R}$.

Lenguajes de consulta relacionales

Una de las funciones primarias de los SGBD es la recuperación de datos a partir de la bd. Una consulta relacional simple o compleja, se trata como una transformación sobre el contenido de la bd considerada como una colección de relaciones. El valor que la consulta devuelve es también una relación.

Un lenguaje de consulta es "una herramienta lingüística bien-definida cuyas expresiones corresponden a una petición a la base de datos" [Cha-Ha80]. Los lenguajes de consulta relacionales más ampliamente estudiados son el álgebra y el cálculo relacional. Adicionalmente, el lenguaje *datalog* surge como un representante de los lenguajes de consulta basados en programación lógica.

Definición de consulta

Sea U el conjunto de todas las relaciones definidas sobre el conjunto de todos los atributos $\boldsymbol{att}$. Sea $I(\boldsymbol{R})$ el conjunto de todas las instancias del esquema de base de datos $\boldsymbol{R}$. Una consulta Q es una función parcial recursiva $Q: I(\boldsymbol{R}) \rightarrow U$. Para un $\boldsymbol{r} \in I(\boldsymbol{R})$ dado, $Q(\boldsymbol{r})$ es el resultado de aplicar la consulta Q sobre $\boldsymbol{r}$.

Las consultas se expresan por medio de expresiones E escritas teniendo en cuenta la sintaxis y la semántica del lenguaje. $E(\boldsymbol{r})$ es el valor que toma la consulta cuando se aplica sobre una instancia de la base de datos $\boldsymbol{r}$.

Dos expresiones E_1 y E_2 son equivalentes con respecto a un esquema de base de datos $\boldsymbol{R}$, $E_1 \equiv_{\boldsymbol{R}} E_2$, si representan la misma consulta, es decir, $E_1(\boldsymbol{r}) = E_2(\boldsymbol{r})$, para cada instancia $\boldsymbol{r} \in I(\boldsymbol{R})$.

Un aspecto que tiene que ver con los lenguajes de consulta es su poder expresivo. La pregunta sobre qué poder debe tener un lenguaje de consulta debería depender más de "su capacidad de recuperación de datos a partir de una base de datos que de la aritmética computacional sobre los datos" [AU79]. Con base en esta apreciación, Aho et al. [AU79] postulan los siguientes dos principios que un lenguaje de consulta debería satisfacer. El primero, que el resultado de una consulta sea independiente de la manera en la cual los datos estén realmente almacenados en la bd. El segundo, que el lenguaje de consulta trate los valores de los datos como objetos sin interpretación. El álgebra y el cálculo relacional satisfacen estos principios [Codd70] y por esta razón se convierten en modelos de lenguajes de consulta. Justamente, Codd [Codd70] introduce estos lenguajes como estándares para medir el poder expresivo de los lenguajes de consulta relacionales.

Sin embargo, el poder expresivo de los lenguajes de consulta relacionales, álgebra y cálculo, basados en lógica de primer orden sin símbolos de función, es limitado. Un ejemplo de esta limitación es el cálculo del cierre transitivo de una relación binaria [AU79]. Ejemplos de estas expresiones, que aparecen en [AU79], son los siguientes: determinar el total de vuelos posibles entre dos ciudades en un periodo de tiempo dado, en el contexto de un sistema de reservas de una aerolínea, o identificar el administrador de nivel más bajo común a un grupo de empleados en sistema de gestión de negocios. Ninguna de estas consultas se puede expresar en álgebra relacional o en cálculo relacional. En [AV92] se extienden el álgebra y el cálculo relacionales con recursión.

Formalmente [AA93], dados dos lenguajes de consulta L_1 y L_2 , L_1 es tan expresivo como L_2, si para cada expresión E_2 de L_2 existe una expresión E_1 de L_2 tal que $E_1 \equiv E_2$.

Paradigmas de consulta

Extraer datos de una base de datos es uno de los tópicos en bases de datos estudiados más extensivamente. Diferentes paradigmas se han propuesto con este propósito: algebraico y no algebraico. El paradigma no algebraico tiene dos vertientes: basado en lógica y basado en programación lógica. En esta sección se presentan estos paradigmas de manera detallada.

Consultas basadas en álgebra relacional

El álgebra relacional es un lenguaje procedimental (procedural) puesto que cualquier expresión relacional describe una serie de pasos para calcular un resultado a partir de una instancia de la base de datos.

El álgebra relacional, introducida por Codd [Codd70], [Codd79], incluye los operadores clásicos de conjuntos: Unión, Intersección y Diferencia y otros aplicables a relaciones como Permutación, Proyección, Renombramiento, Selección y Join. Otros operadores como el Producto cartesiano, Theta-Select, Theta-Join, Natural Join y la División se definen en [Codd79]. Adicionalmente, el álgebra se extiende para tratar con valores nulos.

La aplicación de operadores de Unión, Intersección y Diferencia se restringe a relaciones unión-compatibles —relaciones cuyos atributos se corresponden uno a uno definidos sobre el mismo dominio—.

Algunos operadores algebraicos

Las siguientes definiciones de operadores algebraicos son tomadas de [Codd70], [Codd79] y [AA93].

Proyección

Dada una relación $r(X)$ y un subconjunto Y de X, la proyección de r sobre Y, denotada por $\pi_Y(r)$, es una relación definida sobre los atributos en Y que restringe los atributos de r a los atributos en Y .

$$\pi_Y(r) = \{t(Y) \mid t \in r\} \quad (2.1)$$

Selección

Este operador se define con respecto a una fórmula proposicional. Sea r una relación definida sobre un conjunto de atributos X . Una fórmula proposicional F sobre X se define recursivamente a partir de átomos y conectivos de la siguiente forma:

Los átomos son de la forma $A_1 \theta A_2$ ó $A_1 \theta a$, donde a es una constante y θ es uno de los operadores de comparación <, ≤, = , ≥, >, ≠. Cada átomo definido sobre X es una fórmula proposicional definida sobre X. Si F_1, F_2

son fórmulas proposicionales definidas sobre X, también lo son $\neg(F_1)$, $F_1 \wedge F_2$, $F_1 \vee F_2$.

Una fórmula proposicional tiene asociado un valor booleano (*true*, *false*)

Dada una relación $r(X)$ y una fórmula proposicional F definida sobre X, la selección de r con respecto a F, denotada por $\sigma_F(r)$ es la relación:

$$\sigma_F(r) = \{t \in r \mid F(t) = true\} \quad (2.2)$$

Renombramiento

Operador unario que cambia el nombre de los atributos en una relación. Dada una relación r definida sobre un conjunto de atributos X y una función inyectiva f definida en X (que asigna un nuevo nombre a cada atributo), el renombramiento de r con respecto a f se define como

$$\rho_f(r) = \{\, t \mid \text{existe } t' \in r \text{ tal que } t'[A] = t[f(A)] \text{ para todo } A \in X \,\} \quad (2.3)$$

Restricción o Tetha-Select

Sea θ un operador de comparación <, ≤, = , ≥, >, ≠ aplicable al atributo A y a la tupla c. $R[A\theta c]$ es el conjunto de tuplas de R, cada una de cuyas A-componentes satisface la condición θ con respecto a la tupla c. Otros atributos B de R pueden aparecer en lugar de la tupla c, teniendo en cuenta que A y B estén definidos sobre un dominio común. Cuando θ es la igualdad, el tetha-Select se llama SELECT.

Producto cartesiano (x)

Combina relaciones. Sean R y S relaciones de aridad n y m, respectivamente:

$$R \times S = \{\langle r(1),...,r(n),s(1),...,s(m)\rangle \mid r \in R, s \in S\} \quad (2.4)$$

Theta-Join

Sean $R(A,B_1)$ y $S(B_2,C)$ relaciones con B_1, B_2 definidos sobre un dominio común y θ uno de los operadores de comparación <, ≤, =, ≥, >, ≠ aplicable al dominio de los atributos B_1, B_2. El Theta-Join de R y S, denotado por $R[B_1\theta B_2]S$, es la concatenación de filas de R con filas de S en donde la B_1 —componente de la fila en R— satisface la relación θ con respecto a la B_2 —componente de la fila S—. Cuando θ es la igualdad el operador se llama EQUI-JOIN. Si las relaciones R y S tienen atributos comunes, los nombres de los atributos en la relación resultante se deben especificar.

Natural Join

Sean $r_1(YX)$ y $r_2(XZ)$ dos relaciones, tal que $YX \cap XZ = X$. El *join* natural

de r_1 y r_2, denotado por $r_1 \bowtie r_2$ es una relación definida sobre YXZ consistente en todas las tuplas resultantes de concatenar tuplas en r_1 con tuplas en r_2 que tengan el mismo valor en X.

$$r_1 \bowtie r_2 = \{ t \text{ sobre } YXZ \mid \text{existen } t_1 \in r_1, t_2 \in r_2 \text{ tal que } t[XY] = t_1[XY] \text{ y } t[XZ] = t_2[XZ] \} \quad (2.5)$$

División

Dadas las relaciones $R(A, B_1)$ y $S(B_2, C)$ con B_1, B_2 definidas sobre el mismo dominio, $R[B_1 \div B_2]$ es el subconjunto maximal de $R[A]$ tal que su producto cartesiano con $S[B_2]$ está incluido en R. Contraparte algebraica del cuantificador universal.

Consultas algebraicas conjuntivas

Una buena parte de las consultas son conjuntivas, porque se basan en conjunción y están cuantificadas existencialmente. Las consultas conjuntivas se pueden extender para tratar con la negación y la unión.

Cuando se consideran únicamente los operadores algebraicos primitivos, sin incluir la negación (conjuntivas), y bajo los enfoques nombrado y no-nombrado se conforman el álgebra conjuntiva no-nombrada o álgebra SPC (Select-Project-Cartesian Product) y el álgebra conjuntiva nombrada o SPJR (Select-Project-Natural Join).

Álgebra SPC

El álgebra SPC está constituida por tres operadores algebraicos primitivos: selección, proyección y producto cartesiano. Los operadores SPC se definen teniendo en cuenta que, bajo el enfoque no-nombrado, las tuplas son tuplas ordenadas.

Desde esta perspectiva y de acuerdo con [AHV95], [Codd70], considerando un esquema de base de datos $\boldsymbol{R}$, el operador de selección, en la Ecuación 2.2, tiene dos formas primitivas, $\sigma_{j=a}$ y $\sigma_{j=k}$, donde j, k son enteros positivos y $a \in \boldsymbol{dom}$. El operador toma como entrada una instancia de relación r y retorna otra instancia de relación de la misma aridad. La aridad de la instancia de relación de entrada es, respectivamente, $\geq j$ y $\geq \max\{j, k\}$. La instancia de relación de salida es, respectivamente $\sigma_{j=a}(r) = \{t \in r \mid t(j) = a\}$ o $\sigma_k(r) = \{t \in r \mid t(j) = k\}$, donde j, k son enteros positivos.

La proyección, cuya forma general es $\pi_{j_1,\ldots,j_n}$ donde $j_1,\ldots,j_n$ es una secuencia de enteros positivos con posibles repetidos, toma como entrada una instancia de relación con aridad $\geq \max\{j_1,\ldots,j_n\}$. La instancia de relación resultado es $\pi_{j_1,\ldots,j_n}(r) = \{\langle t(j_1),\ldots,t(j_n)\rangle \mid t \in r\}$.

El operador producto cartesiano toma como entrada un par de relaciones

r, s de aridad n y m. El resultado de su aplicación es la instancia de relación de aridad $n+m$ definida por $r \times s = \{\langle t(1),...,t(n),t'(1),...,t'(m)\rangle \mid t \in r, t' \in s\}$

Una definición inductiva formal del álgebra SPC se presenta en [AHV95].

Álgebra SPJR

El álgebra SPJR o álgebra conjuntiva nombrada tiene cuatro operadores algebraicos primitivos: selección, proyección, *join* natural y renombramiento.

La selección se aplica sobre una instancia r con $A \in sort(r)$ y tiene la forma $\sigma_{A=c}$ y $\sigma_{A=B}$ donde $A, B \in$ ***att*** y $a \in$ ***dom***. Su definición es análoga a la del álgebra SPC.

La proyección tiene la forma $\pi_{A_1,...,A_n}, n \geq 0$ y produce una instancia de relación $sort\{A_1,...,A_n\}$. Este operador no permite repetidos.

El *join* natural, $\bowtie$, toma como entrada dos instancias de relación r, s $sort(r) = V, sort(s) = W$ y entrega como resultado $r \bowtie s = \{$ t de tipo $V \cup W \mid$ para algún $v \in r$ y $w \in s$, $t[V] = v$ y $t[W] = w\}$. Si $sort(r) = sort(s)$, $r \bowtie s = r \cap s$.

Finalmente, el renombramiento se define sobre un conjunto de atributos U como una transformación $U \rightarrow$ ***att***. El renombramiento de un atributo A es $f(A)$ siendo $A \neq f(A)$. La entrada de este operador definida sobre U, es una expresión δ_f, donde f es el renombramiento de U, que entrega como resultado:

$$\delta_f(r) = \{v \text{ sobre } f[U] \mid \text{para algún } u \in r, v(f(A)) = u(A) \text{ para cada } A \in U\}$$

Cuando al álgebra conjuntiva se le añade el operador unión, se aumenta la capacidad disyuntiva a las consultas conjuntivas para producir álgebras SPCU y SPJRU. La única restricción es que la unión sólo es aplicable a instancias del mismo tipo. Con la negación el tratamiento es similar y se puede añadir fácilmente al igual que la unión y la intersección para obtener un álgebra relacional nombrada y no-nombrada, con más poder expresivo. El capítulo 5 del texto guía trata, detalladamente, este tema.

Consultas no algebraicas

En esta sección se introduce, inicialmente, el cálculo relacional propuesto por Codd [Codd72]. Posteriormente, basado en [AHV95], y desde un punto de vista no algebraico, se presentan versiones equivalentes de las consultas conjuntivas expresadas en forma de reglas, utilizando un *tableau* y basado en el cálculo de predicados de primer orden.

Cálculo relacional

Desde un punto de vista lógico, Codd introduce el cálculo relacional en [Codd72] y prueba su equivalencia con respecto al álgebra relacional. Los

lenguajes basados en cálculo son declarativos puesto que las consultas se expresan especificando propiedades del resultado.

Un alfabeto, términos y fórmulas del cálculo relacional se presentan en esta sección. El concepto de variable de tupla se requiere para representar variables que "varían" o que toman valores en una relación. Los valores de la variable son tuplas de una relación dada. Las siguientes definiciones son tomadas de [AA93]:

Una expresión del cálculo relacional tiene la forma:

$$\{A_1 : x_1,..., A_k : x_k \mid f\} \quad (2.6)$$

donde f es una fórmula, $x_1,....,x_k$ son variables apareciendo en f y $A_1,....,A_k$ son atributos. $A_1 : x_1,..., A_k : x_k$ es la lista objetivo de la fórmula y define la estructura del resultado de evaluar la expresión. El resultado es una relación definida sobre $A_1,....,A_k$ conteniendo tupla cuyos valores $c_1,....,c_k$ satisfacen f cuando sustituyen a $x_1,....,x_k$.

Las fórmulas pueden incluir constantes (elementos del dominio D), variables (elementos de un conjunto infinito contable V disjunto de D), nombres de relaciones y atributos del esquema de base de datos, operadores de comparación, conectivos lógicos ($\wedge,\vee,\neg$), cuantificadores $\exists,\forall$ y paréntesis.

Un átomo puede tener la siguiente forma:

- $R(A_1 : x_1,..., A_p : x_p)$ donde $R(A_1,..., A_p)$ es un esquema de relación y las x_i son variables distintas. Si se trabaja bajo un enfoque no-nombrado los átomos se pueden omitir $R(x_1,..., x_p)$.
- $x\theta a$, donde x es una variable, θ es un operador de comparación y a es una constante o una variable.

Las fórmulas se construyen recursivamente mediante átomos combinados con conectivos y cuantificadores.

- Cada átomo es una fórmula. Todas las apariciones de variables en átomos son libres.
- Si f_1 y f_2 son fórmulas, entonces $(f_1) \wedge (f_2), (f_1) \vee (f_2)$ y $\neg(f_1)$ también lo son.
- Si f es una fórmula y x una variable, entonces $\exists x(f)$ y $\forall x(f)$ son fórmulas. La aparición de x en f está ligada. Cada aparición de cualquier otra variable está libre en f.

Con respecto a una instancia de base de datos $\boldsymbol{r} = \{ r_1,...,r_m \}$ definida sobre el esquema $\boldsymbol{R} = \{ R_1(X_1),...,R_m(X_m) \}$, la semántica de una expresión se basa en la definición recursiva del valor de una fórmula, que depende de los valores que tienen las variables libres cuando se sustituyen. Una sustitución asocia cada variable con una constante, es una función $s : V \rightarrow D$.

El resultado (valor) de aplicar una sustitución sobre una fórmula puede ser de una de las siguientes formas:

- Átomo
- *true* para una sustitución s, si la relación r definida sobre $R(A_1 : x_1,..., A_p : x_p)$ contiene una tupla t tal que $t[A_i] = s(x_i)$ para $1 \le i \le p$ y *false* en otro caso.
- Para $x_1 \theta a$ hay dos casos. (a) si a es una variable x_2, entonces $x_1 \theta x_2$ es *true* para s, si $s(x_1)$ satisface la relación θ con $s(x_2)$. (b) si a es una constante c, entonces $x_1 \theta c$ es *true* para s si $s(x_1)$ satisface la relación θ con c.
- Los valores de $(f_1) \wedge (f_2), (f_1) \vee (f_2)$ y $\neg(f_1)$ para una sustitución s se definen con base en la semántica de los conectivos a partir de los valores de f_1 y f_2 en s.
- El valor de $\exists x(f)$ para una sustitución s es *true*, si existe una sustitución s' para la que f es *true* que difiere de s no más que en x; en otro caso es *false*. El valor de $\forall x(f)$ para una sustitución s es *true* si f es *true* en cada sustitución s' que difiere de s a lo sumo en x: en otro caso es *false*.

El valor de una expresión del cálculo orientado a dominio de la forma $\{A_1 : x_1,..., A_k : x_k \mid f\}$ es una relación definida sobre los atributos $A_1,...., A_k$. Las tuplas de la relación las constituyen las sustituciones que hacen *true* a f definida por:

$\{t \mid$ existe s tal que $f(s) = true$ y $t[A_i] = s(x_1)$, para $1 \le i \le k\}$.

Una observación importante

El cálculo relacional orientado a dominio descrito en este capítulo no es dominio independiente, propiedad no deseable. Por ejemplo, el valor de algunas expresiones podría depender no sólo de los valores almacenados en una instancia de la base de datos sino del dominio. Cuando el dominio cambia el resultado puede cambiar. Para dominios infinitos el conjunto de tuplas resultante es infinito y, por lo tanto, el resultado no sería una relación. Por esta razón, se introduce el concepto de dominio activo D_r de $\boldsymbol{r}$, como el conjunto de valores del dominio que aparecen en alguna tupla de alguna relación de $\boldsymbol{r}$. El dominio activo de una expresión E y $\boldsymbol{r}$ es la unión $D_{E,r}$ del dominio activo D_r de $\boldsymbol{r}$ y el conjunto de constantes que aparecen en la fórmula f de la expresión. El cálculo relacional de dominios con rango restringido elimina este problema.

Consultas conjuntivas basadas en reglas, en *tableau* y en cálculo conjuntivo

En [AHV95] se presentan tres versiones para las consultas conjuntivas, que son equivalentes: consultas conjuntivas basadas en reglas, consultas *tableau* y consultas basadas en cálculo conjuntivo. En esta sección se intro-

ducen, brevemente, estas tres versiones para las consultas conjuntivas, que se detalla en la sección 4.2 del texto guía[2].

Consultas conjuntivas basadas en reglas

Una consulta conjuntiva basada en regla llamada simplemente regla, definida sobre un esquema de base de datos $\boldsymbol{R}$, es una expresión de la forma

$$ans(u) \leftarrow R_1(u_1),...,R_n(u_n) \quad (2.7)$$

donde $n \geq 0, R_1,...,R_n$ son nombres de relaciones en $\boldsymbol{R}$ y $u, u_1,...,u_n$ son tuplas libres que se pueden usar como variables o como constantes.

$R_1(u_1),...,R_n(u_n)$, es el cuerpo de la regla y $ans(u)$ la cabeza. Las reglas son de rango restringido puesto que todas las variables que aparecen en la cabeza deben también aparecer en el cuerpo. Una valuación es una asignación de valores para las variables de la regla.

Una consulta conjuntiva q basada en reglas, aplicada sobre una instancia $\boldsymbol{I}$ de $\boldsymbol{R}$ se define como sigue:

$q(\boldsymbol{I}) = \{v(u) \mid v$ es una valuación sobre $\text{var}(q)$ y $v(u_i) \in \boldsymbol{I}(R_i)$, para cada $i \in [1,n]\}$, donde $\text{var}(q)$ es el conjunto de variables apareciendo en q.

$adom(\boldsymbol{I})$ identifica al dominio activo de la instancia de base de datos $\boldsymbol{I}$ consistente en todas las constantes que aparecen en $\boldsymbol{I}$. El dominio activo de una instancia de relación r se define análogamente. $adom(q)$ representa al conjunto de constantes apareciendo en q.

Consultas *tableau*

Las consultas conjuntivas *tableau* son más visuales. Una consulta *tableau* es un par $(\boldsymbol{T}, u)$, donde $\boldsymbol{T}$ es un *tableau* y cada variable que aparece en u también aparece en $\boldsymbol{T}$. La tupla libre u es el *summary* de la consulta *tableau*. Esta tupla *summary* representa al conjunto de tuplas de la respuesta a la consulta. Una respuesta se forma con todas las tuplas u de la base de datos que satisfacen el patrón descrito por $\boldsymbol{T}$.

En la sección 4.2 del texto guía se detallan las consultas *tableau* y se ofrecen ejemplos ilustrativos.

Cálculo conjuntivo

Esta última versión basada en cálculo de predicados es, de acuerdo con [AHV95], una versión de las consultas conjuntivas basadas en reglas. Las expresiones utilizan conjunciones y cuantificadores existenciales.

Sea $\boldsymbol{R}$ un esquema de base de datos. Una fórmula bien formada definida sobre $\boldsymbol{R}$ para el cálculo conjuntivo es una expresión que puede ser:

2 Abiteboul et al. Foundations of Databases. Addison-Wesley Publishing, 1995.

a. un átomo definido sobre $\boldsymbol{R}$
b. $(\varphi \wedge \psi)$, donde φ,ψ son fórmulas definidas sobre $\boldsymbol{R}$
c. $\exists x\varphi$, donde x es una variable y φ es una fórmula definida sobre $\boldsymbol{R}$.

La aparición de una variable x en una fórmula φ está libre si:
I. φ es un átomo
II. $\varphi = (\psi \wedge \xi)$ y x está libre en ψ o en ξ
III. $\varphi = \exists y\psi$, donde x e y son variables distintas y x es una variable libre en ψ.

La aparición de una variable x en φ está ligada si no está libre. El conjunto de todas las variables que tienen al menos una aparición libre en φ se denota por $libre(\varphi)$.

Una consulta de cálculo conjuntivo definida sobre un esquema de base de datos $\boldsymbol{R}$ es una expresión de la forma $\{e_1,....,e_m \mid \varphi\}$ donde φ es una fórmula del cálculo conjuntivo, $\langle e_1,....,e_m \rangle$ es una tupla libre y el conjunto de variables apareciendo en $\langle e_1,....,e_m \rangle$ es exactamente $libre(\varphi)$.

Cuando el enfoque nombrado se usa los atributos se pueden escribir explícitamente así: $\{\langle e_1,...,e_m \rangle : A_1,..., A_m \mid \varphi\}$.

La semántica de las consultas de cálculo conjuntivo se detalla en la sección 4.2 del texto guía [AHV95].

Sea $q = \{e_1,....,e_m \mid \varphi\}$ una consulta del cálculo conjuntivo definida sobre $\boldsymbol{R}$. Para una instancia $\boldsymbol{I}$ definida sobre $\boldsymbol{R}$, la imagen de $\boldsymbol{I}$ bajo q es:

$q(\boldsymbol{I}) = \{v(\langle e_1,....,e_n \rangle) \mid \boldsymbol{I} \models \varphi[v]$ y v es una valuación sobre $libre(\varphi)\}$.

El dominio activo de una fórmula φ, $adom(\varphi)$, es el conjunto de constantes que aparecen en φ. Para una consulta q, $adom(\varphi, \boldsymbol{I})$ es la abreviación de $adom(\varphi) \cup adom(\boldsymbol{I})$. Esto significa que la evaluación de una consulta de cálculo conjuntivo se hace sobre este dominio que es finito.

Al igual que con las consultas basadas en álgebra, es posible adicionar a las consultas basadas en reglas y a las consultas *tableau* la unión. Sin embargo, cuando se añade la disyunción a las consultas conjuntivas basadas en cálculo, surgen los mismos problemas, descritos para el cálculo, en general, que se analizan en el capítulo 5 del texto guía.

Por otra parte, para incluir la negación en las consultas basadas en reglas, se requiere permitir literales negativos en el cuerpo de las reglas. En el caso de las cálculo, al introducir la negación, surgen algunas dificultades puesto que se puede generar como resultado relaciones infinitas.

Para dar solución a este tipo de dificultad, se introduce el concepto de consulta segura, que se detalla en el mismo capítulo 5 del texto guía.

Referencias

[AHV95] Abiteboul S., Hull R., Vianu V. Foundations of Databases. Addison-Wesley Publishing Company, 1995.

[AU79] Aho A. V., Ullman J.D. Universality of Data Retrieval Languages. Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. pp: 110-119, 1979.

[AV92] Abiteboul S., Vianu V. Expressive Power of Query Languages. Reporte Técnico 1587, INRIA. Disponible en http://hal.inria.fr/docs/00/07/49/73/PDF/RR-1587.pdf.

[ChaHa80] Chandra A. K., Harel D. Computable Queries for Relational Data Bases. Journal of Computer and System Science (21): 156-178, 1980.

[ChaMer77] A. K. Chandra and P. M. Merlin. Optimal implementation on conjunctive queries in relational data bases. In Proc. ACM SIGACT Symposium. on the Theory of Computing pp: 77–90, 1977.

[Codd70] Codd E. F. A Relational Model of data for Large shared data banks. Communication of the ACM, 13(6): 377-387, 1970.

[Codd72] Codd E. F. Relational Completeness of Data Base Sublanguages. In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California. Disponible en http://www.cs.berkeley.edu/~christos/classics/Codd72a.pdf.

[Codd79] Codd E. F. Extending the database relational model to capture more meaning. ACM Trans. on Database Systems 4(4):397-434, 1979.

[Date81] Date C. J. A Formal Definition of the Relational Model. ACM SIGMOD Record(13)1: 18 - 29 , 1982.

[SS75] Schmid H.A., Swenson J.R. On the Semantics of the Relational Data Model. Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data, San Jose, California, pp:211–223, 1975 .

Bibliografía básica anotada

[Codd70] Codd E. F. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377-387, 1970.

Este artículo introduce el modelo de datos relacional en el cual la relación es la

estructura de datos y el álgebra relacional el lenguaje de consulta. Se introducen también conceptos de llave primaria, llave foránea y forma normal. El álgebra como lenguaje de consulta se describe a partir de cuatro operaciones (permutación, proyección, *join*, composición y restricción) y su aplicación, de acuerdo con el autor, garantiza independencia entre los datos almacenados y los programas que acceden a éstos. Se introducen también conceptos de derivabilidad, redundancia y consistencia de relaciones. Por otra parte, se propone pero no se describe, el cálculo de predicados de primer orden como un lenguaje para recuperar datos a partir de relaciones normalizadas.

[Codd72] Codd E.F. Relational Completeness of Data Base Sublanguages. In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California. Disponible en http://www.cs.berkeley.edu/~christos/classics/Codd72a.pdf.

El artículo introduce el álgebra y el cálculo relacionales. Se define un sublenguaje de datos como la componente de un lenguaje encargada de recuperar y almacenar datos. Este sublenguaje puede estar inmerso en un lenguaje de programación de propósito general o estar aislado. En esta última forma es un lenguaje de consulta.

Uno de los resultados más importantes que se presentan en el articulo es un algoritmo para traducir una expresión escrita utilizando el sublenguaje de datos ALPHA en otra equivalente del álgebra relacional.

Se muestra, en el artículo, que el álgebra relacional definida es relacionalmente completa. Un álgebra es relacionalmente completa "si dada cualquier colección finita de relaciones $R_1, R_2, ..., R_N$ en forma normal simple, las expresiones del álgebra permiten definir cualquier relación definible a partir de $R_1, R_2, ..., R_N$ por medio de expresiones ALPHA".

[Codd79] Codd E. F. Extending the Database Relational Model to Capture More Meaning. ACM Transaction on Database Systems (4)4: 397-434, 1979.

Con el propósito de dotar de más significado el modelo relacional se introducen, en este artículo, los conceptos de semántica atómica y molecular. Un esquema de clasificación en entidades, propiedades y asociaciones se presenta. Se propone una definición de base de datos completamente relacional apoyada en el cumplimiento de ciertas características.

El autor define una base de datos relacional, relaciones básicas y derivadas, llaves primarias y dominios primarios. Cualquier inserción, actualización o eliminación aplicada a relaciones básicas debe satisface las reglas de integridad de entidad e integridad referencial.

Se revisa de manera detallada el álgebra relacional que excluye valores nulos y tratando, por supuesto, las relaciones como conjuntos. Los valores nulos se tratan después en el artículo significando "valor existente pero desconocido de momento". Para recuperación de datos, se adopta una lógica de tres valores.

Lo que el autor denomina Modelo Relacional de Tasmania (RM/T) se introduce a partir de tipos de entidades, asociaciones y conceptos de agregación, entre otros.

[Chandra88] Chandra A.K. Theory of database queries. In Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 1-9, 1988.

El autor analiza una consulta a una base de datos desde el álgebra y la lógica de primer orden haciendo un importante énfasis en el poder expresivo de los lenguajes de consulta. A partir del reconocimiento de las limitaciones de la lógica de primer orden, y por tanto del álgebra, para expresar consultas (i.e. cierre transitivo), se revisa el concepto de consulta computable. A partir de este concepto se analiza la forma de construir un lenguaje que pueda expresar cualquier consulta computable.

[Date81] Date C. J. A Formal Definition of the Relational Model. ACM SIGMOD Record(13)1: 18 - 29, 1982.

Mediante un conjunto de reglas de producción se define una base de datos relacional. Los constructores que se introducen se revisan de manera detallada (i.e. ⟨relational-database⟩, ⟨domain⟩,⟨relation⟩,⟨attribute⟩). Se definen también, mediante reglas de producción operaciones relacionales. Finalmente, se introducen dos reglas de integridad: de entidad y referencial.

[SS75] Schmid H. A., Swenson J. R. On the Semantics of the Relational Data Model. Proceedings of the 1975 ACM SIGMOD International Conference on Management of data, San Jose, California, pp. 211-223, 1975.

Una de las fortalezas del modelo relacional es que las relaciones, atributos y operaciones están matemáticamente bien definidas. Sin embargo, no se ofrecen elementos sobre la forma de modelar mediante relaciones el mundo real. Este artículo presenta un modelo semántico que se aplica a la teoría relacional, solucionando algunos problemas asociados con la dificultad de expresar conocimiento sobre el mundo real mediante dependencias funcionales. El modelo propuesto está basado en tipos de objetos y sus relaciones y ofrece una forma de darle semántica a la normalización de relaciones.

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

Capítulo 3

OPTIMIZACIÓN DE CONSULTAS RELACIONALES

Optimizar una consulta implica escoger la forma más eficiente de ejecutarla. Se trata de minimizar los recursos necesarios para evaluar su correspondiente expresión de consulta [MCS88]. El problema de optimización exacta es computacionalmente intratable y como además no se dispone de información estadística precisa sobre la base de datos, los algoritmos de evaluación de consultas se apoyan en heurísticas [JK84]. Una expresión representando la consulta del usuario tiene una forma estándar, independientemente del contenido de la base de datos, y se transforma de manera que se mejore su evaluación. Después, ésta se convierte en secuencias de operaciones. Para cada operación se tiene una buena implementación con un costo asociado. Cada una de estas secuencias es un plan de acceso. El optimizador escoge el plan de acceso más barato y lo ejecuta.

El problema de optimización de consultas ha sido ampliamente estudiado para el modelo relacional de datos y es uno de los temas más importantes de investigación en bases de datos. Una buena parte de la investigación en optimización se ha centrado en las consultas conjuntivas. Se parte del supuesto de que el volumen de datos almacenado en la bd es tan grande que no se puede almacenar en la memoria principal. En este sentido, es necesario hacer copias de trozos de los datos almacenados y una vez obtenidos los resultados parciales y totales almacenarlos en disco. El intercambio de páginas entre el disco y la memoria principal es el costo, en términos de tiempo, más alto del procesamiento de una consulta.

Este capítulo está estructurado en tres secciones. Primero, se introduce el tema de optimización de consultas como un problema de búsqueda. Las fórmulas para estimar el tamaño de relaciones intermedias se ha tomado de [CBS98] y [GUW00]. También se describe el proceso de compilación de

una consulta. Finalmente, se discuten algunos aspectos prácticos relacionados con los lenguajes de consulta.

Introducción

Los SGBD le ofrecen al usuario una interfaz para expresar consultas mediante un lenguaje de consulta de alto nivel, generalmente SQL. Una consulta es una expresión en un lenguaje que permite describir los datos que se van a recuperar desde una base de datos [JK84]. Procesar una consulta SQL involucra dos componentes clave en un SGBD: el optimizador y el motor de ejecución de consultas [Chaudhuri98]. El motor de ejecución de la consulta produce un plan de ejecución con base en un conjunto de operadores físicos disponibles para producir una respuesta. El optimizador produce la entrada para el motor de ejecución, como se muestra en la Figura 3.1.

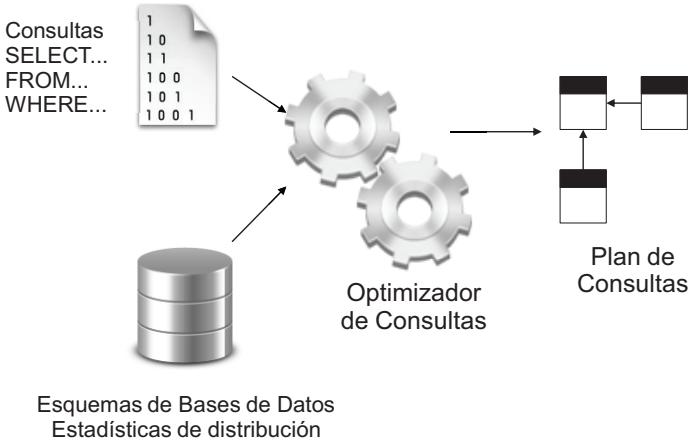

Figura 3.1 Ejecución de consultas

Basada en: http://technet.microsoft.com/en-us/library/ms190623 (SQL.100).aspx

El propósito de la optimización es minimizar el tiempo que requiere dar respuesta a una consulta. Para esto debe generar un eficiente plan de ejecución, que se convierte en un problema de búsqueda. Para resolverlo se requiere disponer de [Chaudhuri98]:

- Un espacio de planes (espacio de búsqueda)
- Una técnica para estimar el costo de cada plan
- Un algoritmo de enumeración para buscar en el espacio de planes.

Por esta razón el proceso de optimización no es trivial, ademas de que se apoya en el concepto de equivalencia entre consultas.

Un problema fundamental en la evaluación y optimización de consultas es el llamado *conjunctive-query containment*. A pesar de que este problema

es intratable [ChaMer77], cuando se imponen restricciones sobre las consultas conjuntivas se convierte en un problema tratable. La noción de equivalencia fuerte introducida en [ASU79] para dos expresiones conjuntivas o SPJ, se define como sigue:

Definición 3.1

Sea E una expresión del álgebra relacional cuyos operandos son esquemas de relación $R_1, R_2, ..., R_k$. Cada esquema de relación R_i tiene asociada una relación $r_i, 1 \le i \le k$. Para una asignación α de relaciones a esquemas de relación, el valor de E, $v_\alpha(E)$, se calcula aplicando los operadores relacionales en E sobre las relaciones operando. Si E es una expresión con operandos $R_1, R_2, ..., R_k$, $V(E)$ es una transformación que asocia el valor $v_\alpha(E)$ con la asignación α de las relaciones $r_1, r_2, ..., r_k$ a $R_1, R_2, ..., R_k$.

Dos expresiones E_1 y E_2 son fuertemente equivalentes si $V(E_1)$ y $V(E_2)$ son la misma transformación. Algunas equivalencias del álgebra relacional se listan en la sección "Generador de planes de consulta".

Aho et al. [ASU79] introducen también matrices especializadas o "tableaux", para representar transformaciones aplicables a expresiones relacionales que se definen a continuación.

Definición 3.2

Un *tableau* es una forma de definir el valor de una expresión. El *tableau* consiste de una matriz cuyas columnas son atributos del universo escritos en un orden fijo. Para una consulta conjuntiva, la primera fila del *tableau* es la lista de todos los atributos; la segunda fila, llamada *summary,* incluye a las variables distinguibles ($a's$) que corresponden a variables libres. El *summary* sólo puede contener variables distinguibles, blancos o constantes. Las demás filas del *tableau* pueden contener variables distinguibles o no distinguibles ($b's$) y constantes. Una misma variable distinguible no puede aparecer en diferentes columnas de la matriz, excepto si aparece en el *summary* de esa columna. Por notación, la variable distinguible a_i aparece en la columna correspondiente al atributo A_i.

Definición 3.3

Sea T un *tableau* y S el conjunto de todos los símbolos apareciendo en T. Una valuación ρ para T, le asocia a cada símbolo de S una constante, tal que si c es una constante en S, entonces $\rho(c) = c$.

Sea ω_0 el *summary* de T y $\omega_1, \omega_2, ..., \omega_n$ las filas. Entonces $\rho(\omega_i)$ es la tupla que se obtiene al sustituir $\rho(v)$ por cada variable v que aparece en ω_i

El esquema de relación objetivo de un *tableau* es el conjunto de atributos cuyas entradas no están en blanco en el *summary*. Si T es un *tableau* e I una instancia, entonces $T(I)$ es la relación sobre los atributos cuyas columnas no están en blanco en el *summary*, tal que:

$T(I) = \{\rho(\omega_0) \mid \textit{para alguna valuación } \rho\,,\, \rho(\omega_i) \textit{ está en I para } 1 \le i \le n\}$

Un ejemplo de *tableau*, tomado de [ASU79] se muestra en la Figura 3.2.

El *tableau* representa la siguiente relación definida sobre el esquema de relación AB:

$T(I) = \{a_1a_2 \mid (\exists b_1)(\exists b_2)(\exists b_3)(\exists b_4)\ a_1b_1b_3 \in I\,,\ b_2a_21 \in I \text{ y } b_2b_1b_4 \in I\}$, donde I es una instancia.

A	B	C
a_1	a_2	
a_1	b_1	b_3
b_2	a_2	1
b_2	b_1	b_4

Figura 3.2 Un ejemplo de tableau

Si I es la instancia {111, 222, 121} y si se asigna a todas las variables 1, las tres filas de T se vuelven 111, que pertenece a I. Por tanto, $\rho(a_1a_2) =$ 11 está en $T(I)$. Si a b_1 y a a_2 se les asigna 2 y 1 a las otras variables, todas las filas se convierten en 121 de tal manera que 12 está en $T(I)$. Si a a_1 y a b_1 se les asigna 2 y 1 a las otras variables, $\rho(a_1b_1b_3) = 222$ está en I y $\rho(b_2b_1b_4) = 121$ está en I, por tanto $\rho(a_1a_2) = 21$ está en $T(I)$.

Si se le asigna 1 a b_2, b_4 y 2 a las otras variables entonces 22 está en $T(I)$.

Por tanto $T(I) = \{11,12,21,22\}$

Aho et al. [ASU79] describen tanto la forma de construir un *tableau* para representar una expresión SPJ, como la manera de probar equivalencia entre *tableaux* mediante una transformación de las filas de un *tableau* en otro que preserva variables distinguibles y constantes y que garantiza que cualquier símbolo de variable no se transforma en dos símbolos de variables diferentes.

A pesar de que el problema de minimización y equivalencia entre *tableaux* es NP-completo, existe un subconjunto de *tableaux* para los cuales el problema se puede resolver en un tiempo limitado. Un *tableau* simple se define en [ASU79]. A partir de un *tableau* simple es posible encontrar una expresión óptima que ejecute selecciones y proyecciones tan pronto como sea posible.

El concepto de equivalencia es fundamental en el tema de optimización y, particularmente, los algoritmos de optimización se apoyan en el concepto de equivalencia, por ejemplo para empujar selecciones antes de *joins*, empujar proyecciones, entre otras, tal y como se muestra más adelante en la sección "Generador de planes de consulta".

El compilador de consultas

El procesador de consultas, componente del SGBD, convierte una consulta en una secuencia de operaciones que se ejecutan sobre una instancia de la base de datos. El procesador debe, por tanto, compilar la consulta, expresada en un lenguaje de alto nivel y ejecutarla. La compilación de una consulta se lleva a cabo en tres etapas, como se muestra en la Figura 3.3, tomada de [GUW00]. En la primera etapa, la consulta se descompone en un árbol que representa su estructura (*parse*). En la segunda etapa, este árbol se transforma en un árbol de expresión en álgebra relacional, que representa *el plan lógico* de la consulta. Finalmente, en la última etapa, el plan lógico se transforma en un *plan físico*. El plan físico describe las operaciones que se deben ejecutar, el orden de ejecución de estas operaciones, los algoritmos que se utilizan para cada operación y la forma en la cual se obtienen los datos y se pasan entre operaciones.

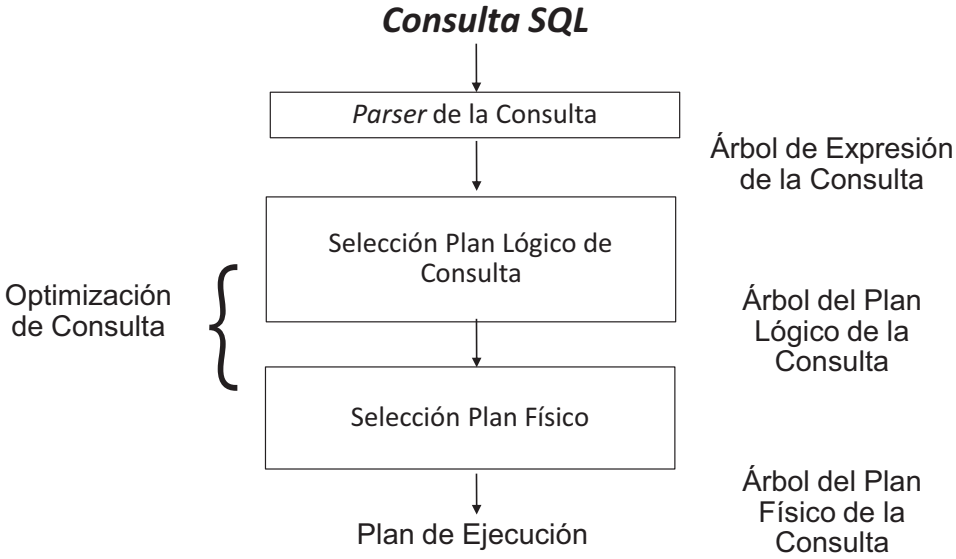

Figura 3.3 Compilación de una consulta

Algunas de las tareas que se llevan a cabo en la etapa de *Parse Query* en el proceso de compilación de consultas se muestran en la Figura 3.4.

El árbol parse

Un árbol *parse* se construye a partir de una expresión SQL, cuyos nodos son átomos o categorías sintácticas definidas por una gramática. Un ejemplo de parte de la gramática se describe a continuación.

Query ::= ⟨SFW⟩
Query ::= (⟨Query⟩)

La categoría sintáctica ⟨SFW⟩ se expresa como:

⟨SFW⟩::= SELECT ⟨SelList⟩ FROM ⟨FromList⟩ WHERE ⟨Condition⟩

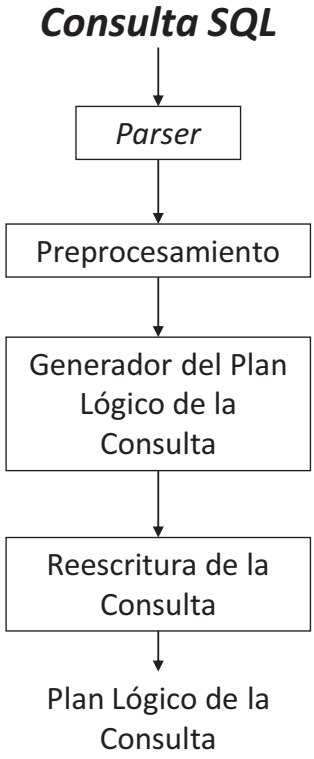

Figura 3.4 Transformación de una consulta

A su vez las categorías sintácticas ⟨SelList⟩, ⟨FromList⟩ y ⟨Condition⟩ corresponden a las siguientes reglas:

⟨SelList⟩ ::= ⟨Attribute⟩ , ⟨SelList⟩
⟨SelList⟩ ::= ⟨Attribute⟩
⟨FromList⟩ ::= ⟨Relations⟩ , ⟨FromList⟩
⟨FromList⟩ ::= ⟨Relations⟩

⟨Condition⟩ ::= ⟨Condition⟩ AND ⟨Condition⟩
⟨Condition⟩ ::= ⟨Tuple⟩ IN ⟨Query⟩
⟨Condition⟩ ::= ⟨Attribute⟩ = ⟨Attribute⟩
⟨Condition⟩ ::= ⟨Attribute⟩ LIKE ⟨Pattern⟩

Un átomo puede ser una palabra clave, un nombre de atributo, un nombre de relación, una constante o un operador como + o <. Las categorías sintácticas, representadas entre ⟨ ⟩, corresponden a nombres de familias de partes de una consulta. Los nodos que son átomos no tienen hijos y los que son categorías sintácticas tienen hijos dependiendo de las reglas de la gramática que tenga el lenguaje. En [CBS98] [GUW00] se presenta una gramática para un subconjunto de SQL. Un ejemplo de esta gramática se describe en [GUW00]. Las palabras clave se escriben en mayúscula.

El árbol de la Figura 3.5, tomada de [CBS98] [GUW00], corresponde a la siguiente consulta:

```
SELECT title
FROM StarsIn
WHERE starName IN (
      SELECT name
      FROM MovieStar
      WHERE birthday LIKE '%1960');
```

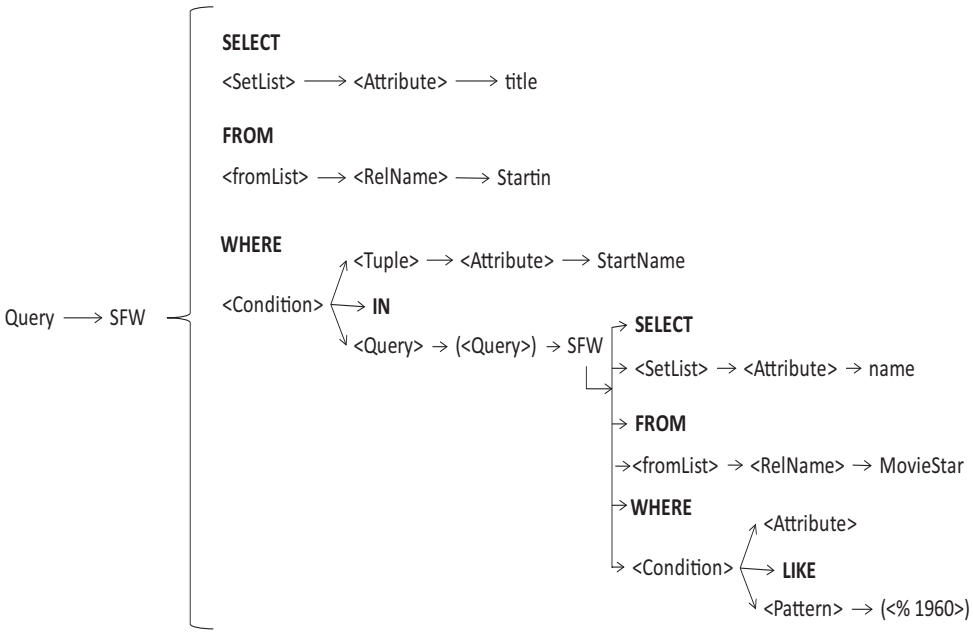

Figura 3.5 Árbol de consulta

El preprocesador de consultas

El preprocesador de consultas es responsable de:

- Reemplazar las relaciones que aparecen en el *From-list* en una consulta y que son vistas, por el árbol *parse* que describe la vista.
- Llevar a cabo el *semantic check.* Esta tarea incluye la verificación de la existencia, en el esquema, de las relaciones y vistas que aparecen el la cláusula FROM, de los atributos que aparecen en las cláusulas SELECT o WHERE, y la verificación de tipos de los atributos.

Generador de planes de consulta

A partir de una consulta, el generador de planes de consulta trabaja para mejorar la expresión algebraica correspondiente a la consulta aplicando propiedades que satisface el álgebra relacional. Algunas de estas leyes se listan a continuación:

Leyes conmutativas y asociativas

$R \times S = S \times R\,;\ (R \times S) \times T = R \times (S \times T)$

$R \bowtie S = S \bowtie R\,;\ (R \bowtie S) \bowtie \mathrm{T} = R \bowtie (\mathrm{S} \bowtie \mathrm{T})$

$R \cup S = S \cup R\,;\ (R \cup S) \cup \mathrm{T} = R \cup (S \cup \mathrm{T})$

$R \cap S = S \cap R\,;\ (R \cap S) \cap \mathrm{T} = R \cap (S \cap \mathrm{T})$

Estas leyes se aplican de manera diferente para conjuntos y para multiconjuntos.

Leyes incluyendo selección

Como la selección contribuye a reducir el tamaño de las relaciones se intenta mover las selecciones lo más abajo posible del árbol de consulta de manera que no se altere la expresión.

$\sigma_{C_1\ AND\ C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$

$\sigma_{C_1\ OR\ C_2}(R) = (\sigma_{C_1}(R)) \bigcup_S (\sigma_{C_2}(R))$

En el caso del *OR* la regla se satisface sólo cuando R es un conjunto.

$\sigma_{C_1}(\sigma_{C_2}(R)) = \sigma_{C_2}(\sigma_{C_1}(R))$

En relación con los operadores producto, unión, intersección, diferencia y *join*, el operador σ se aplica como sigue:

$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$

$\sigma_C(R-S)=\sigma_C(R)-S$ o $\sigma_C(R-S)=\sigma_C(R)-\sigma_C(S)$

Las siguientes leyes sólo se aplican cuando la relación tiene todos los atributos que aparecen en C.

$\sigma_C(R\times S)=\sigma_C(R)\times S$

$\sigma_C(R \bowtie S)=\sigma_C(R) \bowtie S$ o $\sigma_C(R\times S)=\sigma_C(R)\times\sigma_C(S)$ cuando R y S tienen ambos todos los atributos de C.

$\sigma_C(R\cap S)=\sigma_C(R)\cap S$

Leyes incluyendo proyección

Sea $E\rightarrow x$ un término de la lista de una proyección, donde E es un atributo o una expresión formada por atributos y constantes. Los atributos en E son atributos de entrada de la proyección y x es un atributo de salida.

$\pi_L(R \bowtie S)=\pi_L(\pi_M(R) \bowtie \pi_N(S)$, donde M es la lista de todos los atributos de R que pueden ser atributos del *join* o atributos de entrada de L, N es la lista de atributos de S que son atributos de *join* o atributos de entrada de L.

$\pi_L(R \overset{c}{\bowtie} S)=\pi_L(\pi_M(R) \overset{c}{\bowtie} \pi_N(S))$, donde M es la lista de todos los atributos que son o atributos de *join* o atributos de entrada de L y N es la lista de atributos de S que son o atributos de *join* o atributos de entrada de L.

$\pi_L(R\times S)=\pi_L(\pi_M(R)\times\pi_N(S))$, donde M y N son las listas de todos los atributos de R y S respectivamente, que son atributos de entrada de L.

Otras leyes se detallan en [CBS98][GUW00] para tratamiento de duplicados, agrupamiento y agregación.

Un ejemplo que ilustra la aplicación de leyes de transformación, tomado de [CBS98][GUW00] es el siguiente:

Sea $R(a,b,c)$ y $S(c,d,e)$ relaciones y $\pi_{a+e\rightarrow x,b\rightarrow y}(R \bowtie S)$ una expresión. En este caso los atributos de entrada son a,b,e. c es el atributo de *join*. Al aplicar las leyes se obtiene la expresión $\pi_{a+e\rightarrow x,b\rightarrow y}(\pi_{a,b,c}(R) \bowtie \pi_{c.e}(S))$ que es equivalente a la inicial.

Leyes para *join* y producto

$R \overset{c}{\bowtie} S=\sigma_C(R\times S)$

$R \bowtie S=\pi_L(\sigma_C(R\times S))$, donde C es una condición que iguala cada par de atributos de R y S con el mismo nombre y L una lista que incluye un atributo de cada par igualado y todos los otros atributos de R y S.

Un ejemplo de aplicación de algunas de estas leyes, tomado de [AHV95], es el siguiente, en el cual los esquemas de relación son *StarsIn(title, year,*

starName) y Movie(title, year, length, studioName) y *MoviesOf1996* es una vista definida como:

```
CREATE VIEW MoviesOf1996 AS
SELECT *
FROM Movie
WHERE year=1996
```

La consulta: "Which stars worked for which studios in 1996" corresponde a la siguiente expresión SQL:

```
SELECT starName, studioName
FROM MoviesOf1996 NATURAL JOIN StarsIn
```

La vista está definida por medio de la expresión algebraica
$S_{year=1996}(Movie)$

El plan lógico correspondiente a esta consulta se presenta en la Figura 3.6. Un plan equivalente, donde se empujan las selecciones, se muestra en la Figura 3.7.

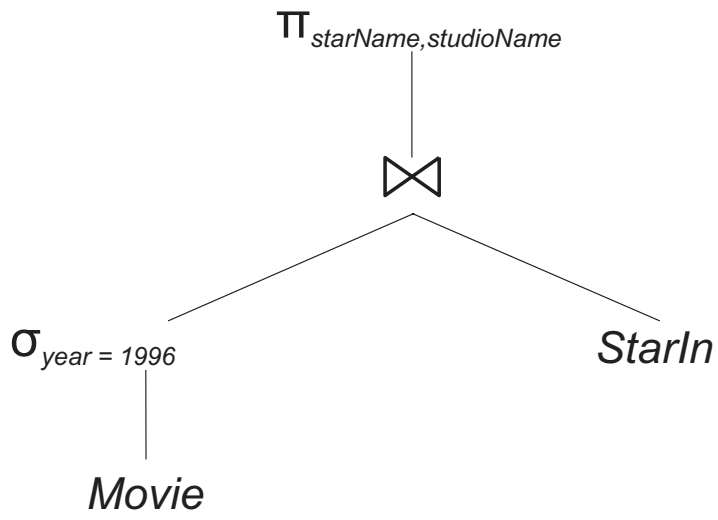

Figura 3.6 Plan lógico

Reescritura de la consulta

Después de que la consulta se transforma en una expresión del álgebra relacional, se aplican las leyes para obtener expresiones equivalentes. En esta etapa se selecciona un plan que se considere es el mejor. Una observa-

ción importante [CBS98][GUW00] se relaciona con el orden de los *joins.* Dependiendo de la forma de ordenar los *joins* y de si los operadores son conmutativos y asociativos, se generarán los diferentes planes.

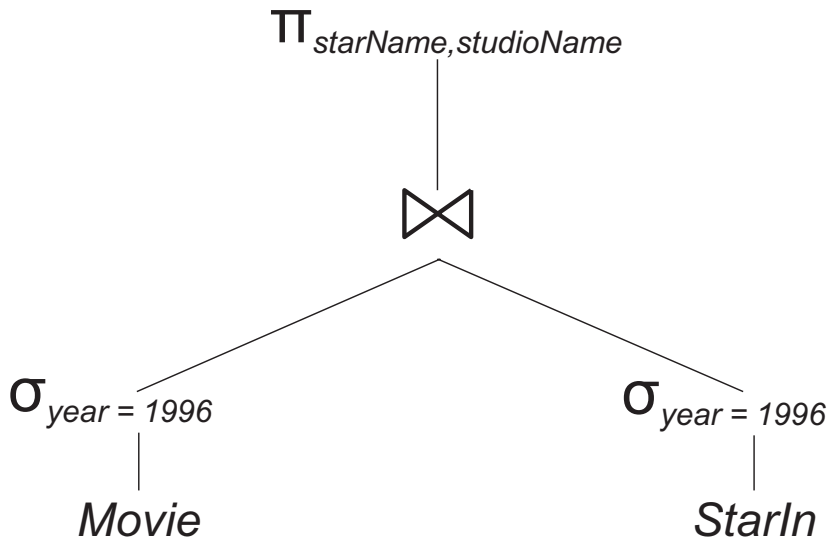

Figura 3.7 Plan lógico transformado

La decisión sobre el orden de los *joins* (más de dos relaciones) se considera como un problema de optimización basada en costo. Cuando se unen dos relaciones se recomienda escoger como argumento izquierdo la relación de menor tamaño. En aquellos casos en los cuales hay más de dos relaciones, el número de posibilidades para unirlos crece rápidamente. Como el orden es importante cuando se tienen n relaciones argumento, existe *n !* formas de unirlos.

En [Chaudhuri98] se presenta el enfoque de optimización para consultas SPJ (*Select-Project-Join*) muy relacionadas con las consultas conjuntivas en el contexto del optimizador de *System-R,* un SGBD relacional experimental que desarrolló IBM. Un ejemplo del orden de los *joins*, tomado de [Chaudhuri98], se muestra en la Figura 3.8 para la secuencia *Join(Join(Join(A,B),C),D)*.

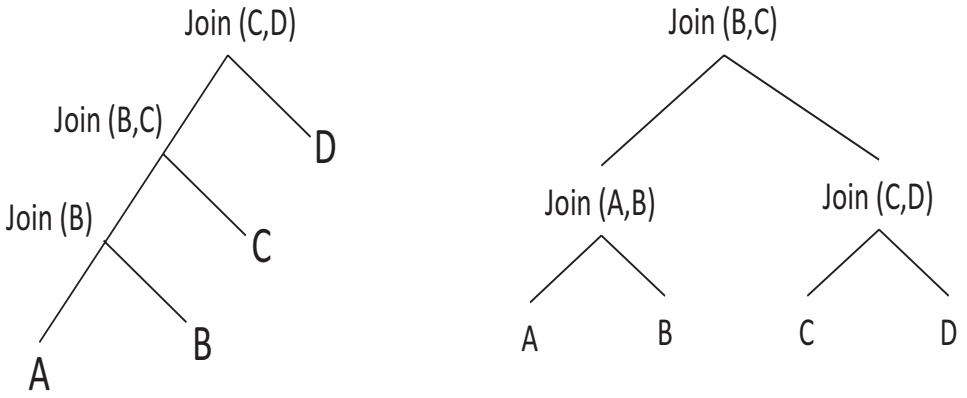

Figura 3.8 Ilustración de dos tipos de join

Por otra parte, los algoritmos de enumeración de planes se apoyan en técnicas heurísticas, programación dinámica y algoritmos genéticos, entre otros.

Finalmente, una vez se cuenta con el plan lógico de la consulta se requiere obtener un plan físico para la consulta. Esta etapa generalmente implica estimar el costo de evaluar expresiones a partir del número de operaciones de entrada y salida (I/O) a disco. Este costo depende de los operadores involucrados, del tamaño de las relaciones intermedias, de los operadores físicos que implementan los operadores lógicos, del orden en que se aplican operaciones iguales y de la forma en la que se pasan los argumentos entre los operadores físicos.

Cómo se estima el costo de los operadores: enumeración basada-en-costo

Después de que se obtiene un plan lógico preferido para una consulta, éste se debe transformar en un plan físico. El plan escogido corresponde a aquel cuyo costo estimado es el menor y es el que ejecuta el motor de ejecución de consultas. Este plan se escoge de entre varios que se derivan del plan lógico y cuyo costo se ha estimado.

Esta tarea implica obtener, por cada posible plan, el orden en el cual se agrupan operadores como *join*, unión e intersección, el algoritmo que se va a usar para cada operador, los operadores adicionales (i.e. *scanning, sorting*) y la forma en que se pasan los argumentos entre los operadores. Los costos se deben estimar para cada plan con base en ciertos parámetros de los datos.

- *Estimación del tamaño de relaciones intermedias*

El número de operaciones I/O que se necesitan para gestionar relaciones intermedias depende del número de tuplas de la relación y del tamaño de

cada tupla. Con base en el número de atributos de la relación y en el tipo de atributo se puede estimar el total de *bytes* por tupla. No es posible calcular exactamente cuántas tuplas tendrá la relación y por esta razón se aplican algunas reglas para estimar este valor.

Sean:

$B(R)$, el número de bloques que se necesitan para mantener las tuplas de R,

$T(R)$, el número de tuplas de la relación y

$V(R,a)$, el número de valores distintos del atributo a de R.

Si se consideran todos los atributos de R, $V(R,[a_1,a_2,...,a_n])$ representa el total de valores distintos que tiene R, considerando todos los atributos.

Proyección

Esta operación se puede calcular puesto que por cada tupla argumento, la salida sólo cambia en el tamaño de la tupla. Si se necesita eliminar duplicados se debe aplicar un operador δ con este propósito.

Selección

Este operador generalmente reduce el número de tuplas sin cambiar su tamaño. Dependiendo del tipo de selección, la estimación se calcula como sigue:

Sea $S = \sigma_{A=c}(R)$, siendo c una constante y A un atributo de R. La estimación del tamaño de la relación resultante utilizada es

$$T(S) = T(R)/V(R,A)$$

Esta estimación asume que los valores del atributo A aparecen con la misma frecuencia en la base de datos. Se pueden usar otras distribuciones. Cuando la selección utiliza una desigualdad, es más difícil hacer una estimación. Si el operador de comparación es $\neq$, [CBS98][GUW00] recomiendan asumir que todas las tuplas la satisfacen. Suponiendo que la desigualdad la cumple una parte de las tuplas, por ejemplo la tercera parte, $T(S)$ se podría estimar como:

$$T(S) = T(R)/3$$

En [CBS98][GUW00] se proponen fórmulas para estimaciones que incluyen en las condiciones conectivos lógicos como *AND* y *OR*.

Join natural

Para tratar de estimar el tamaño de la relación resultante de un *join* es necesario hacer algunos supuestos. Inicialmente se asume que el operador de comparación para el *join* $R(X,Y) \bowtie S(Y,Z)$, es la igualdad y que Y es

una relación simple de solamente un atributo. Para hacer esta estimación se requeriría determinar cómo están relacionados los valores de Y en R y S. En [CBS98],[GUW00] se presentan los siguientes casos de análisis:

- Si las relaciones tienen valores disjuntos con respecto a los valores de Y, $T(R \bowtie S) = 0$
- Si Y es la llave de S y llave foránea de R, cada tupla en R hace *join* con exactamente una tupla de S y $T(R \bowtie S) = T(R)$.
- Si casi todas las tuplas de R y S tienen los mismos valores de Y, $T(R \bowtie S)$ se puede estimar como $T(R)\ T(S)$.

Es necesario además trabajar bajo dos supuestos [CBS98],[GUW00]:

- *Inclusión de conjuntos de valores*
  Si Y es un atributo que aparece en varias relaciones, entonces cada relación toma sus valores de la cabeza de una lista fija de valores $y_1, y_2, y_3, ...$ y tiene todos los valores en ese prefijo. Por lo tanto, si R y S son dos relaciones con un atributo Y y $V(R,Y) \leq V(S,Y)$, entonces cada valor de Y en R será un valor de Y de S.

Este supuesto se satisface cuando Y es una llave de S y llave foránea en R; en otros casos no se satisface.

- Preservación *de conjuntos de valores*
  Si se calcula el *join* de una relación R con otra relación, entonces todo atributo A que no sea atributo de *join* no pierde valores del conjunto de posibles valores. Es decir, si A es un atributo de R pero no de S entonces $V(R \bowtie S, A) = V(R, A)$.

Este supuesto se satisface cuando los atributos de *join* de $R \bowtie S$ son llave para S y llave foránea para R.

- *Selección de plan basada en costo*

La evaluación de una expresión está determinada por el número de operaciones I/O a disco. Este número depende a su vez de los operadores lógicos que se hayan escogido para implementar la consulta, del tamaño de las relaciones intermedias, de los operadores físicos seleccionados para implementar operadores lógicos, del orden de operaciones similares y del método utilizado para pasar argumentos entre operadores físicos.

La estimación de algunos parámetros como el número de tuplas de la relación $T(R)$ y el total de valores de un atributo en la relación $V(R,A)$, se puede obtener a partir de las estadísticas que guarda el SGBD. Para esto es necesario, por ejemplo, escanear una relación para obtener su total de tuplas y de valores por atributo.

Por su parte, el total de bloques ocupados por la relación R, $B(R)$, se puede calcular contando el número de bloques ocupados o calculando aproximadamente este valor dividiendo $T(R)$ entre el tamaño de tupla. Es posible también pedirle al SGBD histogramas de valores de un atributo, lo que ayuda a estimar mejor el tamaño de un *join*.

Por otra parte, el uso de heurísticas como la de empujar selecciones lo más abajo posible en el árbol de consulta, facilita tomar decisiones en relación con si una transformación reduce o no el costo de un plan.

- *Enumeración de planes físicos*

Convertir un plan lógico en uno físico implica también estimar costos. Un enfoque básico, que es exhaustivo, considera todas las posibles combinaciones de opciones (e.g. orden de *joins*, implementaciones físicas de operadores) y a cada una le estima su costo. El de menor costo será el escogido.

Se puede también utilizar enfoques *top-down* o *bottom-up* para seleccionar planes físicos de consulta que requieren explorar el espacio de posibles planes físicos. Bajo el enfoque *top-down* [CBS98][GUW00], por ejemplo, se parte de la raíz del árbol correspondiente al plan lógico. Para cada implementación posible del operador de la raíz se calcula el costo de las posibles formas de evaluar sus argumentos.

Es posible aplicar heurísticas también para seleccionar un plan físico. Por ejemplo, una heurística voraz para ordenar *joins* se puede aplicar. Se escogen para *join* aquellas relaciones cuyo resultado estimado sea el menor y el proceso se repite sobre el resultado de ese *join* y otra relación del conjunto de relaciones a operar mediante *join*. Algunas de las heurísticas más utilizadas se detallan en [CBS98],[GUW00].

Aspectos prácticos de los lenguajes de consulta

Tomando como referencia el capítulo 7 del texto guía [AHV95], en esta sección se describe *Query-by-Example* [Zloof75] y se introduce SQL.

SQL (*Structure Query Language*) se desarrolló a partir de Sequel [CAE+85] lenguaje de consulta de System R [ABC+76]. Además de ofrecer facilidades para recuperación de datos basado en el bloque básico SELECT-FROM-WHERE, SQL ofrece también opciones para manipular datos (asignación, actualización, inserción, eliminación), crear y eliminar relaciones, vistas de relaciones e índices. Facilidades de control de datos también se ofrecen y que tienen relación con la integridad y con el acceso a los datos. Por otra parte, SQL se puede utilizar como un lenguaje *stand-alone* o embebido en un lenguaje anfitrión.

Query-by-Example (QBE) está orientado a usuarios no programadores. Las consultas se formulan llenando una tabla esqueleto con un ejemplo de una posible respuesta, al igual que las consultas basadas en *tableau*. En una

consulta se distinguen los elementos ejemplo o variables (subrayadas) de los elementos constantes (no subrayadas). Se puede insertar 'P.' en aquellos datos que se sean de salida.

Referencias

[ABC+76] Astrahan M. M., Blasgen M. W., Chamberlin D. D., Eswaran K. P., Gray J. N., Griffiths P. P., King W. F., Lorie R. A., McJones P. R., Mehl J. W., Putzolu G. R., Traiger I. L., Wade B. W., Watson V. System R: relational approach to database management. *ACM Transaction on Database Systems*(1)2: 96-137, 1976.

[ASU79] Aho A. V., Sagiv Y., Ullman J. D. Efficient Optimization of a Class of Relational Expressions. *ACM Transaction on Database Systems* (4)4: 435-454, 197.

[CAE+85] Chamberlin D. D., Astrahan M. M., Eswaran K. P., Griffiths P. P., Lorie R. A., Mehl J. W., Reisner P., Wade B. W. SEQUEL2: A Unified Approach to Data Definition, Manipulation and Control. *IBM J. RES. DEVELOP.,* 1985, Disponible en http://www.research.ibm.com/journal/rd/206/ibmrd2006E.pdf.

[CGM90] Chakravarthy U. S., Grant J., Minker J. Logic-Based Approach to Semantic Query Optimization. *ACM Transaction on Database Systems* (15)2: 162-207, 1990.

[ChaMer77] A. K. Chandra and P. M. Merlin. Optimal implementation on conjunctive queries in relational data bases. In *Proceeding of ACM SIGACT Symposium on the Theory of Computing*, pp. 77-90, 1977.

[Chaudhuri98] Chaudhuri S. An Overview of Query Optimization in Relational Systems. In *Proceedings of PODS 98,* pp. 34-43 Seattle USA, 1998.

[ChaudVar93] Chaudhuri S., Vardi M. Optimization of Real Conjunctive Queries. *Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,* Washington, D.C. HPL-9326, 1993.

[Graefe93] Graefe G. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys* (24)2: 73-169, June 1993. Disponible en http://infolab.stanford.edu/~widom/cs346/graefe.pdf.

[Ioannidis96] Ioannidis Yannis. Query Optimization. Disponible en http://infolab.stanford.edu/~widom/cs346/ioannidis.pdf.

[JK84] Jarke M., Koch J. Query Optimization in Database Systems. *ACM Computing Surveys* (16)2: 111-152, 1984.

[KS87] Kumar A., Stonebraker M. The Effect of Join Selectives on Optimal Nesting Order. *Sigmod Record* (16)1: 28-41, 1987.

[MCS88] Mannino M., Chu P., Sager T. Statistical Profile Estimation in Database Systems. *ACM Computing Survey* (20)3: 191-221, 1988.

[NPS91] Negri M., Pelagatti G., Sbatella L. Formal Semantics of SQL queries. *ACM Transactions on Database Systems*(16)3: 523-53, 1991.

[SACLP79] Selinger P. G., Astrahan M. M., Chamberlin D. D., Lorie R. A., Price T. G. Access Path Selection in a Relational Database Management System. *Proceedings of ACM SIGMOD International Conference on Management of Data,* pp. 23-34, 1979.

[Zloof75] Zloop Moshé. Query-by-Example: The Invocation and Definition of Tables and Forms. *Proceedings of the 1st International Conference on Very Large Data Bases,* Massachusetts, pp. 1-24, 1975.

Bibliografía básica anotada

[ASU79] Aho A.V., Sagiv Y., Ullman J.D. Efficient Optimization of a Class of Relational Expressions. *ACM Transaction on Database Systems* (4)4: 435-454, 1979.

Optimizar una consulta implica encontrar una expresión equivalente a esta que sea más barata de implementar computacionalmente. En este artículo se examina la dificultad computacional para determinar si dos consultas son equivalentes. El análisis se restringe a expresiones SPJ que generalmente cubre un conjunto amplio de consultas comunes.

Las transformaciones que se hacen a una expresión para convertirla en otra equivalente se basan en una matriz llamada por los autores un *tableau*. Un *tableau* es una matriz en la cual las columnas corresponden a los atributos del universo en un orden fijo.

[ChaudVar93] Chaudhuri S., Vardi M. Optimization of Real Conjunctive Queries. *Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Washington, D.C., HPL-9326, 1993.

El artículo examina las técnicas de optimización de consultas conjuntivas bajo una semántica de multiconjunto (*bag-theoretic setting*). Se introduce la equivalencia de consultas bajo esta semántica como base para encontrar expresiones equivalentes a una consulta que sean menos costosas computacionalmente.

[Chaudhuri98] Chaudhuri S. An Overview of Query Optimization in Relational Systems. In *Proceedings of PODS 98*, pp. 34-43, Seattle USA, 1998.

Se presenta en enfoque de optimización de *System-R*. El proceso de optimización se analiza considerándolo como un problema de búsqueda en un espacio de estados. El estudio hace énfasis en consultas conjuntivas (SPJ). El espacio de búsqueda para el optimizador son árboles de operadores que representan secuencias lineales de *joins y* que son equivalentes por las propiedades conmutativas y asociativas del *join*. El operador *join* se puede implementar, por ejemplo, como un *nested-loop* o como un *sort-merge*. Para cada plan parcial o completo se asigna un costo estimado mediante un modelo de costo. El modelo se apoya en estadísticas sobre las relaciones y los índices (número de valores distintos por atributo, número de páginas de datos por relación, etc.), en fórmulas para estimar el poder de selectividad de los predicados y en fórmulas para estimar costos de CPU e I/O de la ejecución de la consulta.

[CGM90] Chakravarthy U.S., Grant J., Minker J. Logic-Based Approach to Semantic Query Optimization. *ACM Transaction on Database Systems* (15)2: 162-207, 1990.

La gran mayoría de las propuestas de optimización de consultas se ha basado en el uso de propiedades de los operadores y en diferentes implementaciones de éstos, llamado por el autor optimización sintáctica. El proceso de optimización se vería positivamente mejorado si se utilizara información semántica sobre el dominio de las aplicaciones produciendo optimización semántica de consultas. En este artículo se proponen técnicas de optimización que tienen en cuenta conocimiento semántico.

La optimización semántica que se propone se aplica a bases de datos deductivas que también se puede aplicar a bases de datos relacionales, de acuerdo con los autores.

Una base de datos deductiva consta de una base de hechos (EDB, *extensional database*), reglas deductivas o axiomas (IDB, *intensional database*), un conjunto de restricciones de integridad (IC, *integrity constraint*) y la regla de inferencia SLDNF (*Linear resolution for definite Horn clauses with negation as failure*). Todas las cláusulas son de rango restringido. Una consulta es una cláusula objetivo o meta.

Una respuesta Q a una consulta $\leftarrow Q(x_1,...,x_n)$ es cualquier tupla $\langle a_1, a_2,..., a_n \rangle$, tal que $Q\langle a_1, a_2,..., a_n \rangle$ se puede probar a partir de la base de datos.

El enfoque de optimización propuesto consta de dos fases: modificación de la consulta usando la base de datos intensional y optimización del resultado aplicando técnicas convencionales.

[Ioannidis96] Ioannidis Yannis. Query Optimization. Disponible en http://infolab.stanford.edu/~widom/cs346/ioannidis.pdf.

Se centra en optimización de consultas conjuntivas conocidas también como consultas *Horn* no recursivas o consultas *select-project-join*, SPJ, en un sistema de gestión de base de datos centralizado. El recorrido de una consulta incluye su paso por varios módulos del SGBD como en la Figura 3.9.

Una abstracción del proceso de optimización se presenta integrada por dos etapas, tal y como se muestra en la Figura 3.10: *rewriting* y *planning*. El *rewriter* transforma la consulta de entrada para producir otras equivalentes y más eficientes. Algunas de las transformaciones que se aplican a la consulta son, entre otras, aplanar las consultas anidadas y reemplazar las vistas por su definición.

Por su parte, el *planner* analiza todos los planes de ejecución posibles y escoge el más barato. Para hacer esto necesita una estrategia de búsqueda que le permita examinar el espacio de planes. Junto con los módulos *algebraic space* y *method-structure space*, se determina el plan de más bajo costo con base en el costo de los planes que el módulo de *planner* entrega. Estos costos los calculan los módulos *Cost Model* y *Size-Distribution Estimator*.

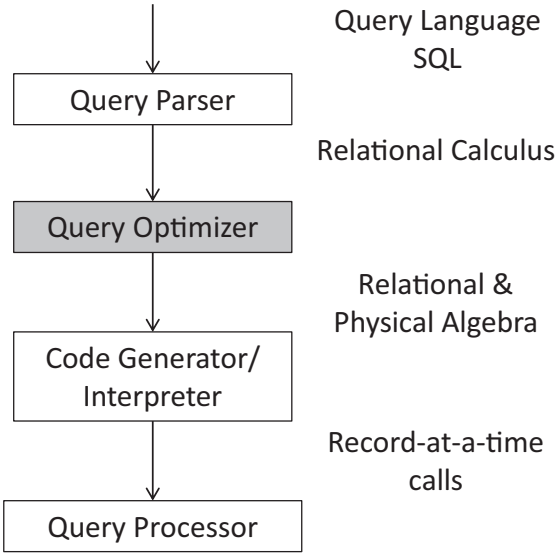

Figura 3.9 Recorrido de una consulta

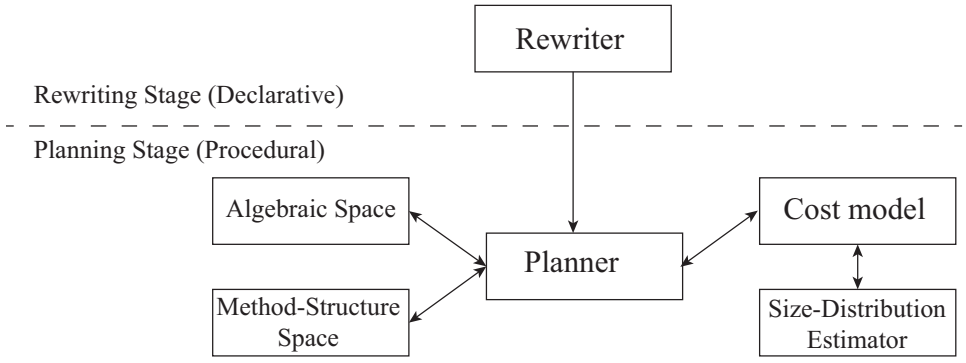

Figura 3.10 Arquitectura del optimizador de consultas

El *algebraic space* determina órdenes de ejecución de las acciones a considerar, que se representan en álgebra relacional en forma de árbol o de fórmula. El *method-structure space* produce todos los planes de ejecución posibles con base en las diferentes opciones de implementación existentes. El cálculo del costo de cada plan de ejecución lo determina el *cost model* a partir de fórmulas específicas que permiten calcular el costo de cada operación con base en disponibilidad de buffer y en los tamaños estimados de las relaciones, de los índices y de los resultados de las sub-consultas que ofrece el *size-distribution estimator*.

Una consulta del usuario se aplana y se representa como un árbol de consulta. Las hojas del árbol son relaciones de la base de datos y los nodos son operadores algebraicos. Los siguientes árboles de consulta de la Figura 3.11, tomados de [Ioannidis96] representan la siguiente consulta sobre las relaciones emp(name, age, sal, dno) y dept(dno, dname, floor, budget, mgr, anno).

```
SELECT name, floor
FROM emp, dept
WHERE emp.dno=dept.dno AND sal>100K
```

Como el espacio de búsqueda puede ser muy grande para una consulta, los SGBD lo reducen. En [Ioannidis96] se detallan las formas de hacer esta reducción. Para escoger el plan de ejecución más barato, el *planner* necesita utilizar una estrategia de búsqueda. Algunas de las más usadas se basan en programación dinámica. Otras estrategias propuestas se presentan en [Ionnidis96].

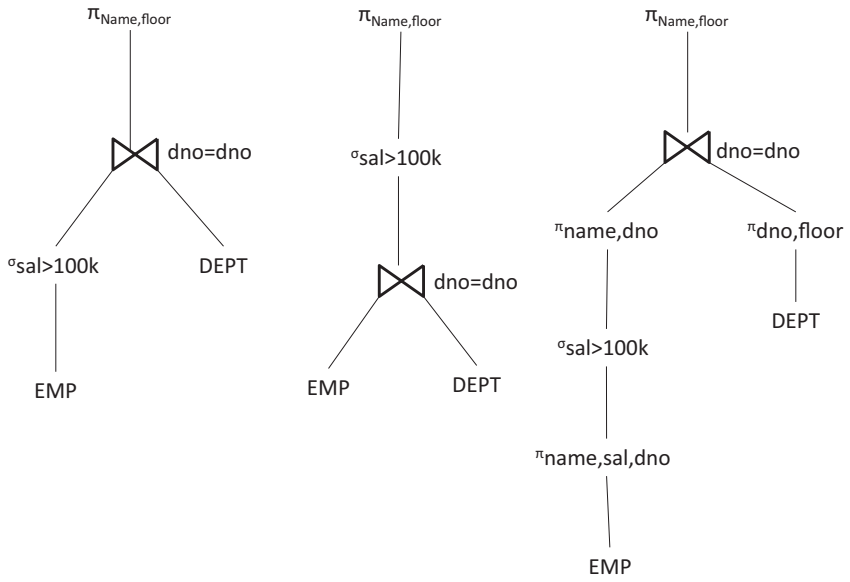

Figura 3.11 Árboles de consulta

[JK84] Jarke M., Koch J. Query Optimization in Database Systems. *ACM Computing Surveys* (16)2: 111-152, 1984.

Se revisan las técnicas de optimización en el marco del cálculo relacional para un SGBD centralizado. Optimizar una consulta significa minimizar el tiempo de respuesta a la consulta. Los costos que se deben minimizar incluyen los de transmisión de datos, los de cargar páginas desde dispositivos de almacenamiento secundario a la memoria principal, los de almacenamiento y los de CPU. En sistemas centralizados, el costo dominante es el tiempo de acceso a dispositivos de almacenamiento secundario. El enfoque de optimización propuesto es *top-down* que incluye los siguientes pasos:

1. Representar la consulta internamente de manera que el sistema tenga un nivel de libertad suficiente para llevar a cabo el proceso de optimización. Las consultas se pueden representar como expresiones del álgebra relacional, del cálculo relacional, como un grafo o en forma de *tableau*. Estas últimas sirven para representar consultas conjuntivas.
2. Aplicar transformaciones lógicas a la expresión de la consulta. Las transformaciones incluyen estandarización, simplificación y sustitución de expresiones por otras mejoradas.
3. Transformar la consulta en secuencias de operaciones elementales que tengan asociada una implementación y un costo (planes de acceso).
4. Calcular el costo total de cada plan, escoger el más barato y ejecutarlo.

Se tratan también temas relacionados con consultas múltiples, con el tratamiento de consultas de alto nivel, que incluyen varias relaciones, y con la gestión de consultas en bases de datos distribuidas.

[SACLP79] Selinger P. G., Astrahan M. M., Chamberlin D. D., Lorie R. A., Price T. G. Access Path Selection in a Relational Database Management System. *Proceedings of ACM SIGMOD Internacional Conference on Management of Data,* pp. 23-34, 1979.

El artículo describe la forma como el optimizador de *System R*, SGBD relacional, referente en temas de optimización de consultas, escoge caminos de acceso para consultas simples y complejas.

Después de que una consulta SQL ha pasado por las etapas de *parsing* y el optimizador dispone de estadísticas sobre las relaciones referenciadas en la consulta y los caminos de acceso existentes, éste debe seleccionar uno de estos últimos.

El artículo detalla la estrategia utilizada por *System R* para calcular el costo de cada plan de acceso tanto para consultas simples como complejas. Se ofrecen también fórmulas para estimar costos de acceso a la o las relaciones referenciadas que dependen de si la relaciones tiene o no índices de acceso definidos o si está clusterizada. Se describe también la forma de seleccionar caminos de acceso para operaciones de *join*, cuyo costo depende del orden en el cual se tomen las relaciones que aparecen en la consulta.

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

Capítulo 4

TEORÍA DE DEPENDENCIAS

El modelo relacional inicialmente propuesto por [Codd70] se enriquece semánticamente mediante la definición de restricciones de integridad. Varios tipos de restricciones han sido propuestas y particularmente las de dependencia de datos han sido ampliamente estudiadas [Codd70], [Fagin77], [BFH77], [CFP82]. Un interesante problema relacionado con las dependencias, que ha sido también estudiado ampliamente, tiene que ver con la implicación de dependencias a partir de un conjunto de éstas.

Por otra parte, el estudio de las dependencias también se relaciona con el diseño de esquemas de bases de datos. Si se parte de una relación universal y se aplican estrategias de descomposición para obtener formas normales, se pueden obtener buenos esquemas. Otra posibilidad de hacerlo es aplicar un modelo semántico, por ejemplo, el modelo entidad-relación [Chen76] para modelar una aplicación particular y obtener después un esquema relacional con dependencias.

En este capítulo se revisan las dependencias de datos. El capítulo está estructurado en cuatro secciones. Primero se definen las dependencias funcionales y las reglas de inferencia para generar otras dependencias implicadas a partir de las primeras. Luego se introducen las dependencias de inclusión y el conjunto de reglas para inferir otras. Las dependencias de unión se introducen después. En esta misma sección se definen las dependencias multivaluadas como un caso particular de una dependencia de unión. Se ofrecen reglas para inferir dependencias generadas con base en un conjunto de dependencias base. Al final se trata el tema de diseño de bases de datos a partir de dos principales enfoques. El primero, basado en dependencias, y el segundo, tomando como punto de partida un modelo de datos semánticamente más rico.

Dependencias funcionales

Las instancias de relación cambian en el tiempo. Se insertan, modifican y eliminan tuplas a las relaciones de la base de datos. Sin embargo, los esquemas de relación definidos, entre otros, a partir de nombres, tipos de atributos y restricciones de integridad son invariantes. Algunas de las propiedades de las relaciones están determinadas por asociaciones que existen entre los atributos de la relación.

Una de las restricciones de integridad básicas está fuertemente relacionada con el concepto de llave o clave. En el modelo relacional las llaves pueden ser primarias, candidatas o foráneas, significando, de manera precisa, el rol que juegan.

Una relación o tabla relacional tiene asociada exactamente una llave primaria (PK) k [Codd90], cuyo valor identifica una tupla o fila de manera única (propiedad de unicidad). Si la llave primaria es compuesta (combinación de atributos o columnas) y alguno de sus atributos se elimina, no se garantiza la propiedad de unicidad. Esta segunda propiedad se conoce como minimalidad.

Cuando una llave primaria, definida sobre el mismo dominio, aparece en otra relación, se denomina llave foránea (FK). La integridad referencial restringe la aparición de valores de las llaves foráneas y se puede definir como sigue. Esta restricción de integridad se aplica sobre un par de llaves, una primaria y otra foránea. La integridad referencial es un caso particular de las dependencias de inclusión [CFP82], que se definen más adelante.

Definición 4.1

Sea k una llave foránea que toma valores sobre un dominio D y sea D un dominio sobre el cual toman valores una o más llaves primarias. Cada valor de k debe existir en la base de datos como un valor de una llave primaria definida sobre el dominio D.

Definición 4.2

Una dependencia funcional es una expresión de la forma $X \rightarrow Y$, donde X y Y son conjuntos de atributos. Una relación R satisface la dependencia funcional $X \rightarrow Y$, denotada por $R \models X \rightarrow Y$ si y sólo si para cada par de filas μ y ν de R, $\mu[X] = \nu[X]$ implica $\mu[Y] = \nu[Y]$, es decir si $\pi_X(\mu) = \pi_X(\nu)$ entonces $\pi_Y(\mu) = \pi_Y(\nu)$.

La siguiente es una forma alternativa para definir una dependencia funcional [AHV95].

Sea U un conjunto de atributos. Una dependencia funcional sobre U es una expresión de la forma $X \rightarrow Y$, donde $X, Y \subseteq U$. Una dependencia de llave (*key dependency*) sobre U es una dependencia funcional que tiene la forma $X \rightarrow U$.

Una dependencia funcional sobre un esquema de base de datos ***R*** es una expresión de la forma $R : X \rightarrow Y$, donde $R \in \boldsymbol{R}$ y $X \rightarrow Y$ es una dependencia definida sobre $sort(R)$.

Definición 4.3

Sean $\sum$ un conjunto de dependencias funcionales (fd) y σ una dependencia funcional. Se dice que $\sum$ implica lógicamente a σ (o que σ es una consecuencia lógica de $\sum$), denotado por $\sum \models \sigma$, siempre que cada dependencia funcional en $\sum$ se satisface para una relación R entonces también se satisface para σ.

El cierre de un conjunto de dependencias funcionales $\sum$ sobre un conjunto de atributos U, que se denota por $\sum^{*,U}$ o $\sum^{*}$, se calcula así:

$$\sum^{*,U} = \{ X \rightarrow Y \mid XY \subseteq U \text{ y } \sum \models X \rightarrow Y \}$$

[Armstrong74] propuso un conjunto de reglas de inferencia, axiomas de Armstrong, para un conjunto de dependencias funcionales. Las reglas sirven para inferir otras dependencias o restricciones que son implicadas por el conjunto de dependencias inicial.

El conjunto de reglas es completo, lo que significa que cualquier dependencia o restricción implicada por el conjunto de dependencias se puede derivar de éste aplicando las reglas de inferencia.

Reglas de inferencia para dependencias funcionales (Axiomas de Armstrong)

Básicas

FD1 - Reflexividad

$$\forall X, X \rightarrow X$$

FD2 - Aumentatividad

$$\{ X \rightarrow Y, Z \supseteq X \} \Rightarrow Z \rightarrow Y$$

FD3 - Transitividad

$$\{ X \rightarrow Y, Y \rightarrow Z \} \Rightarrow X \rightarrow Z$$

Adicionales

FD4 - Descomposición

$$\{ X \rightarrow YZ \} \Rightarrow X \rightarrow Y, X \rightarrow Z$$

FD5 - Unión

$$\{X \to Y, X \to Z\} \Rightarrow X \to YZ$$

FD6 - Seudotransitividad

$$\{X \to Y, WY \to Z\} \Rightarrow WX \to Z$$

X, Y, Z, W son variables que toman valores en conjuntos de atributos.

Probando implicación lógica de dependencias funcionales

Calcular el cierre de un conjunto de dependencias es costoso computacionalmente, puesto que el número de dependencias derivadas de un conjunto de dependencias Σ es exponencial en relación con el número de atributos del esquema subyacente.

Sin embargo, se puede determinar, en tiempo lineal al tamaño de Σ, si para una dependencia $X \to Y$, $\Sigma \models X \to Y$. Esto se hace mediante el cálculo del cierre de un conjunto de atributos.

Definición 4.4

Sean Σ un conjunto de dependencias funcionales y $X \subseteq U$ un conjunto de atributos. El cierre de X bajo Σ, denotado por $(X, \Sigma)^{*,U}$ o simplemente X^*, cuando Σ y U se entienden del contexto se calcula como:

$$\{ A \in U \mid \Sigma \models X \to A \}$$

El siguiente lema es tomado de [AHV95].

Lema

Sea Σ un conjunto de dependencias funcionales y $X \to Y$ una dependencia funcional. Entonces $\Sigma \models X \to Y$ si $Y \subseteq X^*$. Mediante un algoritmo presentado en [AHV95] se puede calcular este conjunto.

Dependencias de inclusión (IND)

Una dependencia de inclusión (IND) [CFP82] se define sobre dos esquemas de relación, como sigue:

Definición 4.5

Sean $R_i[A_1,...,A_m]$ y $R_j[B_1,...,B_p]$ esquemas de relación no necesariamente distintos. Si X e Y son secuencias de k distintos elementos de $A_1,...,A_m$ y $B_1,...,B_p$, respectivamente, entonces $R_i[X] \subseteq R_j[Y]$ es una dependencia de inclusión.

Si $r_1,...,r_n$ es una instancia r de una base de datos definida sobre $\boldsymbol{R} = \{R_1[U_1],...,R_n[U_n]\}$, entonces r satisface la IND $R_i[X] \subseteq R_j[Y]$ sí $r_i[X] \subseteq r_j[Y]$.

Una completa axiomatización para dependencias de inclusión se presenta en [CFP82].

Reglas de inferencia para dependencias de inclusión

IND1 – Reflexividad

$R[X] \subseteq R[X]$, si X es una secuencia de atributos distintos de R.

IND2 - Proyección y permutación

Si $R[A_1,..., A_m] \subseteq S[B_1,..., B_m]$ entonces $R[A_{i_1},..., A_{i_k}] \subseteq S[B_{i_1},..., B_{i_k}]$ para cada secuencia $i_1,...,i_k$ de enteros distintos de $\{1,..., m\}$.

IND3 - Transitividad

Si $R[X] \subseteq S[Y]$ y $S[Y] \subseteq T[Z]$ entonces $R[X] \subseteq T[Z]$

Probando implicación lógica de dependencias de inclusión

En [CFP82] se propone un procedimiento de decisión para determinar si $\sum \models \sigma$, donde $\sum$ es un conjunto de IND y σ es una IND simple.

Cuando se combinan dependencias funcionales y de inclusión, se requieren nuevas reglas de inferencia. En [Mitchell83] se presenta un conjunto de reglas de inferencia que, de acuerdo con el autor, es confiable.

Dependencias *join* (de unión)

Esta sección es tomada del texto guía del curso [AHV95].

Las dependencias *join* o de unión están relacionadas con el operador *join natural* del álgebra relacional.

Para un conjunto de atributos U, conjuntos $X_1,..., X_n \subseteq U$ e instancias I_j definidas sobre X_j para $j \in [1,n]$, el *join* natural de las I_j se define como sigue:

$\bowtie_{j=1}^{n}\{I_j\} = \{s \mid s$ toma valores en $\bigcup X_j \mid \pi_{X_j}(s) \in I_j$ para cada $j \in [1,n]\}$

Una instancia I satisface una dependencia *join* si se puede expresar como el *join* de algunas de sus proyecciones.

Definición 4.6

Una dependencia *join* o de unión definida sobre un conjunto de atributos U, es una expresión de la forma $\bowtie[X_1,...X_n]$, donde $X_1,...X_n \subseteq U$ y $\bigcup_{i=1}^{n} X_i = U$.

I, definida sobre U satisface $\bowtie[X_1,...X_n]$, si $I = \bowtie_{j=1}^{n}\{\pi_{X_j}(I)\}$. La aridad de una dependencia de unión depende del número de conjuntos de atributos que la conforman.

Una relación R satisface la dependencia de unión (*join dependency*) $*[X_1,...X_m]$, si R es el *join* de sus proyecciones $R[X_1],...R[X_m]$ [Fagin79]. De acuerdo con Aho et al. [ABU79] el *join* de las proyecciones $R[X_1],...R[X_m]$ de R se define como:

$\{t \mid$ *existen tuplas* $w_1,...w_m$ *de* R *tal que* $w_i(X_i) = t(X_i)$ *para cada* $i, 1 \le i \le m\}$

La relación R satisface, por tanto, la dependencia $*[X_1,...X_m]$ si y sólo si R contiene cada tupla t para la cual existen tuplas $w_1,...w_m$ de R (no necesariamente distintas) tal que $w_i(X_i) = t(X_i)$ para cada i con $1 \le i \le m$.

Para familias de dependencias *join* o de unión, no existe axiomatización.

Las dependencias multivaluadas son un caso particular de una dependencia de unión. Fagin [Fagin77] las introduce como sigue:

Definición 4.7

Sea R una relación definida sobre conjuntos de atributos $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ donde $\mathbf{X} = \{X_1,...X_m\}$, $\mathbf{Y} = \{Y_1,...Y_n\}$ y $\mathbf{Z} = \{Z_1,...Z_r\}$. En $R(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ se asume que $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ son disjuntos dos a dos. $\mathbf{x}$ representa al conjunto de entradas $x_1,...,x_m$ para los atributos $X_1,...X_m$, análogamente para $\mathbf{y}$ y $\mathbf{z}$. Sea $\mathbf{Y_{XZ}} = \{y : (x,y,z) \in R\}$ que es no vacío si y sólo si $\mathbf{x}$ y $\mathbf{z}$ aparecen juntas en una tupla de R.

$R(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ satisface la dependencia multivaluada (mvd) $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$, si $\mathbf{Y_{XZ}}$ depende únicamente de $\mathbf{x}$. Fagin [Fagin77] prueba también que $R(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ satisface $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ si y sólo si R es el *join* de sus proyecciones $R_1(\mathbf{X}, \mathbf{Y})$ y $R_2(\mathbf{X}, \mathbf{Z})$. Por complementación, $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ se satisface en $R(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ si y sólo si $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ se satisface.

Expresada de otra forma, una dependencia multivaluada (mvd) definida sobre un conjunto de atributos U, es una expresión de la forma $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ con $X, Y \subseteq U$. Una relación I definida sobre U satisface $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ si $\bowtie[XY, X(U-Y)]$.

El siguiente conjunto de axiomas para fd y mvd fue propuesto por Beeri et al. [BFH77] cuya prueba de completitud se detalla en [BFH77].

Reglas de inferencia para dependencias multivaluadas

MVD0 - Complementación

Sean X, Y, Z conjuntos tales que su unión es U y $Y \cap Z \subseteq X$. Entonces $X \rightarrow\rightarrow Y$ si $X \rightarrow\rightarrow Z$

MVD1 - Reflexividad

Si $Y \subseteq X$ entonces $X \rightarrow\rightarrow Y$

MVD2 - Aumentatividad

Si $Z \subseteq W$ y $X \rightarrow\rightarrow Y$ entonces $XW \rightarrow\rightarrow YZ$

MVD3 - Transitividad

Si $X \rightarrow\rightarrow Y$ y $Y \rightarrow\rightarrow Z$ entonces $X \rightarrow\rightarrow Y - Z$

Cuando $Y \cap Z = \phi$ (Y, Z son disjuntos) la regla es la clásica transitividad [BFH77], Si $X \rightarrow\rightarrow Y$ y $Y \rightarrow\rightarrow Z$ entonces $X \rightarrow\rightarrow Z$.

Reglas de inferencia adicionales

MVD4 Seudotransitividad

Si $X \rightarrow\rightarrow Y$ y $YW \rightarrow\rightarrow Z$ entonces $XW \rightarrow\rightarrow Z - YW$

MDV5 - Unión

Si $X \rightarrow\rightarrow Y_1$ y $X \rightarrow\rightarrow Y_2$ entonces $X \rightarrow\rightarrow Y_1 Y_2$

MDV6 - Descomposición

Si $X \rightarrow\rightarrow Y_1$ y $X \rightarrow\rightarrow Y_2$ entonces $X \rightarrow\rightarrow Y_1 \cap Y_2$, $X \rightarrow\rightarrow Y_1 - Y_2$, $X \rightarrow\rightarrow Y_2 - Y_1$.

Reglas de inferencia adicionales para combinación de fd y mvd

Cuando existen dependencias funcionales y multivaluadas juntas, se requiere de tres reglas adicionales:

FD-MVD1 Si $X \rightarrow Y$ entonces $X \rightarrow\rightarrow Y$

FD-MVD2 Si $X \rightarrow\rightarrow Z$ y $Y \rightarrow Z'$ ($Z' \subseteq Z$), donde Y, Z son disjuntos, entonces $X \rightarrow Z'$

FD-MVD3 Si $X \rightarrow\rightarrow Y$ y $XY \rightarrow Z$ entonces $X \rightarrow Z - Y$

Por su parte, Maier et al. [MMS79] proponen un procedimiento, el *Chase*, para determinar dependencias implicadas por un conjunto de dependencias funcionales y de unión.

Diseño y dependencias

Diseñar un esquema relacional implica tener alguna buena forma de agrupar atributos en tablas para producir un esquema de la base de datos, con base en información semántica que tiene el diseñador sobre los datos [AHV95]. En [BST75], por ejemplo, se propone obtener esquemas relacionales a partir de la consideración de una dependencia funcional como un concepto elemental. En este enfoque, de refinamiento del esquema, la entrada consiste de un esquema inicial y de un conjunto de dependencias (funcionales y multivaluadas). Las dependencias se utilizan para mejorar el esquema mediante el concepto de Forma Normal. Se trata de obtener un esquema transformado que preserve los datos, los metadatos y que satisfaga formas normales deseables. Bernstein [Bernstein76], por su parte, propone un algoritmo para sintetizar un esquema de relación, en 3FN conteniendo un mínimo número de relaciones, a partir de relaciones funcionales entre atributos.

Otra forma en la cual se puede llevar a cabo el diseño de una base de datos es usar un modelo semántico más rico, que después se transforme en términos relacionales o mediante el refinamiento del esquema relacional. Los modelos semánticos ofrecen constructores más ricos semánticamente para modelar el mundo real y son fáciles de transformar en esquemas relacionales.

Cualquiera que sea el enfoque que se utilice, la tarea consiste en trasformar un esquema en otro relacional que preserve las dependencias entre los datos, satisfaga propiedades descritas por medio de formas normales y conserve información del esquema y de las dependencias (metadatos).

En esta sección, en la primera parte, se presenta el enfoque de transformación, basado en refinamiento de esquemas. En la segunda parte se introducen los modelos semánticos.

Diseño basado en refinamiento de esquemas

El proceso de normalización, estrategia básica de refinamiento de esquemas, fue introducido inicialmente por Codd en [Codd70]. El concepto de base de datos relacional se extiende para incluir dependencias en [DF92]. Un esquema de relación es un par (U,Σ) donde U es un conjunto de atributos y Σ es un conjunto de dependencias (restricciones) que considera solamente al conjunto de atributos en U. Se dice que una dependencia σ es una dependencia del esquema (U,Σ), si σ incluye sólo atributos de U y es consecuencia lógica de Σ.

Formas normales

Intuitivamente y de acuerdo con Date y Fagin [DF92], las formas normales se pueden ver como una manera de estructurar relaciones en una base de datos relacional exentas de ciertas anomalías. El proceso de normalización consiste en llevar un esquema de relación a su más alto nivel de normalización, transformándolo en un conjunto de esquemas cada uno de los cuales está en este nivel. Un esquema de relación se descompone en dos o más relaciones de manera que su composición pueda reconstruir la relación original. La propiedad de descomposición de una relación se define en [Delobel78] como sigue:

Definición 4.8

Sea $R(X,Y,Z)$ una relación. R se puede descomponer si existen relaciones S y T tal que:

a. S y T son proyecciones de R, $S = R[X,Y]$ y $T = R[X,Z]$

b. El *join* natural de S y T es R, $R = S * T$

La primera forma normal (1FN) restringe a que sea atómico el valor de cada atributo A de R, es decir el valor no puede ser un grupo repetitivo. Una relación no-normalizada se convierte a 1FN, eliminando dominios que tengan relaciones como elementos [Codd71]. La segunda (2NF) y la tercera forma normal (3NF) se introducen también en [Codd71] a partir de los conceptos de atributo primo (Definición 4.9), dependencia completa (definición 4.10) y dependencia no-transitiva (Definición 4.12).

Definición 4.9

Un atributo primo de una relación R es un atributo que participa a lo sumo en una llave candidata de R. Los otros atributos son no-primos.

Definición 4.10

Sean X,Y conjuntos distintos (no necesariamente disjuntos) de atributos de R y sea $X \rightarrow Y$. Si X no depende funcionalmente de ningún subconjunto de Y, entonces se dice que Y depende de manera completa de X en R.

Definición 4.11

Una relación R está en segunda forma normal (2FN) si está en 1FN y cada atributo no-primo de R depende completamente de cada llave candidata de R.

Definición 4.12

Sea R una relación definida sobre un conjunto de atributos U y sean $X \subseteq U$ y $A \in U$. A depende transitivamente de X si existe $Y \subseteq U$, tal que $X \rightarrow Y$, $Y \not\rightarrow X$, $Y \rightarrow A$ y $A \notin Y$.

Definición 4.13

Una relación R está en tercera forma normal (3FN) si está en 2FN y cada atributo no-primo de R no depende transitivamente de ninguna llave candidata de R.

Una relación está en 3FN [Zaniolo82] si cada atributo que dependa transitivamente de una llave, es una llave.

Cuando una relación en 3FN tiene múltiples llaves candidatas compuestas que se traslapan, la relación sigue teniendo anomalías a la hora de actualizarla. La forma normal de Boyce y Codd [Codd75] ofrece una forma más restrictiva que la 3FN basada en el concepto de determinante que resuelve esta anomalía.

Definición 4.14

Un atributo es un determinante en una relación, si existe algún otro atributo que depende de manera completa de éste.

Una definición alternativa propuesta en [DF92] para la 3FN es la siguiente:

Definición 4.15

Si A es un atributo que depende funcionalmente de algún conjunto X de atributos ($X \rightarrow A$), entonces una de las siguientes opciones se satisface:

a. $A \subseteq X$, y la dependencia es trivial,

b. X contiene una llave y por tanto X determina funcionalmente cada atributo,

c. X es parte de la llave

Si la opción c se elimina, se obtiene la Forma Normal Boyce-Codd (FNBC), más restrictiva que 3FN.

Definición 4.16

Una relación R esta en Forma Normal Boyce-Codd (BCNF), si para cada dependencia funcional no trivial $X \rightarrow A$, X es una superllave (un superconjunto de una llave) para R.

En [BB79] se analiza la dificultad de determinar si una relación en 3FN está en FNBC.

[Fagin77] introduce una nueva forma normal más fuerte que la 3FN. Esta nueva forma normal se define a partir del concepto de dependencia multivaluada trivial. Una dependencia multivaluada $A \rightarrow\rightarrow B$ en una relación R es trivial si $B \subseteq A$ o si $A \cup B = R$, donde A, B, C son atributos.

Generalizando la notación, como en [Fagin77], sean $\mathbf{X}, \mathbf{Y_1}, \mathbf{Y_2}, ..., \mathbf{Y_k}$ conjuntos que particionan los nombres de las columnas de $R(\mathbf{X}, \mathbf{Y_1}, \mathbf{Y_2}, ..., \mathbf{Y_k})$. $\mathbf{X} \rightarrow\rightarrow \mathbf{Y_1} \mid \mathbf{Y_2} \mid ... \mid \mathbf{Y_k}$, significa que $\mathbf{X} \rightarrow\rightarrow \mathbf{Y_i}$ se satisface para el conjunto $\mathbf{Y_i}$. Las dependencias $X \rightarrow\rightarrow \Phi$ y $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ son triviales y siempre se satisfacen para una relación $R(\mathbf{X}, \mathbf{Y})$.

Definición 4.17 4FN [Fagin77]

Un esquema de relación R está en 4FN, si siempre que la dependencia multivaluada trivial $\mathbf{X} \rightarrow\rightarrow \mathbf{Y}$ se satisface en R, entonces la dependencia funcional $\mathbf{X} \rightarrow \mathbf{A}$ se satisface para cada nombre de columna A de R.

Cuando una relación se descompone en dos nuevas relaciones, estas relaciones resultantes tienen la propiedad de unión sin pérdida (*lossless join*) lo que significa que la relación original se puede obtener a partir del *join* de las nuevas relaciones.

Esta propiedad de descomposición asegura que no se producen filas espurias cuando las relaciones nuevas se unen mediante el *join* natural. Aho et al. [ABU79] proponen algoritmos para determinar si el resultado del *join* de varias relaciones corresponde al esperado en presencia de dependencias funcionales y multivaluadas.

Cuando la descomposición produce más de dos relaciones es necesario entonces tratar con dependencias de unión (*join*) y con la 5FN. [Fagin79] introduce la forma normal de proyección-unión (*projection-join normal form*) o 5FN, en la que los únicos operadores permitidos son la proyección y el *join*.

En [Fagin79] se asumen las siguientes reglas de normalización que utilizan como únicos operadores la proyección y el *join*:

- Cada relación debe estar en 1FN
- Cuando una relación se reemplaza por un conjunto de relaciones, cada una de las nuevas relaciones es una proyección de la relación original
- La relación original debe ser el *join* de las nuevas relaciones.

Con base en un algoritmo de pertenencia y en el concepto de dependencias de llave, Fagin [Fagin79] define la 5FN como sigue:

Definición 4.18 5FN (Forma normal proyección-unión)

Un esquema de relación en 1FN R con atributos X está en forma norma PJ o 5FN, si para cada dependencia de unión σ en R existe una dependen-

cia de llave $K \rightarrow X$ en R tal que $K \rightarrow X \models \sigma$. Por tanto, cada dependencia de unión es el resultado de una llave.

El proceso de transformación de esquemas relacionales debería garantizar la preservación de dependencias, la satisfacción de propiedades del esquema final resultante expresado en términos de formas normales y la preservación de metadatos.

Diseño basado en modelos semánticos

Una colección de herramientas para describir las entidades del mundo real, integra un modelo de datos [SKS96]. El diseño de bases de datos relacionales basado en refinamiento de esquemas utilizando el concepto de normalización, asume homogeneidad de los datos. De acuerdo con Kent [Kent79], los modelos basados en registros, uno de cuyos principales representantes es el modelo relacional, tienen limitaciones e inconvenientes en relación con su capacidad para representar información semántica.

Para resolver estas limitaciones y como alternativa al diseño de bases de datos relacionales se propone utilizar un modelo conceptual de alto nivel, más rico en constructores para representar el mundo y para dar soporte a la percepción del mundo real, y después transformarlo en un esquema relacional.

De acuerdo con Hull y King [HK87], los primeros modelos semánticos se introdujeron como herramientas para ofrecer mecanismos de abstracción más complejos para especificar esquemas de bases de datos soportados por modelos basados en registros. Muchas propuestas de modelos semánticos surgen por tanto [Chen76] [HMc78] [Shen78] [HMc81] [TYF86] [AH87] [HK87] [HK88] [PM88] [SKS96].

El modelo entidad-relación [Chen76], por ejemplo, permite modelar el mundo real mediante entidades, atributos, valores de atributos y relaciones. A partir de un modelo E-R es fácil derivar un esquema relacional en 3FN. Una propuesta alternativa es extender las capacidades de un sistema relacional añadiéndole capacidades de modelos de datos semánticos.

En este sentido, Tsur y Zaniolo [TZ84] extienden el SGBD INGRES para dotarlo de constructores semánticos preservando la compatibilidad con sistemas relacionales existentes. Por ejemplo, es posible especificar que un atributo puede tener como valor el nulo añadiendo este valor al dominio del atributo. La operación de agregación se ofrece, también, mediante referencia a atributos.

En general, los modelos semánticos usan como componentes básicos objetos, atributos y relaciones entre objetos, constructores de tipos complejos y relaciones ISA. En particular, GSM (*Generic Semantic Model*) introducido en [HK87], representa los objetos o entidades del sistema mediante nodos triángulo asociados con tipos de datos abstractos. Un objeto tiene atributos asociados con tipos y subtipos que se representan por medio de

óvalos. Los tipos básicos se denominan *printables*. Usando flechas sencillas se unen dominio y rango de un atributo.

GSM incorpora también relaciones tipo-subtipo comúnmente llamadas relaciones ISA (is-a) que se representan con flechas dobles. Los subtipos de un tipo se representan con nodos en forma de círculo. Por otra parte y a partir de tipos atómicos, es posible construir otros tipos mediante agregación. Los tipos construidos mediante agregación se representan en el esquema con un nodo X. Un constructor adicional, de asociación o agrupamiento, que permite construir conjuntos de elementos de un tipo, se representa en el esquema por medio de un *.

Un ejemplo de un modelo GSM ilustrado en [HK87] se representa mediante un esquema relacional en 3FN, más un conjunto de dependencias de llave y de inclusión que capturan la semántica del esquema original. En [AHV95] se ilustra la conversión de un esquema de bases de datos diseñado aplicando GSM a otro esquema relacional.

Información incompleta

Los nulos (*null value*) son valores especiales que aparecen en las bases de datos del mundo real y que Codd [Codd79] interpreta con dos tipos de significado: "propiedad inaplicable" y "valor desconocido en el momento". Los valores nulos introducen en las bases de datos un valor adicional: el desconocido, a la hora de trabajar con expresiones lógicas. Una lógica de tres valores se propone y adopta cuando se recupera información de una base de datos. Se introduce para esto el valor de verdad "desconocido", denotado por ω. Las siguientes reglas, Tabla 4.1, para esta lógica de tres valores se presentan en [Codd79].

Adicionalmente, también en [Codd79] se proponen reglas adicionales para expresiones que involucren cuantificadores universales y existenciales, se define el principio de sustitución de valores nulos, consistente con la lógica de tres valores propuesta, y se analizan los operadores del álgebra relacional teniendo en cuenta el valor nulo.

El valor nulo es una especie de marca significando el hecho de que un valor en la base de datos es desconocido que puede representar un valor aplicable o inaplicable denotado por Codd [Codd90], respectivamente, como A-mark (applicable) e I-mark (inapplicable).

AND	*F*	ω	*T*
F	*F*	*F*	*F*
ω	*F*	ω	ω
T	*F*	ω	*T*

OR	*F*	ω	*T*
F	*F*	ω	*T*
ω	ω	ω	*T*
T	*T*	*T*	*T*

NOT(F) = T NOT(ω) = ω NOT(T) = F

Tabla 4.1 Reglas AND y OR para lógica de tres valores. Tomado de [Codd79]

Críticas a este modelo [Date2005] [Rubinson2007] se basan en que se pueden producir resultados erróneos a una consulta SQL cuando existen valores nulos. Esto debido a que el *null* es una bandera que indica que el valor de un atributo es desconocido y no un valor, puesto que los dominios no pueden incluirlo.

Referencias

[ABU79] Aho A. V., Beeri C., Ullman J. D. The Theory of Joins in Relational Databases. *ACM Transaction on Database Systems* (4)3: 297-314, 1979.

[AH87] Abiteboul S., Hull R. IFO: A Formal Semantic Database Model. *ACM Transaction on Database Systems* (12)4: 525-565, 1987.

[Armstrong74] Armstrong W.W. Dependency Structures of Database Relationships. *Proceedings of IFIP'74,* pp. 580-583, 1974.

[Bernstein76] Bernstein P. A. Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Transaction on Database Systems* (1)4: 277-298, 1976.

[BB79] Beeri C., Bernstein P. Computational Problems related to the design of Normal Form Relational Schemas. *ACM Transaction on Database Systems* (4)1: 30-59, 1979.

[BFH77] Beeri C., Fagin R., Howard J.H. A Complete Axiomatization for Functional and Multivalued Dependencias in Database Relations. *Proceedings on 1977 ACM SIGMOD International Conference on Management of Data*: 47-61, 1977.

[BST75] Bernstein P. A., Swenson J. R., Tsichritzis D. C. A Unified Approach to Functional Dependencies and Relations. *Proceedings on 1975 ACM SIGMOD International Conference on Management of Data,* pp. 237-245, 1975.

[CFP82] Casanova M. A., Fagin R., Papadimitriou C. H. Inclusion Dependencies and their interaction with functional dependencies. *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems,* pp. 171-176, 1982.

[Chen76] Chen P. P. The Entity-Relationship Model- Towards a Unified View of Data. *ACM Transactions on Database Systems (TODS)* (1)1: 9-36, 1976.

[Codd70] Codd E. F. A relational model of data for large shared data banks. *Communications of the ACM,* 13(6):377-387, 1970.

[Codd71] Codd E. F. Further Normalization of the Data Base Relational Model. *IBM Research Report, San Jose, California RJ909:* (1971). In ACM SIGMOD Anthology Volume 5, Issue 1, 1971.

[Codd75] Codd E. F. Recent Investigation in Relational Database Systems. *ACM Pacific 1975,* pp. 15-20. In ACM SIGMOD Anthology Volume 5, Issue 1.

[Codd79] Codd E. F. Extending the database relational model to capture more meaning. *ACM Trans. on Database Systems* 4(4):397-434, 1979.

[Codd87] Codd E. F. More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable information). *SIGMOD Record* (16)1:42-50, 1987.

[Codd90] Codd E. F. The Relational Model for Database Management: Version 2. Addison-Wesley Longman Publishing Co, Inc., 1990.

[Delobel78] Delobel C. Normalization and Hierarchical Dependencies in the Relational Data Model. *ACM Transaction on Database Systems* (3)3: 201-222, 1978.

[DF92] Date C. J., Fagin R. Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases. *ACM Transaction on Database Systems* (17)3: 465-476, 1992.

[Fagin77] Fagin R. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Transaction on Database Systems* (2)3: 262-278, 1977.

[Fagin77a] The decomposition versus synthetic approach to relational database design. *Proceedings of VLDB77,* pp. 441-446, 1977.

[Fagin79] Fagin R. Normal Forms and relational database operators. *SIGMOD'79, Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston Massachusetts,* pp. 153-160, 1979.

[Kent79] Kent W.. Limitations of Record-Based Information Models. *ACM Transaction on Database Systems* (4)1: 107-131, 1979.

[HMc78] Hammer M., McLeod D. The semantic data model: a modelling mechanism for database applications*. Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data* Austin, Texas pp. 26-36, 1978.

[HMc81] Hammer M., McLeod D. Database Description with SDM: A Semantic Database Model. *ACM Transaction on Database Systems* (6)3: 351-386, 1981.

[HK88] Hull R., King R. Semantic Database Modeling: Survey, Applications and Research Issues. *ACM Computing Surveys* (19)3: 202-260, 1987.

[Maier88] Maier David. The Theory of Relational Databases. Computer Science Press, 1988. Disponible en http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html.

[MMS79] Maier D., Mendelzon A., Sagiv Y. Testing Implications of Data Dependencies. *ACM Transaction on Database Systems* (4)4: 455-469, 1979.

[Mitchell83] Mitchell John C. Inference Rules for Functional and Inclusion Dependencies. *Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems,* pp. 58-69, 1983.

[PM88] Peckhan J., Maryanski F. Semantic Data Models. *ACM Computing Surveys* (20)3: 153-189, 1988.

[Rubinson2007] Rubinson C. Nulls, three-valued Logic, and ambiguity in SQL:critiquing date's critique. *ACM SIGMOD Record* (36)4: 13-18, 2007.

[SKS96] Silberschatz A., Korth H., Sudarshan S. Data Models. *ACM Computing Surveys* (28)1: 105-108, 1996.

[Shen78] Shen Stewart N. T. A Semantic Approach in designing relational data bases. *Proceedings of the ACM 1978 Annual Conference,* pp. 596-601, 1978.

[SS75] Schmid H. A., Swenson J. R. On the semantic of relational data model. *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data, San Jose, California,* pp. 211-223, 1975.

[TYF86] Teorey T. J., Yang D. Fry J. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *Computing Surveys* (18)2:197-222, 1986.

[TZ84] Tsur S. Zaniolo C. An implementation of GEM: supporting a semantic data model on a relational back-end. *ACM SIGMOD Record* (14)2: 286-295, 1984.

[Vassiliou79] Vassiliou Y. Null Values in Data Base Management: A Denotational Semantics Approach. *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data,* Boston, Mass., pp. 162-169, 1979.

[Zaniolo82] Zaniolo C. A New Normal Form for the Design of Relational Database Schema. *ACM Transaction on Database Systems* (7)3: 489-499, 1982.

Bibliografía básica anotada

[Armstrong74] Armstrong W.W. Dependency Structures of Database Relationships. *Proceedings of IFIP'74,* pp. 580-583, 1974.

Se propone un conjunto de reglas de inferencia o axiomas para derivar dependencias funcionales (restricciones) a partir de un conjunto de dependencias funcionales inicial. Se prueba que este conjunto de reglas es completo.

[BB79] Beeri C., Bernstein P. Computational Problems related to the design of Normal Form Relational Schemas. *ACM Transaction on Database Systems* (4)1: 30-59, 1979.

El artículo analiza varios problemas relacionados con dependencias funcionales y diseño de esquemas.

En relación con la derivación de dependencias funcionales a partir de un conjunto de dependencias, se analiza el inconveniente de ser lineal, es decir, que pueden existir varias derivaciones de una df que solo difieran en el orden de aplicación de las reglas. Adicionalmente, en una derivación se pueden, por una parte, aplicar reglas que son redundantes y por otra, aplicar reglas necesarias formalmente (reflexividad y aumentatividad), pero que intuitivamente no agregan información nueva. Para facilitar la manipulación de dependencias funcionales se introduce, en el artículo, un árbol de derivación que, de acuerdo con los autores, elimina estos inconvenientes.

Con respecto al diseño de esquemas en 3FN, a pesar de que es posible construirlos algorítmicamente con base en dependencias funcionales, no es posible extenderlos para alcanzar esquema en FNBC. Varias razones se presentan:

No todo conjunto de dependencias funcionales se puede representar mediante un esquema relacional en FNBC

Es computacionalmente difícil (NP-hard) determinar si un esquema relacional representando un conjunto de dependencias funcionales está en FNBC

Es computacionalmente muy difícil determinar si un conjunto de dependencias funcionales dado se puede representar mediante un esquema FNBC.

Se muestra que los problemas de determinar si una relación en 3FN está en FNBC y el de encontrar llaves, asociados con diseño de esquemas, son NP-completos. Se proponen además algoritmos para el problema de pertenencia de una dependencia funcional a un conjunto de dependencias (*membership problem*) y para obtener esquemas relacionales a partir de un conjunto de dependencias.

[BFH77] Beeri C., Fagin R., Howard J.H. A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations. *Proceedings on 1977 ACM SIGMOD International Conference on Management of Data,* pp. 47-61, 1977.

En este articulo se presentan conjuntos de reglas de inferencia aplicables a conjuntos de dependencias funcionales, multivaluadas y combinadas. Los autores

prueban que los conjuntos de reglas son completas para la familia de dependencias funcionales y multivaluadas.

[CFP82] Casanova M. A., Fagin R., Papadimitriou C. H. Inclusion dependencies and their interaction with functional dependencies. *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems:* 171-176, 1982.

Se estudian las dependencias de inclusión (Ind) y su interacción con dependencias funcionales. En una dependencia de inclusión la proyección sobre m columnas de la relación R es un subconjunto de la proyección de m columnas de la relación S. Desde un punto de vista de diseño de bases de datos, las dependencias de inclusión permiten decidir selectivamente cuales datos deben estar duplicados y en cuales relaciones.

Una axiomatización y un procedimiento de decisión para *Ind* se presentan. Adicionalmente, se demuestra también que para un conjunto de dependencias *Ind* y *fd* no existe una axiomatización completa.

[DF92] Date C. J., Fagin R. Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases. *ACM Transaction on Database Systems* (17)3: 465-476, 1992.

Se introduce la forma normal DK/NF, *domain-key normal form*, basada en los conceptos de dominio y llave.

[Delobel78] Delobel Claude. Normalization and Hierarchical Dependencies in the Relational Data Model. *ACM Transaction on Database Systems* (3)3: 201-222, 1978.

Haciendo uso de las dependencias funcionales y multivaluadas se introduce un modelo de diseño lógico de bases de datos relacionales. Adicionalmente, se define una nueva dependencia denominada por el autor de descomposición jerárquica de primer orden (*first-order hierarchical decomposition-FOHD*) que se relaciona con una organización jerárquica de los datos.

A partir del concepto de descomposición de una relación se estudian las condiciones bajo las cuales una relación se puede descomponer y la relación que existe entre descomposición y dependencias (funcionales, multivaluadas y FOHD).

Una FOHD es una expresión denotada como $X : Y_1 \mid Y_2 \mid ... \mid Y_k$, donde $X, Y_1, Y_2, ..., Y_k$ son conjuntos disjuntos. Una relación $R(X, Y_1, Y_2, ..., Y_k, W)$, donde $X, Y_1, Y_2, ..., Y_k, W$ son conjuntos disjuntos de atributos satisface la FOHD si para cada X -valor se tiene que: $R[X, Y_1, Y_2, ..., Y_k, W] = R[x, Y_1] \times R[x, Y_2] \times ... \times R[x, Y_k]$.

En una FOHD $X : Y_1 \mid Y_2 \mid ... \mid Y_k$, X se denomina segmento raíz, $Y_1, Y_2, ..., Y_k$ segmentos y el par (X, Y_i) se denomina rama.

Algunas propiedades entre las dependencias funcionales, multivaluadas y

FOHD también se estudian. Finalmente, se ilustran descomposiciones que se derivan de una relación original.

[Fagin77] Fagin Ronald. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Transaction on Database Systems* (2)3: 262-278, 1977.

Se estudian reglas de inferencia para dependencias funcionales y multivaluadas existentes en una relación. Se analizan conjuntos de reglas para los df, los mvd y para combinaciones de éstas. Se muestra además que el conjunto de reglas es completo para una familia de dependencias funcionales y multivaluadas.

[Fagin81] Fagin R. A Normal Form for Relational Databases That is based on Domains and Keys. *ACM Transaction on Database Systems* (6)3: 387-415, 1981.

Se define la forma normal Proyección-Unión (*Projection-Join Normal Form PJ/NF*), una forma normal en la cual sólo se permiten los operadores proyección y unión, y que es más restrictiva que la 4FN y que se corresponde con la 5FN.

[Maier88] Maier David. The Theory of Relational Databases. Computer Science Press, 1988. Disponible en http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html.

En relación con el razonamiento sobre dependencias se dedica, en este libro, el Capítulo 8 a la revisión del *tableau* como base para el *chase*, un mecanismo para razonar sobre dependencias. Un *tableau* se define como una tabla conteniendo variables tomadas de un conjunto V y cuyo esquema está conformado por nombres de atributos asignados a cada una de sus columnas. V está compuesto de variables distinguibles y no-distinguibles. El esquema R de un *tableau* se convierte en una plantilla del esquema R.

Mediante el *chase* es posible encontrar para un *tableau* y un conjunto de dependencias dado, otro *tableau* equivalente al *tableau* inicial que satisface el conjunto de dependencias.

[Zaniolo82] Zaniolo C. A New Normal Form for the Design of Relational Database Schema. *ACM Transaction on Database Systems* (7)3: 489-499, 1982.

Se aborda el problema de diseño de bases de datos en el marco del modelo relacional y de dependencias funcionales. Teniendo en cuenta algunos problemas y limitaciones de la 3FN y FNBC, el autor introduce una definición de la 3FN.

Mediante ejemplos se ilustran las limitaciones de la FNBC en relación con la 3FN. En particular, una de ellas es la existencia de algoritmo eficiente para obtener esquemas de relaciones en 3FN y no para el caso de la FNBC.

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

Capítulo 5

MODELOS DE DATOS DE TERCERA GENERACIÓN

Los sistemas de gestión de bases de datos relacionales, considerados de segunda generación, se han utilizado, generalmente, para modelar aplicaciones de procesamiento de datos orientados a los negocios. A pesar de que éstos tienen como fortaleza su fundamentación teórica, no ofrecen las funcionalidades requeridas en aplicaciones relativamente complejas, no necesariamente orientadas a los negocios. Algunas de estas características, de acuerdo con [PBRV90] son los tipos abstractos de datos y las restricciones de integridad complejas, entre otras.

Las limitaciones del modelo relacional, como principal representante de los modelos orientados a registros, generaron la necesidad de ofrecer a los usuarios modelos semánticamente más ricos y más flexibles a la hora de diseñar aplicaciones en ciertos dominios. En particular, una de las limitaciones del modelo relacional analizadas por [Kent79] se relaciona con la dificultad para modelar información semántica de manera precisa.

En particular, en algunos dominios tales como el diseño en ingeniería (e.g. CAD/CAM, CASE), se requiere que los sistemas de gestión de bases de datos le ofrezcan al usuario otras funcionalidades para apoyar el desarrollo de esta nueva generación de aplicaciones. Por otra parte, los lenguajes de programación pueden ofrecer estas y otras facilidades pero adolecen de la persistencia y de acceso concurrente a los datos.

Varios enfoques, post-relacionales, se proponen para hacer frente a estas limitaciones. Por una parte, los *modelos semánticos*, precursores de los modelos de bases de datos orientados a objetos, que ofrecen formas más ricas para describir el mundo, surgen como alternativa de solución. Como mecanismo para representar interrelaciones estructurales complejas entre los datos estos modelos cuentan con un conjunto de constructores.

Otro de los primeros enfoques de solución parte del modelo de datos relacional, permitiendo relaciones más complejas, estructuralmente conocidas como *relaciones no normalizadas o anidadas*. Este tipo de enfoque permite que el constructor de relación se pueda aplicar repetidamente. Debido a la necesidad de contar con estructuras de datos que faciliten el modelamiento de aplicaciones no orientadas a negocios, Makinouchi [Makinouchi77] introduce relaciones no-normalizadas al modelo relacional que mantienen la consistencia con la teoría de la 3FN. Esta es una de las primeras propuestas para permitir que una entrada en una relación pueda ser a su vez un conjunto. En esta propuesta los dominios asociados a las columnas de una relación pueden ser conjuntos de conjuntos. Este tipo de columnas se denomina columnas relación, para distinguirlas de las columnas cuyos dominios son atómicos. De la misma forma las relaciones pueden ser independientes o dependientes si algunas de sus columnas son a su vez un conjunto.

Jaeschke y Schek [JS82], por su parte, introducen relaciones NF^2 (*Non-First - Normal-Form*) y definen los operadores NEST y UNNEST para este tipo de relaciones, como extensión del álgebra relacional. Cuando se trabaja con relaciones NF^2, no es necesario tratar con dependencias multivaluadas. El operador NEST transforma relaciones en 1FN a NF^2 y el UNNEST justo produce el resultado contrario.

Por otra parte, los modelos de objetos complejos son más flexibles que los de relaciones no normalizadas en el sentido en que los constructores de tupla y conjunto (algunos ofrecen constructores adicionales como *list* y *array*) se pueden aplicar arbitrariamente para producir *objetos complejos* [AB95].

Dos enfoques populares en la comunidad de bases de datos se propusieron, por otra parte, para abordar esta necesidad: el primero, incluye características objeto en un *nuevo modelo de gestión de bases de datos* y el segundo, enriquece el modelo propuesto por Codd produciendo un modelo *relacional extendido*.

El nuevo modelo de datos, el modelo de datos orientado a objetos, incorpora características de los lenguajes de programación orientados a objetos, de los modelos semánticos, de valor complejo y del relacional. De acuerdo con Lord y Gupta [LG02], los SGBDOO ofrecen ventajas en relación con los sistemas relacionales, como la rapidez de navegación, reutilización de código y la cercanía entre la representación del mundo real y la variedad de constructores semánticos y de relaciones. Su línea de tiempo se muestra en la Figura 5.1, tomada de [LG02].

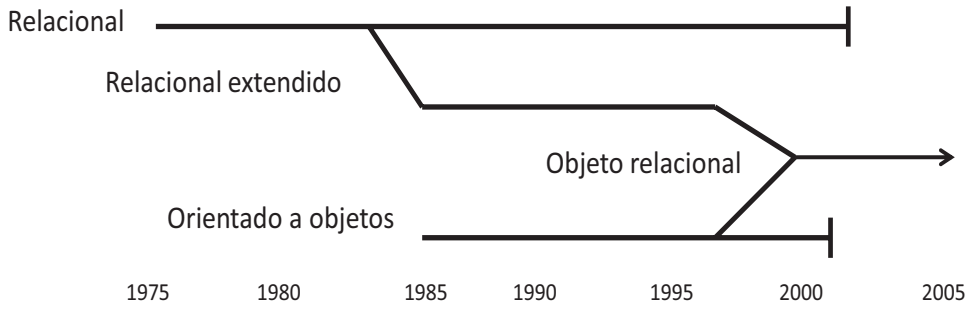

Figura 5.1 Línea de tiempo de tecnologías de bases de datos

Por su parte, el modelo relacional extendido, conocido también como objeto-relacional, es un modelo híbrido en el cual se incluyen conceptos orientados a objetos al modelo relacional. De esta manera, se puede obtener un sistema de gestión de bases de datos de tercera generación.

En varias propuestas de desarrollo de sistemas de gestión de bases de datos [Stonebraker86], [SR86], [CR86], [CDV88], [CD86], [CDGH+88], [LLPS91], [DD95], [IMBDB2], [InformixDS], se intenta extender sus funcionalidades para dotarlos de estructuras más ricas (tipos de datos y funciones definidas por el usuario) y de reglas para enriquecer la semántica de la base de datos. Las extensiones, de acuerdo con [CH90], se deben dar en diferentes niveles. En el nivel de interfaz de usuario se debe ofrecer soporte a tipos abstractos de datos (ADT) y lenguajes de consulta con nuevos operadores. En el bajo nivel se requiere de implementaciones para estos nuevos operadores, métodos de acceso y nuevas estructuras de índices. Las distintas propuestas de extensión como POSTGRES, Starburst, PROBE, EXODUS, Sabrina, R^2D^2, entre otros, se analizan con respecto a estas extensiones en [CH90].

De acuerdo con [CMCD94], entre las nuevas funcionalidades que tendría que tener un sistema de gestión de bases de datos para dar soporte a estos nuevos requerimientos están:

- Un subsistema rico y extensible de tipos de datos integrado con el lenguaje de consulta SQL. Este subsistema debe permitirle al usuario crear tipos de datos y funciones que encapsulen la conducta de objetos complejos,
- Un eficiente subsistema de reglas para proteger la integridad de la base de datos, enriquecer la semántica de la base de datos, y
- Un mejoramiento en el desempeño orientado a la gestión de objetos complejos (*e.g.* alto grado de interactividad, necesidad de acceso navegacional a los datos, extensión de lenguajes de consulta con mayor poder expresivo, mejoramiento de los mecanismos de acceso a los datos).

Stonebraker et al. [SRLG+94] proponen los siguientes tres principios básicos para guiar el desarrollo de los sistemas de gestión de bases de datos (DBMS) de tercera generación:

- **Principio 1:** Los DBMS de tercera generación, además de ofrecer servicios tradicionales de gestión de datos, deberán dar soporte a estructuras ricas de objetos y reglas.
- **Principio 2:** La tercera generación debe incluir a la segunda.
- **Principio 3:** La tercera generación debe estar abierta a otros subsistemas (e.g. lenguajes de cuarta generación, interfaces de paquetes gráficos, sistemas distribuidos, sistemas de soporte a la decisión, lenguajes de programación).

Trece proposiciones relacionadas con estos tres principios se detallan también con el propósito de asegurar el éxito de los sistemas de tercera generación.

En este capítulo se introducen los dos enfoques principales que son alternativas de solución a estas limitaciones. Se introducen las bases de datos de objetos complejos, previo a los sistemas de gestión de bases de datos orientadas a objetos. Posteriormente, se presentan los fundamentos del modelo relacional extendido.

El capítulo está organizado en dos grandes secciones. En la primera, se presentan los fundamentos del modelo relacional extendido. En esta sección se describe un álgebra relacional extendida propuesta por Date y Darwen [DD95]. Posteriormente, se introduce el modelo de datos orientado a objetos. El concepto de objeto complejo basado en [AHV95], [AB95] y [AK90], un álgebra de objetos complejos y el estándar de la ODMG para sistemas de gestión de bases de datos OO se tratan también en esta sección.

Modelo de datos objeto-relacional

A partir de cuatro tipos generales de sistemas de gestión de datos (Sistemas de Archivos, SGBD Relacionales, SGBD Objeto-Relacional y SGBD Objeto), en [SM96] se introduce el modelo objeto-relacional, considerado por el autor como la siguiente generación de sistemas de gestión de datos y actualmente el que tiene mayor participación en el mercado. De acuerdo con los autores, los SGBD objeto-relacional deben dar soporte (en un contexto SQL) a:

- Tipos de datos extensible,
- Objetos complejos,
- Herencia, y
- Sistema de reglas de producción.

Según Scholl [Scholl96], las diferentes opciones de extensión del mode-

lo relacional se basan en la extensión de los componentes de un modelo de datos: los tipos de datos primitivos, los tipos de constructores y los operadores aplicables a cada tipo primitivo y a cada tipo de constructor.

La estandarización de los sistemas objeto-relacional, de acuerdo con [LG02], se ha definido en el estándar ANSI/ISO SQL99 [SQL99]. El estándar incluye características como los tipos definidos por el usuario, métodos y colecciones, entre otras.

Por su parte, Date y Darwen, en el Tercer Manifiesto [DD95], caracterizan los sistemas de gestión de bases de datos del futuro, fundamentados en el modelo relacional. Proponen una serie de prescripciones, prohibiciones y recomendaciones para estos futuros sistemas que sirven como fundamento del modelo objeto-relacional.

Un representante de los sistemas de gestión de bases de datos relacional extendido, es POSTGRES [SR86] [Stonebraker86] [SAH87]. POSTGRES, sucesor del sistema de gestión de bases de datos relacional INGRES, ofrece, entre otros, soporte a objetos complejos, extensibilidad de tipos de datos, operadores y métodos de acceso.

Álgebra relacional extendida

En [DD95] se propone la siguiente álgebra, llamada A, para el modelo relacional extendido. Una clase se asocia con un dominio y una relación r se define en términos de una cabeza (*Hr*) y un cuerpo (*Br*). *Hr*, representa el esquema de la relación r y *Br* es el conjunto de tuplas de la relación.

Definición formal

Sea r una relación, a un atributo, T el tipo del atributo, a y v un valor de tipo T. Una cabecera *Hr* es un conjunto de pares ordenados $\langle a,T\rangle$ para cada atributo a de r. Dos pares $\langle a_1,T_1\rangle$ y $\langle a_2,T_2\rangle$ son nombres de dos atributos diferentes.

Sea *tr* una tupla y *Hr* una cabecera. La tupla *tr* es un conjunto de tripletas ordenadas $\langle a,T,v\rangle$, donde cada atributo a_i de *Hr* está asociado con una tripleta $\langle a_i,T_i,v\rangle$.

El cuerpo *Br* de una relación r es un conjunto de tuplas t. Pueden existir tuplas t_j en la cabeza *Hr* que no están en *Br*.

Observaciones

- Cada cabecera y cuerpo *Hr* y *Br*, respectivamente, se considera un conjunto.
- Un subconjunto de la cabecera *Hr* (respectivamente de cuerpo *Br*) es una cabecera *Hr´* (respectivamente de cuerpo *Br´*)

En el *álgebra-A* objeto-relacional propuesta en [DD95] los operadores

aplicados a una relación r, consideran acciones semánticas diferentes aplicados a *Hr* y a *Br*.

Operadores

Los siguientes cinco operadores básicos y dos derivados integran el *álgebra-A: AND, OR, NOT, RENAME, REMOVE.* Los operadores derivados son *COMPOSE y TCLOSE.* Los dos operadores derivados permiten, respectivamente, el cálculo de la composición de relaciones como una generalización de la composición de funciones y el del cierre transitivo. Los operadores básicos se definen como sigue:

Sean r y s dos relaciones siendo $r \equiv (Hr, Br)$ y $s \equiv (Hs, Bs)$ con *Hr*, *Hs* y *Br*, *Bs* siendo las cabeceras y cuerpos de r y s, respectivamente.

Operador AND

Este operador calcula la conjunción de dos relaciones r_1 y r_2. La cabecera *Hs* de la relación resultante s es la unión de las respectivas cabeceras Hr_1 y Hr_2, de las relaciones r_1 y r_2. El cuerpo *Bs* de la relación resultante s es la conjunción de algunas tuplas de Br_1 y Br_2.

$$s \leftarrow r_1 AND r_2 ; Hs = Hr_1 \cup Hr_2 ; \; Bs = \left\{ t_s \mid \exists t_{r_1}, t_{r_2} ((t_{r_1} \in Br_1) \wedge (t_{r_2} \in Br_2) \wedge (t_s = t_{r_1} \cup t_{r_2})) \right\}$$

Este operador corresponde al *join* natural del álgebra relacional.

Operador OR

El operador *OR* generaliza el operador *UNION* del álgebra relacional. La cabecera *Hs* de la relación resultante s, es la unión de las respectivas cabeceras Hr_1 y Hr_2, de las relaciones r_1 y r_2. El cuerpo *Bs* de la relación resultante s es la disyunción de tuplas de los cuerpos Br_1 y Br_2.

$$s \leftarrow r_1 OR r_2 ; \; Hs = Hr_1 \cup Hr_2 ; Bs = \left\{ t_s \mid \exists t_{r_1}, t_{r_2} ((t_{r_1} \in Br_1) \vee (t_{r_2} \in Br_2) \wedge (t_s = t_{r_1} \cup t_{r_2})) \right\}$$

Operador NOT

Este operador permite calcular el complemento de una relación r,*(Hr, Br)*. La cabecera *Hs* de la relación resultante s es igual a la cabecera *Hr* de r. El cuerpo *Bs* de s contiene todas las tuplas t_s, que no pertenecen al cuerpo *Br* de r.

$s \leftarrow NOT(r)$; $Hs=Hr$; $Bs= \{ t_s \mid \forall t_r (t_r \notin B_r) \wedge (t_s = t_r)) \}$ siendo t_r una tupla perteneciendo a *Br*.

Operador RENAME

Este operador permite renombrar un atributo nombrado a de r por otro nombre de atributo b de la relación resultante s sin cambiar el tipo T. La

cabecera *Hs* de s es igual a la cabecera *Hr* de r excepto que el par $\langle a,T\rangle$ se reemplaza por $\langle b,T\rangle$.

$$s \leftarrow RENAME(a,b)\,;\, Hs = \{Hr - \{\langle a,T\rangle\}\} \cup \{\langle b,T,v\rangle\}$$

$$Bs = \{t_s \mid \exists t_r, \exists v(\, t_r \in Br) \wedge (v \in T) \wedge (\langle a,T,v\rangle \in t_r) \wedge (t_s = \{t_r - \{\langle a,T,v\rangle\}\} \cup \{\langle b,T,v\rangle\}))\}$$

Operador REMOVE

Este operador produce una relación eliminando el atributo a de la relación r. La cabecera *Hs* de s es igual a *Hr* de r sin el par $\langle a,T\rangle$. El cuerpo *Bs* de s es un subconjunto de tuplas t_r de r.

$$s \leftarrow r \;\; REMOVE \;\; a \;\; , \text{ donde } \langle a,T\rangle \in Hr, Hs = Hr - \langle a,T\rangle$$

$$Bs = \{t_s \mid \exists t_r, \exists v((\; t_r \in B \;\;) \wedge (\; v \in T \;) \wedge (\; \langle a,T,v\rangle \in t_r \;) \wedge (\; t_s = t_r - \{\langle a,T,v\rangle\}\;)\;)\}$$

Otra álgebra para el modelo objeto-relacional se propone en [BAS04], en donde un tipo o dominio se especifica mediante el operador *Op* equivalente a una función que devuelve un valor perteneciente a un dominio o tipo. El álgebra propuesta, denotada por A*, está integrada por operadores basados en cálculo de primer orden y en álgebra extendida, que de acuerdo con los autores, incrementa el poder expresivo del modelo.

Aspectos prácticos del lenguaje de consulta en sistemas objeto-relacional (SQL3)

El modelo objeto-relacional (ORDBMS) ofrece al usuario la posibilidad de crear nuevos tipos de datos, funciones y operadores sobre estos nuevos tipos y objetos complejos (objetos construidos a partir de múltiples tipos de datos mediante constructores). Con base en estas nuevas características, el procesamiento de una consulta en los ORDBMS debe tener en cuenta estos cambios. El optimizador necesita métodos especiales para acceder de manera eficiente a los nuevos tipos.

El lenguaje de consulta SQL-2003 tiene tanto características relacionales como orientadas a objeto. Las primeras son las mismas de los lenguajes relacionales y las últimas se relacionan con nuevos tipos de datos y de predicados, de aspectos relacionados con semántica extendida, seguridad y características de bases de datos activas [EM99] [EMK+04].

El constructor de ADT (*abstract data type*) es uno de los más importantes en SQL3. De acuerdo con Cattell [Cattell94] los ADT y las tablas en SQL3 no están integrados de manera adecuada. Los parámetros y variables de rutinas en SQL3 pueden ser ADT. Sin embargo, sólo se puede utilizar como tipo de un atributo de una tabla en una base de datos.

Los tipos de datos nuevos que se incluyen a una versión SQL relacional son, entre otros, los BLOB (*Binary Large Object*), CLOB (*Character Large Object*), BOOLEAN (con posibilidad de usar directamente los valores *true, false o unknown*), BIGINT, MULTISET, XML, y los tipos compuestos *ARRAY* y *ROW.* Adicionalmente, se permiten tipos estructurados definidos por el usuario (UDT). Los tipos estructurados definidos por el usuario se pueden definir de manera que puedan tener uno o más atributos de cualquier tipo definido. Los atributos están encapsulados y sus valores sólo son accesibles mediante funciones como *get* y *set.* Adicionalmente, los tipos pueden participar en jerarquías de tipo.

Funciones que pueden retornar tablas, es decir, un *multiset de rows* se pueden definir, al igual que funciones definidas por el usuario. Generadores de secuencias también están disponibles en esta nueva versión de SQL.

Entre los nuevos predicados están *SIMILAR* y *DISTINCT*. Una de las extensiones más importantes es la posibilidad de expresar consultas recursivas.

Optimización de consultas objeto-relacional

Construir un optimizador es una tarea compleja y que toma mucho tiempo. En el caso de los sistemas objeto-relacional, la estimación del costo de ejecución de una consulta compleja es difícil sobre todo si se tiene en cuenta que los usuarios pueden definir tipos de datos, operadores, funciones y métodos.

Las propuestas de optimización para sistemas objeto-relacional se basan en reglas, y en la optimización de métodos de acceso y de las funciones y operadores definidos por el usuario [SR86], [Kabra99], [GD87], [Freytag87].

El optimizador de EXODUS [GD87], un sistema extensible de base de datos, se diseñó basado en reglas. El optimizador trabaja sobre árboles y produce planes de ejecución que también lo son. Los operadores son primitivas que ofrece el modelo de datos cuyas implementaciones específicas se denominan métodos. Los nodos del plan de ejecución consisten de métodos con sus argumentos. En este contexto el proceso de optimización se basa en reordenar el árbol de la consulta y seleccionar métodos [GD87]. La entrada al optimizador es un conjunto de operadores, un conjunto de métodos, reglas algebraicas para trasformar los árboles de consulta y reglas para asignar métodos a operadores.

MESH es una estructura de datos en la cual se mantiene información sobre todas las alternativas exploradas hasta el momento y OPEN contiene el conjunto de las siguientes transformaciones aplicables. Las reglas para transformar árboles de consulta y para seleccionar métodos se deben definir inicialmente en el archivo de descripción del modelo.

El algoritmo básico de optimización se muestra en la Figura 5.2.

```
while (OPEN is not empty)
        Select a transformation from OPEN
        Apply it to the correct node (s) in MESH
        Do method selection and cost analysis for the new nodes
        Add newly enable transformations to OPEN
```

Figura 5.2 Algoritmo de optimización de EXODUS

En POSTGRES, Stonebraker et al. [SR87] [Fong95], el optimizador debe tener en cuenta los métodos de acceso definidos por el usuario y es necesario dar soporte a procedimiento, a campos tipo POSTQUEL (objetos complejos y procedimientos), a *triggers* y reglas. Para mejorar la ejecución de consultas POSTGRES ofrece un catálogo para almacenar consultas compiladas que son más rápidas que las que se deben *parsear* y optimizar en tiempo de ejecución.

El algoritmo de optimización implementado en POSTGRES, implementado en LISP, se basa en el propuesto en [SAC+79]. El optimizador crea un árbol con nodos *scan* y *join* que representa el plan de la consulta. Varias opciones se tienen para implementar ambas operaciones.

Cuando el usuario define una nueva operación, es necesario especificar su negación, puesto que las expresiones de la consulta se convierten a Forma Normal Conjuntiva y las leyes de DeMorgan se aplican. Esta información se almacena en el catálogo del sistema, al cual el *parser* puede acceder. Una descripción detallada del optimizador se presenta en [SAC+79].

Modelo de datos orientado a objetos

Los modelos de valor complejo [AB95] [RKS88], basados en el paradigma orientado a objetos y de relaciones anidadas, surgen como alternativa para relajar la exigencia del modelo relacional de manejar relaciones en 1FN, ofreciendo formas de manipular datos estructurados jerárquicamente.

Desde un punto de vista de los sistemas de gestión de datos, un sistema orientado a objetos incluye los conceptos de objeto, identidad de objeto, objetos compuestos, métodos, encapsulación y herencia, entre otros.

Un objeto representa una entidad que tiene ciertas características. Los objetos que comparten las mismas características se definen y referencian como un grupo. En un SGBDOO, los objetos son persistentes. El sistema genera un identificador de objeto, OID, único por objeto que se usa para reverenciarlo. Un objeto tiene propiedades o características denominadas *atributos*. Un atributo puede ser simple o complejo dependiendo de su valor. Los atributos complejos pueden ser colecciones, referencias representando relaciones mediante *OID* o atributos derivados. Los objetos compues-

tos se crean con base en otros objetos y se denominan también objetos de estructura compleja.

En relación con los métodos almacenados en bd, la mayoría de los sistemas los almacenan por fuera de la base de datos.

Objetos complejos

Un objeto complejo es una relación en la cual las entradas no son atómicas y pueden a su vez ser relaciones [AHV95]. A partir de valores atómicos y aplicando constructores de tupla y conjunto, sin restricciones de orden o profundidad, se genera un valor complejo [AB95].

Al igual que en la definición del modelo relacional introducida en el Capítulo 2, se asume la existencia de conjuntos infinitos contables de nombres de relación $\{R_1, R_2, ...\}$, atributos $\{A_1, A_2, ...\}$ y constantes $D = \{d_1, d_2, ...\}$, ***relname***, ***att*** y ***dom***, respectivamente. Se asume, además, que los conjuntos son entre ellos disjuntos.

Definición 5.1

La sintaxis τ de un tipo complejo es la siguiente:

$\tau = D \mid \langle\ B_1 : \tau, ..., B_k : \tau\ \rangle \mid \{\tau\}$, donde $k \geq 0$ y $B_1, ..., B_k$ son atributos distintos.

Definición 5.2

La interpretación de τ, denotada por $\|\tau\|$, se define como:

1. $\|D\| = D$
2. $\|\{\tau\}\| = \{\ v_1, ..., v_j\} \mid j \geq 0, v_i \in \|\tau\|, i \in [1, j]\ \}$
3. $\|\langle B_1 : \tau_1, ...B_k : \tau_k\ \rangle\| = \{\langle B_1 : v_1, ..., B_k : v_k\rangle \mid v_j \in \|\tau_j\|, j \in [1,k]\}$

Un objeto complejo puede ser entonces una constante, un conjunto o una tupla.

Un esquema de base de datos es un par $(\Re, S)$ donde $\Re$ es un conjunto finito de nombres de relación y S es un tipo.

Una instancia $\Im$ de un esquema $(\Re, S)$ es una función tal que para cada $R \in \Re$, $\|\{S(R)\}\|$.

Una instancia $\Im$ de $(\{R_1, R_2, R_3\}, S)$ tomada de [AK90] se muestra en la Figura 5.3. En esta instancia $S(R_2) = \langle A : D, A_1 : D, A_2 : D\rangle$ y $S(R_1) = S(R_3) = \langle A : D, B : \{\langle A_1 : D, A_2 : D\rangle\}\rangle$.

Un álgebra para objetos complejos

Abiteboul y Kanellakis [AK90] introducen un álgebra para objetos complejos. Los dos constructores de tupla y de conjunto (finitos) de objetos complejos también se definen. De esta manera, los objetos complejos se pueden representar mediante árboles finitos cuyos nodos internos pueden ser alguno de estos constructores.

Asumiendo la sintaxis y la interpretación de un valor complejo, como la introducida en el ítem "Objetos complejos", se definen operadores de conjunto, operaciones de tupla y el producto cartesiano.

En [AHV95] se presenta el álgebra para objetos complejos ALG^{cv} que se basa en un conjunto de operaciones básicas definidas para tuplas y para conjuntos. Adicionalmente, se definen otras operaciones, constructivas y destructivas que conforman el núcleo (*core*) de ALG^{cv}.

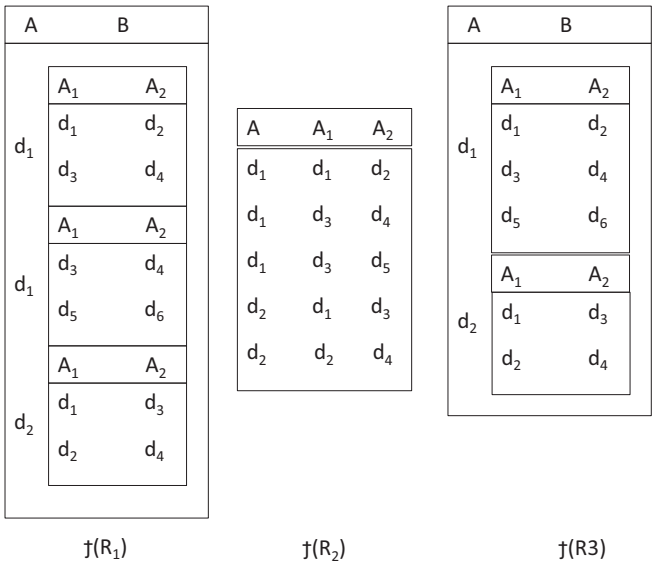

Figura 5.3 Instancia de base de datos

Álgebras objeto

Definir un álgebra objeto facilita la aplicación de transformaciones sintácticas como estrategia de optimización de expresiones de consulta. Para el caso del modelo datos orientado a objetos y debido fundamentalmente a la falta de un estándar, varias álgebras objeto han sido propuestas [CDLR89], [HS91], [LMSV+93], [SO90], [SZ89], [SZ89a]. Cada álgebra ha estado asociada con cada uno de los modelos de bases de datos orientadas a objetos.

Basado en el modelo de datos orientado a objetos ENCORE [ZM91], Shaw et al. [SZ89] presentan un álgebra objeto. Operadores como *Select, Image , Project, Ojoin* se definen. Adicionalmente, se definen también operaciones aplicables a conjuntos (*Union, Difference, Intersection*).

En el álgebra, propuesta en [SO90], se definen operadores y operandos que producen conjuntos de objetos, de manera que la propiedad de cierre se satisface. Cinco operadores integran esta álgebra: Unión, Diferencia, *Select, Generate* y *Map*, cuya definición y semántica se detalla en [SO90].

Subieta et al. [SKLY95] analizan el álgebra propuesta por Leung et al. [LMSV+93]. Varias críticas se sustentan centradas en aspectos matemáticos, conceptuales y prácticos. Teniendo en cuenta que un álgebra objeto debe ser el fundamento de un lenguaje de consulta, operaciones de los lenguajes como *update, create, insert o delete*, generan problemas a la hora de capturar su semántica. En este mismo estado están las jerarquías de objetos complejos, la herencia, los apuntadores en estructuras de datos, las consultas basadas en orden y los nombres utilizados para algunas consultas, entre otras.

Desde un punto de vista matemático, algunas de las críticas recaen en la falta de claridad y formalismo al definir el álgebra. Por ejemplo, proponer un álgebra requiere definir el conjunto sobre el cual se definen los operadores que conforman el álgebra. La propuesta de álgebra [LMSV+93] no define este conjunto matemáticamente. Problemas adicionales detallan en [SKLY95].

Una propuesta de solución a estas limitaciones se ofrece en [SKLY95], [SBM93]. La propuesta consiste en re-construir conceptos de lenguaje de consulta desde una perspectiva de un lenguaje de programación.

Base de datos orientada a objetos (OODB)

Los sistemas de bases de datos orientados a objetos deben soportar los tipos básicos de conjuntos y tuplas para poder construir y manipular objetos complejos. Algunas de las características más importantes de un sistema de gestión de bases de datos orientadas a objetos (SGBDOO) se introdujeron, inicialmente, en [Bancilhon88], cuando aún no se contaba con un modelo común ni con unos fundamentos formales para el modelo.

De acuerdo con [AHV95], los modelos de bases de datos orientados a objetos se construyen a partir de la conjunción de tres mundos: los objetos complejos, los modelos semánticos de bases de datos y conceptos de la programación orientada a objetos. En este sentido los modelos de bases de datos OO deben ofrecer el manejo de objetos y de identificadores de objetos, tipos y valores complejos, clases, métodos, jerarquías ISA, herencia y *binding* dinámico y encapsulación.

El Manifiesto-OO [ADMB+], una respuesta a la falta de un modelo de datos, describe las características que un sistema debe tener para que se pueda considerar un SGBDOO. Las características están agrupadas en tres clases: obligatorias, opcionales y abiertas.

Las características obligatorias tienen relación con dos aspectos fundamentales. El primero, ser un sistema de gestión de bases de datos se eviden-

cia en características como la persistencia, la gestión de almacenamiento secundario, la concurrencia, recuperación y consultas *ad-hoc*. La segunda, ser un sistema orientado a objetos se evidencia en características como los objetos complejos, la identidad de objeto, encapsulación, las clases o tipos, la herencia, *overriding y late binding*, la extensibilidad y mediante la existencia de un lenguaje de manipulación de datos. Cada una de estas características se detalla en [ADMB+].

Las características consideradas opcionales son la herencia múltiple, el chequeo e inferencia de tipos, la distribución a pesar de ser una característica independiente de los SGBDOO y el soporte a versiones, entre otras.

A pesar de su riqueza semántica y de su cercanía, en términos de modelamiento, al mundo real, aún no ha penetrado con fortaleza en el mercado. Lord y Gupta [LG02] explican este hecho basados en factores técnicos. Algunos de estos factores son: su complejidad, el mecanismo procedimental de navegación, sobre todo cuando se trata de consultas anidadas, y su falta de fundamentación matemática.

Definición formal de una base de datos OO

La definición formal del modelo de bases de datos OO se introduce en [AB95] y en [AHV95].

Sean $\hat{D}_1, \hat{D}_2, ...$ un conjunto de nombres de dominios, asociados con dominios $D_1, D_2, ...$ y un conjunto de nombres de atributos $A_1, A_2, ...$ Los valores atómicos son elementos de los dominios. Mediante constructores aplicados a estos valores atómicos se pueden construir valores complejos. Cada valor tiene asociado un tipo y este valor es una instancia de un tipo.

Definición 5.1
Tipos y valores se definen de la siguiente manera:

- Si $\hat{D}$ es un nombre de un dominio, entonces D es un tipo ***atómico***. Cada valor a de D es un valor de este tipo.
- Si $T_1, T_2, ..., T_n$ son tipos y $A_1, A_2, ..., A_n$ son atributos distintos, entonces $[T_1 : A_1, T_2 : A_2, ..., T_n : A_n]$ es un tipo ***tupla***. Si $v_1, v_2, ..., v_n$ son valores de los tipos $T_1, T_2, ..., T_n$ entonces $[A_1 : v_1, A_2 : v_2, ..., A_n : v_n]$ es un valor de ese tipo. La tupla vacía [] es el único valor del tipo $T_{[]}$.
- Si T es un tipo, entonces $\{T\}$ es de tipo ***conjunto***. Un conjunto finito de valores de tipo T es un valor de tipo $\{T\}$.

Definición 5.2
Un esquema de base de datos se define así:

$\mathbf{DB} = \langle [\hat{D}_1, ..., \hat{D}_k], [R_1 : T_1, ..., R_n : T_n] \rangle$, donde $T_1, T_2, ..., T_n$ son tipos *conjunto* involucrando únicamente los nombres de dominios $\hat{D}_1, ..., \hat{D}_k$.

Una instancia de **DB**, DB $= \langle [D_1,...,D_k],[R_1,...,R_n] \rangle$ donde D_i es un dominio y R_i es un valor de $DOM = (T_i, D_1,..., D_k)$.

Definición 5.3

Una consulta $\mathbf{DB} \rightarrow T$ es una función parcial de instancias de bases de datos **DB** a instancias de T. *q(DB),* es el resultado de aplicar una consulta *q* a una instancia de base de datos *DB*.

El estándar ODMG

El grupo ODMG (*Object Data Management Group*) [CBB+97] define una especificación para objetos persistentes. El estándar propuesto se basa en el paradigma orientado a objetos sin considerar un modelo de bases de datos existente. "El modelo ODMG soporta la noción de clase, de objetos con atributos y métodos y la de herencia con especialización" [BF94].

Las componentes del estándar ODMG son: el modelo objeto, los lenguajes de especificación de objetos, un lenguaje de consulta objeto (OQL) e implementaciones ODMG de los lenguajes C++, Smaltalk y Java, denominados lenguajes de manipulación de objetos.

El modelo objeto ODMG se define a partir de los siguientes elementos [Cattell94]:

- Objetos con *OID*
- Tipos
- Operaciones para definir la conducta de los objetos
- Valores de un conjunto de propiedades de los objetos que definen su estado. Las propiedades pueden ser atributos de los objetos o relaciones entre uno o más objetos.
- Nombres significativos que se pueden dar a los objetos. Los objetos pueden tener más de un nombre.
- Operaciones para los objetos de un tipo dado (*operation signatures*)
- Atributos para tipos objeto
- Relaciones en las que pueden participar los objetos de un tipo
- Tipos colección: *set, bag, list, array*
- Tipos objeto en un grafo subtipo/supertipo
- Un conjunto *extent*, que se puede usar para contener todas las instancias de un tipo dado
- Llaves o conjuntos de propiedades utilizadas para identificar, de forma única y mediante un valor, instancias de un tipo (llaves en el modelo relacional).

Un SGBDOO debe ofrecer como primitivas básicas *objetos***,** con un identificador único, y *literales* (sin identificadores). Cada una de estas primitivas tienen asociado un *tipo***.** Los objetos tienen estado. El estado se define mediante valores que toma un conjunto de *propiedades*. Las propiedades son

atributos del objeto o *relaciones*, entre el objeto y otros objetos. A un objeto se le asocia una conducta definida mediante *operaciones*.

La base de datos almacena objetos. Múltiples usuarios pueden acceder y compartir estos objetos. El esquema de la base de datos se define mediante un Lenguaje de Definición de Objetos (ODL). ODL facilita la definición de tipos objeto. La implementación de los tipos objeto se puede hacer mediante un lenguaje de programación. El estándar ODMG no ofrece un Lenguaje de Manipulación de Datos (DDL).

Los tipos tienen asociada una especificación externa (operaciones aplicables a sus instancias, propiedades y excepciones que puedan surgir) y una o más implementaciones (aspectos internos de los objetos del tipo). La especificación externa de un tipo corresponde a una descripción abstracta de las operaciones, excepciones y propiedades que pueden ver los usuarios del tipo. La implementación de un objeto consta de una representación y de un conjunto de métodos.

Un tipo se define en ODL por medio de su clase. La siguiente es parte de la gramática de ODL, tomada de [CBB+97]. Información detallada se puede encontrar en [CBB+97] o en [CBB+00].

⟨interface⟩ ::= ⟨interface_dcl ⟩
| ⟨forward_dcl⟩

⟨interface_dcl ⟩ ::= ⟨interface_header ⟩
{[⟨interface_body ⟩]}

⟨forward_dcl⟩ ::= ***interface*** ⟨identifier⟩

⟨interface_header⟩ ::= ***interface*** ⟨identifier⟩
[⟨inheritance_spec⟩]

⟨class⟩ ::= ⟨class_header⟩ {⟨interface_body⟩}

⟨class_header⟩ ::= ***class*** ⟨identifier⟩
[***extends*** ⟨scopedName⟩]
[⟨inheritance_spec⟩]
[⟨type_property_list⟩]

Por otra parte, el lenguaje de consulta OQL, basado en SQL-92, se puede utilizar independientemente para acceder a objetos, o embebido en un lenguaje de programación. OQL es un lenguaje tipado puesto que cada expresión en una consulta tiene un tipo. Es un lenguaje declarativo pero no completo computacionalmente.

Los siguientes son algunos ejemplos de consultas, tomadas de [CBB+97].

```
select distint x.age
from Persons x
where x.name=”Pat”
```

El resultado de esta consulta es el *conjunto* (literal) de edades de las personas con nombre “Pat”.

```
select distint struct(a:x.age, s:x.sex)
from Persons x
where x.name=”Pat”
```

En este caso para cada persona se construye un literal de tipo conjunto que contiene su edad y sexo.

Los objetos se pueden acceder por su nombre generalmente navegando a través de los datos. Un ejemplo [CBB+97] de una expresión de camino es *p.spouse.address.city.name*, en donde se trata de recuperar el nombre de la ciudad en donde vive la esposa de una persona p. Otros ejemplos más complejos, incluyendo predicados y *join,* se detallan en [CBB+97].

Optimización en SGBDOO

A pesar de que buena parte de los problemas de optimización abordados bajo el enfoque relacional se podrían considerar como la base para la optimización en SGBD objeto, sin embargo, características tales como los TDA, herencia de tipos y métodos, propias de este último, lo hacen difícil y complejo.

De acuerdo con [MZD91], un primer problema que deben enfrentar los optimizadores es la variedad cambiante de tipos. Deberían también poder aplicar transformaciones sobre constructores de tipos abstractos de datos y trabajar con expresiones de consultas anidadas. No parece posible, en general, para los autores, ofrecer transformaciones aplicables universalmente, como sí existen para el modelo relacional.

En [RS93] se describen procesos de optimización para consultas expresadas en COOL, un lenguaje de consulta de COCOON [SLR+]. La arquitectura del optimizador se detalla en [RS93].

Propuestas de optimización basadas en reglas de transformación de expresiones algebraicas de consulta se detallan en [BK90]. En [DS89] se describe el optimizador de IRIS, un SGBD orientado a objetos desarrollado por *Hewlett-Packard*, basado en dos tipos de reglas y en heurísticas. Unas reglas se usan para transformar la expresión de una consulta en otra más simple y otras para adicionar información sobre métodos de acceso y orden de aplicación de *joins*. De acuerdo con los autores, el diseño del optimizador basado en reglas facilita añadir nuevas reglas dependiendo de si se introducen nuevos tipos de datos u operadores. Las reglas de transformación de expresiones son de tres tipos: las que siempre son válidas, aquellas cuya

validez depende del esquema de bases de datos y las que son anotaciones al árbol representando el plan de ejecución de la consulta.

Debido a que las consultas OO introducen novedades, en relación con los sistemas relacionales tales como el soporte a métodos y a operadores definidos por el usuario, caminos, estructura compleja de objetos, mantenimiento de la herencia y polimorfismo y encapsulación, el optimizador requiere contar con nuevas técnicas de re-escritura, otras opciones de algoritmos y métodos de acceso y modelos de costos para los planes [GS05]. El proceso de optimización de consultas OO debe ser una adaptación del proceso de optimización relacional y por tanto estará integrado por tres etapas, como se muestra en la Figura 5.4: re-escritura, ordenamiento de operaciones y ejecución.

Los autores proponen reglas de re-escritura que se detallan en [GS05] y estrategias para tratar el ordenamiento de operaciones que es más complicada en consultas objeto debido a la estructura de la consulta, que generalmente es más compleja.

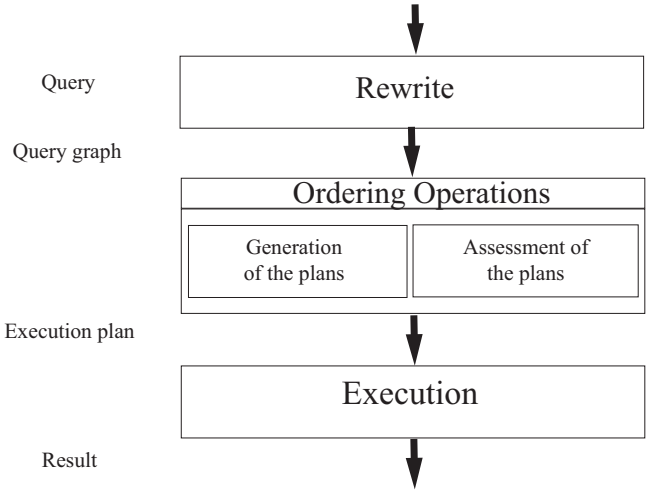

Figura 5.4 Proceso de optimización de consultas OO

El proceso de optimización de una consulta involucra los niveles lógico y físico. A nivel lógico se buscan expresiones equivalentes utilizando propiedades semánticas del lenguaje. A nivel físico se usa un modelo de costo que trabaja con información del sistema para poder escoger el mejor algoritmo para ejecutar la consulta. El trabajo en optimización relacional se ha tratado de adaptar al modelo objeto sobre todo en el nivel lógico [CD92].

Por otra parte, el proceso de optimización en el modelo objeto es diferente y exige el desarrollo de nuevas técnicas si se tiene en cuenta, por ejemplo, que los objetos referencian otros, que la estructura de un objeto es más compleja y que se pueden invocar métodos en una consulta.

El proceso de optimización de consultas OO propuesto por [CD92] integra varias técnicas. De acuerdo con Cluet y Delobel [CD92] el formalismo que se propone unifica distintas técnicas de reescritura, facilita el tratamiento de subconsultas duplicadas y de heurísticas para reducir la fase de reescritura.

El modelo de optimización que se propone tiene en cuenta información relacionada con enlaces inversos entre clases (*inverse links*), con los conjuntos conteniendo todas las instancias de una clase (*extents*), con los índices sobre caminos (*path indexes*) y con información sobre *clustering* de objetos en disco.

Referencias

[AB95] Abiteboul S., Beeri C. On the Power of Languages for the Manipulation of Complex Values. *VLDB Journal* (4)4:727-794,1995.

[ADMB+89] Atkinson M., DeWitt D., Maier D., Bancilhon F., Dittrich K., Zdonik S. The Object-Oriented Database System Manifesto. Proceeding First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan 1989. *Morgan Kaufmann Series In Data Management Systems Building an object-oriented database system: the story of 02,* pp.1-20.1989. Disponible en http://citeseer.ist.psu.edu/atkinson89objectoriented.html.

[AK90] Abiteboul S., Kanellakis P. Database Theory Column: Query Languages for Complex Object Databases. *ACM SIGACT News* (21)3: 9-18, 1990.

[Bancilhon88] Bancilhon F. Object-Oriented Database System. *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'88,* pp. 152-162, Austin, Texas, 1988.

[BAS04] Bahloul S. N., Amghar Y., Sayah M. A*: Álgebra for an Extended Object/Relational Model. *International Journal of Computer Science & Applications* (I)II:76-95, 2004.

[BF94] Bancilhon F., Ferran G. ODMG-93: The Object Database Standard. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (17):3-14, 1994.

[BOS91] Butterworth P., Otis A., Stein J. The Gemstone Object Database Management System. *Communication of the ACM* (34)10:64-77, 1991.

[CD86] Carey M.J., DeWitt D.J. The Architecture of the EXODUS extensible DBMS. *Proceedings of 1986 International Workshop on Object-Oriented Data-*

base Systems, Pacific Grove,California, USA, pp. 52-65, 1986. ACM SIGMOD Anthology, Vol. 2, Issue 3.

[CD92] Cluet S., Delobel C. A General Framework for the Optimization of Object-Oriented Queries. *ACM SIGMOD Record* (21)2:383-392 , 1992.

[CDGH+88] Carey M. J., DeWitt D. J., Graefe G., Haight D. M., Richardson J. E., Schuh D. T., Shekita J. E., Vandenberg S. The EXODUS extensible DBMS Project: An Overview. Disponible en: http://pages.cs.wisc.edu/~dewitt/includes/oodbms/exodus88.pdf.

[CDLR89] Cluet S., Delobel C., Lécluse C., Richard P. Reloop, an Algebra Based Query Language for an Object-Oriented Database System. *Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), Kyoto Research Park,* Kyoto, Japan, 1989, pp. 313-332.

[CH90] Carey M., Haas L. Extensible Database Management Systems. *SIGMOD Record* (19)4:54-60, 1990.

[CMCD94] Cheng J., Mattos N., Chamberlin D. G., DeMichiel L. Extending Relational Database technology for new applications. *IBM System Journal(33)2:264-279, 1994.* Disponible en http://www.research.ibm.com/journal/sj/332/ibmsj3302E.pdf.

[CDV88] Carey M, DeWitt D., Vandenberg S. A Data Model and Query Language for EXODUS. *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data,* pp. 413-423, Chicago, Illinois, 1988.

[CR86] Cattell R. G. G., Rogers T. R. Combining Object-oriented and Relational Model of Data. *Proceedings on the 1986 international workshop on Object-oriented database systems Pacific Grove,* pp. 212-213, California, United States, 1986.

[DD95] Darwen H., Date C.J. The Third Manifesto. *ACM SIGMOD Record* (24)1:39-49, 1995.

[Deux91] Deux O. et al. The O_2 System. *Communication of the ACM* (34)10:34-48, 1991.

[DS89] Derrett N, Shan M. Rule-Based Query Optimization in IRIS. *Proceedings of the 17th Conference on ACM Annual Computer Science Conference, CSC'89,* pp. 78-86, 1989.

[EM99] Eisenberg A., Melton J. SQL:1999, formerly known as SQL3. SIGMOD Record(28)1, 1999. Disponible en http://www.sigmod.org/record/issues/9903/.

[EMK+04] Eisenberg A., Melton J, Kulkarni K., Michels J., Zemke F. SQL: 2003 has been published. *SIGMOD Record* (33)1:119-126, 2004.

[FeMa2001] Fegaras L., Maier D. Optimizing Object Queries Using and Effective Calculus. *ACM Transaction on Database Systems* (25)4: 457-516, 2000.

[Fong97] Fong J. Converting relational to object-oriented databases. *ACM SIGMOD Record* (26)1: 1997.

[Fong95] Fong Z. The Design and Implementation of the POSTGRES Query Optimizer. M.S. Report. University of California, Berkeley. Disponible en http://db.cs.berkeley.edu/papers/UCB-MS-zfong.pdf.

[Freytag87] Freytag J. C. A Rule-Based View of Query Optimization. *ACM SIGMOD Record* (16)3:173-180, 1987.

[GGT96] Gardarin G., Gruser J. R.,Tang Z. H. Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Databases. *Proceedings of the 22nd VLDB Conference,* Bombay, India, pp. 390-401, 1996.

[GD87] Graefe G., DeWitt D. The EXODUS Optimizer Generator. Department of Computer Science, University of Wisconsin. Disponible en http://www.seas.upenn.edu/~zives/03s/cis650/P160.PDF.

[GGMR00] Grant J., Gryz J., Minker J. Raschid L. Logic-Based Query Optimization for Object Databases. *IEEE Transactions on Knowledge and Data Engineering* (12)4:529-547, 2000.

[Grigoriev07] Grigoriev E. Why the formal relational data model can be considered as a basis of object-oriented systems. Disponible en http://arxiv.org/vc/arxiv/papers/0708/0708.0361v1.pdf.

[GS05] Sassi M., Grissa-Touzi A. Contribution to the Query Optimization in the Object-Oriented Databases. Proceedings of World Academy of Science, Engineering and Technology (6): 267-270, 2005. Disponible en http://www.waset.org/pwaset/v6/v6-64.pdf.

[HS91] Heuer A., Scholl M.H. Principles of Object-Oriented Query Languages. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.9060.

[IBMDB2] DB2. Disponible en http://www-306.ibm.com/software/data/db2/9/

[InformixDS] Informix Dynamic Server, IBM. Disponible en http://www-306.ibm.com/software/data/informix/ids/.

[JS82] Jaeschke G., Schek H.J. Remarks on the Algebra of Non First Normal Form Relations. *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems,* pp. 124-138, Los Angeles, California, 1982.

[Kabra99] Kabra N. Query Optimization for Object-Relational Database Systems. PhD Thesis, University of Wisconsin, Madison 1999. Disponible en http://pages.cs.wisc.edu/~navin/research/thesis.pdf.

[Kent79] Kent William. Limitations of Record-Based Information Models. *ACM Transactions on Database Systems* (4)1: 107-131, 1979.

[LG02] Lord C., Gupta S. The Evolution of Object-Relational Databases. *Carnegie Mellon 45-872 Information Resources Management, 2002.* Disponible en http://www.chrisandtrudi.com/Chris/Portfolio/ObjectOriented%20Database%20Survey.pdf

[LLPS91] Lohman G. M., Lindsay B., Pirahesh H., Schiefer K. B. Extensions to Starburst: Objects, types, functions, and rules. *Communication of the ACM* (34)10:94-109, 1991.

[LMSV+93] Leung T. W., Mitchell G., Subramanian B., Vance B., Vandenberg S. L., Zdonik S. B. The AQUA Data Model and Algebra. *Technical Report CS-93-09, March 1993, Brown University.* Disponible en http://www.cs.brown.edu/research/pubs/techreports/reports/CS-93-09.html.

[McClure97] McClure S. Object Database vs. Object-Relational Databases. International Data Corporation *IDC Bulletin #14821E - August 1997.* Disponible en http://www.geog.ubc.ca/courses/geog516/notes/ordbms.htm.

[MZD91] Mitchell G., Zdonik S., Dayal U. Object-Oriented Query Optimization: What's the problem? *Technical Report CS91-41, Brown University.* Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.6094

[PBRV90] Premerlani W., Blaha M., Rumbaugh J.E., Varwig T. An Object-oriented Relational Database. *Communications of the ACM* (33)11: 99-109, 1990.

[Petrini07] Petrini J. Object-relational query processing. Disponible en http://user.it.uu.se/~torer/kurser/mdb/2007/TermPapers/JohanPetrini.pdf.

[RKS88] Roth M.A., Korth H.F., Silberschatz A. Extended Álgebra and Calculus for Nested Relational Databases. *ACM Transaction on Database Systems* (13)4: 389-417, 1988.

[RS93] Rich C., Scholl M. H. Query Optimization in an OODBMS. *Computer Science Department, University of Ulm.* Disponible en www.inf.uni-konstanz.de/dbis/publications/download/RS:BTW93.ps.gz

[SAC+79] Selinger P. G., Astrahan M. M., Chamberlin D. D., Lorie R. A., Price T.G. Access Path Selection in Relational Database Management System. *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data,* pp.:23-34 Boston, Massachusetts, 1979.

[SAH87] Stonebraker M., Anton J., Hanson E. Extending a Database Systems with Procedures, *ACM Transaction on Database Systems* (12)3:350-376, 1987. Disponible en http://citeseer.ist.psu.edu/11110.html.

[Scholl92] Scholl M. Extensions to the relational data model. In P. Loucopoulos and R. Zicari, editors, Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development. Jhon Wiley & Sons, New York, 1992. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.224

[SKLY95] Subieta K., Kambayashi Y., Leszczylowski J., Yokota K. A Critique of Object Algebras, 1995. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.9613.

[SKL95] Subieta K., Kambayashi Y., Leszczylowski J. Procedures in Object-Oriented Query Languages. *Proceedings of 21th International Conference on Very Large Databases,* pp. 182-193, 1995. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.2906.

[SKL95a] Subieta K., Kambayashi Y., Leszczylowski J. Towards an Integrated Query/Programming Language for Object Bases: a Broad Outlook. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.3888.

[SLR+] Scholl M.H., Laasch C., Rich C., Schek H.J., Tresch M. The COCOON Object Model. *Technical Report No. 192, Department of Computer Science, ETH* Zurich, 1992. Disponible en http://citeseer.ist.psu.edu/219515.html.

[SQL99] ANSI/ISO/IEC International Standard (IS). Database Language SQL-Part 2: Foundation (SQL/Foundation), September 1999. Disponible en http://www.ncb.ernet.in/education/modules/dbms/SQL99/ansi-iso-9075-2-1999.pdf.

[Stonebraker86] Stonebraker M. Inclusion of New Types in Relational Database Systems, *Proceedings of the Second International Conference on Data Engineering,* pp. 262-269, IEEE Computer Society 1986. Disponible en http://citeseer.ist.psu.edu/stonebraker86inclusion.html.

[SR86] Stonebraker M., Rowe L. The Design of POSTGRES**.** *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data,* pp. 340 - 355, Washington, D.C., United States ,1986. Disponible en http://citeseer.ist.psu.edu/stonebraker86design.html.

[SRLG+94] Stonebraker M., Rowe L., Lindsay B., Gray J., Carey M., Brodie M., Bernstein P., Beech D. Third-Generation Database System Manifesto. *Morgan Kaufmann Series In Data Management Systems Readings in database systems* (2nd ed.):932-945, 1994. Disponible en http://www.cl.cam.ac.uk/teaching/2003/DBaseThy/or-manifesto.pdf.

[SO90] Straube D. D., Ozsu M. T. Queries and Query Processing in Object-Oriented Database Systems. *ACM Transaction on Database Systems*(8)4: 387-430, 1990. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.717.

[SZ89] Shaw G. M., Zdonik S. B. A Query Algebra for Object-Oriented Databases. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7725.

[SZ89a] Shaw G.M., Zdonik S.B. Object-Oriented Queries: Equivalence and Optimization. *Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD'89), Kyoto Research Park,* Kyoto, Japan, 1989, pp. 281-295, 1989. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.5439.

[ZM91] Zdonik S. B., Mitchell G. ENCORE: An Object-Oriented Approach to Database Modelling and Querying. *Data Engineering* (14)2:53-57, 1991. Disponible en http://sites.computer.org/debull/91JUN-CD.pdf.

[Wang00] Wang Q. Cost-Based Object Query Optimization. Disponible en http://www.edbt2000.uni-konstanz.de/phd-workshop/papers/Wang.pdf.

Bibliografía básica anotada

[AAC+94] Annevelink J., Ahad R., Carlson A., Fishman D., Heytens M., Kent W. Object SQL- A Language for the Design and Implementation of Object Databases. *In Modern database systems: the object model, interoperability, and beyond:* 42-68, 1995, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. Disponible en http://www.hpl.hp.com/techreports/94/HPL-94-02.pdf.

Se describe OSQL, Object SQL, un lenguaje de bases de datos que combina características de SQL y lenguajes de programación OO. OSQL se desarrolló como parte del proyecto IRIS de los laboratorios de Hewlett-Packard. Algunas de las características que ofrece OSQL son, identidad de objetos, sistema de tipos con herencia múltiple, agregación de objetos (listas y conjuntos), entre otras.

El modelo de OSQL se apoya en los conceptos de objetos, tipos y funciones. Este lenguaje está disponible en HP OpenODB un sistema de gestión de bases de datos OO.

[ADMB+89] Atkinson M., DeWitt D., Maier D., Bancilhon F., Dittrich K., Zdonik S. The Object-Oriented Database System Manifesto. Proceeding First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan 1989. *Morgan Kaufmann Series In Data Management Systems Building an object-oriented database system: the story of 02,* pp. 1-20. 1989. Disponible en http://citeseer.ist.psu.edu/atkinson89objectoriented.html.

En el artículo se define un sistema de gestión de bases de datos OO. Tres tipos de características de los sistemas se proponen: Obligatorias, opcionales y abiertas.

Entre las características obligatorias están el ser un SGBD, lo que implica ofrecer persistencia, gestión de almacenamiento secundario, concurrencia, recuperación y consultas *ad-hoc*. Adicionalmente, se espera que sea orientado a objetos, lo que significa que ofrezca soporte a objetos complejos, identidad de objeto, encapsulación, herencia, clases, entre otras.

Las características opcionales, que mejoran el sistema son, entre otras, soporte a herencia múltiple, chequeo de tipos y gestión de transacciones. Las características abiertas les ofrecen a los diseñadores más opciones. Entre estas se proponen, los constructores adicionales y un sistema de tipos.

[AK90] Abiteboul S., Kanellakis P. Database Theory Column: Query Languages for Complex Object Databases. *ACM SIGACT News* (21)3: 9-18, 1990.

Las bases de datos de objetos complejos se proponen como una extensión de una base de datos relacional. En este artículo se presentan lenguajes para manipular estas bases de datos: álgebra y cálculo multi-tipo.

[Bancilhon88] Bancilhon F. Object-Oriented Database System. *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,PODS'88,* pp. 152-162, Austin, Texas, 1988.

El artículo presenta una revisión sobre el estado de la investigación en el tema de las bases de datos orientadas a objetos. Se caracteriza un SGBD orientado a objetos. Entre las características que el autor propone debe ofrecer un SGBDOO están la encapsulación, la identidad de objeto, los tipos y las clases, la herencia y *overriding and late binding.*

A pesar de que los SGBD relacional unidos con un lenguaje de programación de propósito general, de acuerdo con el autor, son capaces de hacer "...casi todas las

cosas… puesto que aseguran persistencia, son confiable, comparten datos, pueden modelar datos y ejecutar cualquier cálculo", aún existen aplicaciones difíciles de modelar con el relacional y que podrían ser lentas. En este sentido el autor compara los dos tipos de sistemas desde el punto de vista de la facilidad de programación y de la velocidad de cómputo.

Algunas de las ventajas de los SGBDOO sobre los SGBDR son el poder trabajar con objetos complejos, la identidad de objeto, la extendibilidad, la posibilidad de almacenar datos y programas bajo en mismo formalismo y sistema y la herencia, entre otros. Por otra parte, el cambio de modelo hace las cosas más difíciles puesto que la simplicidad del modelo relacional es una de sus ventajas. Adicionalmente, el modo de consulta *ad-hoc* sobre bases de datos relacionales cuentan con buenos lenguajes e interfaces de consulta. En los SGBDOO la manipulación de datos es más navegacional.

[BK90] Bancilhon F., Kim W. Object-Oriented Database Systems: In transition. *SIGMOD Record* (19)4:49-53, 1990.

En este artículo se discuten cinco tópicos que se consideraban relevantes en el desarrollo e investigación en sistemas de gestión de bases de datos OO: modelos y optimización de consulta, interfaces de usuario, metodologías de diseño, mecanismo de vistas y pruebas de desempeño.

Se proponen retos y líneas de trabajo investigativo en cada uno de estos tópicos.

Algunos de estos retos se relacionan son:

- Contar con lenguajes de consultas que accedan a datos encapsulados en los objetos mediante la invocación de métodos.
- Extender la propiedad de cierre de los lenguajes de consulta relacionales a los lenguajes OO
- Optimización de consultas OO, debido a que este proceso es mas complejo en SGBDOO puesto que los datos tienen estructuras son más complejas y una consulta incluye la invocación a métodos
- Desarrollo de metodologías OO específicamente para el diseño de bases de datos OO.

[Cattell91] Cattell R.G.G. (Guest Editor) What are next-Generation Database System. *Communication of the ACM* (34)10:31-33, 1991.

En este artículo se presentan los principales enfoques de los sistemas de gestión de bases de datos de tercera generación llamados orientados a objetos o relacionales extendidos. Las características que deben ofrecer estos nuevos modelos incluyen los objetos, OIDs, procedimientos encapsulados, objetos compuestos y relaciones entre objetos. Se espera que estas nuevas características se añadan a las características más importantes de los sistemas de gestión tradicionales. En relación con aspectos de modelamiento de datos, la nueva generación de sistemas de gestión debería combinar las mejores características de los sistemas de gestión de bases de datos relacionales y las de los lenguajes de programación orientados a objetos.

Desde este punto de vista los sistemas se agrupan en dos grandes categorías, dependiendo de su origen: lenguajes de programación o sistemas de bases de datos. Esto significa que algunos surgen como extensiones de los lenguajes de programación ofreciendo persistencia, concurrencia, control y consultas, entre otras características. Otros, por su parte extienden las funcionalidades de un RDBMS con características de la programación OO (i.e. referencias a objetos, procedimientos).

[Chaudhri93] Chaudhri A. Object Oriented Management System: An Overview. Disponible en http://citeseer.ist.psu.edu/158152.html.

Se discuten las limitaciones del modelo de datos relacional y relacional extendido (poder expresivo limitado, tipos de datos disponibles muy simples, problemas de impedancia debido a la combinación de lenguajes anfitriones y SQL, problemas de desempeño).

Los requerimientos de un sistema de gestión de bases de datos objeto se relacionan y discuten: persistencia, transacciones, concurrencia, recuperación, consulta y seguridad.

Con base en estos requerimientos se clasifican los modelos de sistemas de gestión de bases de datos orientados a objetos en las siguientes categorías:

- Basados en *Modelos de datos*, que incluyen propuestas de modelos anidados (no en 1FN), modelos funcionales y modelos semánticos.
- Basados en *Arquitecturas* se incluyen propuestas de gestores de objetos, sistemas de bases de datos extendidos, lenguajes de programación de bases de datos, generadores de sistemas de bases de datos, entre otros.
- Modelos de *servidores de almacenamiento*
- Modelos basados en *lenguajes para modelos de datos*.

[CDV88] Carey M, DeWitt D., Vandenberg S. A Data Model and Query Language for EXODUS. *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data,* pp. 413-423, Chicago, Illinois, 1988.

Se presenta el modelo de datos EXTRA y el lenguaje de consulta EXCESS para el sistema de bases de datos extensible EXODUS.

Hay consenso por parte de los investigadores en que los sistemas de gestión de bases de datos OO son el futuro, sin embargo no lo hay en lo que el sistema debería ser. Las propuestas varían desde los lenguajes de programación OO con persistencia hasta sistemas de bases de datos completos basados en modelos semánticos de datos. El primer enfoque, de acuerdo con los autores, es retroceder a los lenguajes de manipulación de datos navegacionales lo que haría difícil pensar en consultas ad-hoc y el la optimización de accesos.

Otra de las orientaciones es la de extender el modelo relacional para ofrecer soporte a objetos complejos, nuevos tipos de datos manteniendo características asociadas al lenguaje de manipulación de datos amigable al usuario y potente.

En el modelo de datos propuesto EXTRA la base de datos es una colección de objetos nombrados y persistentes que pueden ser simples o complejos. En este

modelo se separa la declaración de los tipos de sus instancias. EXTRA cuenta con tipos básicos y constructores para definir tipos esquema. Además de los tipos básicos enteros, punto flotante, booleanos y cadenas también da soporte a ADTs. Los constructores de tipos son tupla, conjunto y arreglos de longitud fija y variable y referencias.

Los objetos complejos, objetos compuestos de otros objetos, cada uno de los cuales a su vez puede estar compuesto de otros, se pueden construir mediante cuatros tipos de constructores de objetos, adicionales al constructor tipo tupla: *ref, own, set y array.*

EXCESS, el lenguaje de consulta, permite formular consultas sobre conjuntos de objetos, conjuntos de referencias y conjuntos de valores propios (*own*) mediante una sintaxis uniforme. Las consultas especifican el dominio en el cual las variables toman valores mediante la expresión *range of* $\langle \text{var}iable \rangle$ variable *is* $\langle range\_specification \rangle$. Se presentan detalles y ejemplos del lenguaje de consulta.

[GGT96] Gardarin G., Gruser JR.,Tang ZH. Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Databases. *Proceedings of the 22nd VLDB Conference,* Bombay, India, pp. 390-401, 1996.

En lenguajes de consulta objeto, las expresiones de camino representan navegación a través de los objetos, puesto que un objeto puede referenciar otros objetos mediante apuntadores. La navegación se puede restringir utilizando predicados y los objetos seleccionados en una consulta se pueden representar mediante variables. Estas expresiones se denominan expresiones de camino cualificadas.

En este artículo se evalúan estrategias en las cuales se combinan operaciones de *join* tradicionales con navegación basada en apuntadores. Se propone para esto un modelo para estimar los costos de los algoritmos de recorrido de colecciones.

Para evaluar una expresión de camino se analizan tres algoritmos: *Depth-First-Fetch (DFF), Breadth- First-Fetch (BFF) y Reverse- Breadth- First-Fetch (RBFF).*

[GGMR00] Grant J., Gryz J., Minker J. Raschid L. Logic-Based Query Optimization for Object Databases. *IEEE Transactions on Knowledge and Data Engineering* (12)4:529-547, 2000.

Técnicas de optimización semántica se aplican en el proceso de optimización de consultas objeto. Usando restricciones de integridad, las consultas se transforman en otra consulta equivalente pero más eficiente. Información sobre jerarquías de clase, relaciones entre los objetos, entre otros, se expresa como restricciones de integridad.

Una consulta en OQL (no anidada) se expresa como una consulta lógica (*Datalog*), que se optimiza. Los constructores (*e.g. set, array, list*) no se tienen en cuenta en el proceso de optimización.

A partir del esquema de la base de datos, especificada en ODL, en una primera fase de pre-optimización, éste se transforma en expresiones lógicas. Se asume que las restricciones de integridad están expresadas en términos de la lógica.

En la fase de optimización, se construye una expresión lógica para la consulta OQL. A partir de aquí se obtienen las consultas equivalentes a la original, que se evalúan convencionalmente mediante métodos basados en costo para encontrar la consulta optimizada OQL. Se aplican heurísticas para guiar el proceso de generación de consultas equivalentes.

[Kabra99] Kabra N. Query Optimization for Object-Relational Database Systems. PhD Thesis, University of Wisconsin, Madison 1999. Disponible en http://pages.cs.wisc.edu/~navin/research/thesis.pdf.

En el caso de los SGBD objeto-relacional (SGBDOR), el problema de optimización es una tarea muy difícil y que toma mucho tiempo, puesto que los usuarios pueden definir funciones, tipos de datos, operadores y métodos. Son problemas abiertos de investigación la forma de recoger estadísticas y la estimación de la selectividad de los predicados cuando se trata con funciones definidas por el usuario. Por lo tanto, es difícil estimar el costo de ejecución de una consulta en SGBDOR.

En esta tesis el autor ofrece una solución a estos problemas mediante una arquitectura de optimización de consultas extensible que denomina OPT++. OPT++ es una herramienta para construir optimizadores de consulta, que, de acuerdo con el autor, es flexible, extensible y fácil de usar.

Se describe también el algoritmo *Dynamic Re-Optimization*, para detectar planes sub-óptimos en tiempo de ejecución, mejorando el desempeño y re-optimizando la consulta dinámicamente. El algoritmo asume que se cuenta con un plan de ejecución anotado, con anotaciones de tamaño de relaciones intermedias y costo de ejecución de cada operador en la consulta. Durante la ejecución de la consulta y en ciertos puntos específicos se toman estadísticas que se usan para mejorar las estimaciones iniciales. Las estimaciones nuevas se comparan con las iniciales para detectar planes sub-óptimos. Cuando las nuevas estimaciones mejoran lo que falta por ejecutar del plan de ejecución, éste se cambia. El algoritmo se evalúa en el sistema *Paradise*.

[Kent79] Kent William. Limitations of Record-Based Information Models. *ACM Transactions on Database Systems* (4)1: 107-131, 1979.

Entre las limitaciones del modelo relacional referenciadas por Kent [Kent79] están, entre otras, la homogeneidad de los registros, dado que en el mundo real, a pesar de que grupos de individuos pueden formar un "tipo" de cosas, existen variaciones relevantes entre los individuos de ese conjunto. Otra limitación se refiere a que la estructura de registros no siempre corresponde a las características que intuitivamente cualquiera asocia con una entidad, sobre todo si se tiene en cuenta la normalización. Los sistemas normalizados reducen el número de opciones para tratar relaciones uno-a-muchos y muchos-a-muchos. La descripción de los registros no ofrecen evidencia clara ni explicita no implícitamente de la existencia de relaciones. No es fácil distinguir atributos de relaciones.

[McClure97] McClure S. Object Database vs.Object-Relational Databases. In-

ternational Data Corporation *IDC* Bulletin #14821E - 1997. Disponible en http://www.geog.ubc.ca/courses/geog516/notes/ordbms.htm.

Se analizan y comparan los sistemas de gestión de bases de datos OO y relacionales. Los dos modelos se comparan con respecto al modelo de datos subyacente, el lenguaje de consulta y el modelo computacional que utiliza. Adicionalmente se analizan los sistemas denominados post-relacionales o relacionales extendidos. Finalmente, se comparan los tres sistemas utilizando como criterios el estándar referente, soporte a programación orientada a objetos, simplicidad de uso, simplicidad para hacer desarrollos, extensibilidad, relaciones de datos complejos, desempeño y madurez del producto, entre otros.

[PBRV90] Premerlani W., Blaha M., Rumbaugh J.E., Varwig T. An Object-oriented Relational Database. *Communications of the ACM* (33)11: 99-109, 1990.

Se presenta una técnica para construir un sistema de gestión de bases de datos orientado a objetos a partir de un sistema de gestión de base de datos relacional y un lenguaje de programación orientado a objetos (LPOO). Un SGBDOO se define como la intersección de bases de datos y tecnología de programación orientadas a objetos.

La posibilidades son entonces extender un SGBDR con nuevos tipos de datos, operadores y métodos de acceso o extender los LPOO con funcionalidades de bases de datos como la persistencia, autorización y concurrencia y tendría que ser confiable administrando grandes volúmenes de datos.

La propuesta de los autores coincide con la segunda opción, extender el LPOO. El mecanismo para gestionar datos usa y bloquea una porción de la base de datos que copia en RAM. Sobre esta copia se llevan a cabo todas las tareas de lectura y escritura. Cuando se termina la tarea los datos se llevan a la base de datos para hacerlos disponibles. El SGBDR es un recurso de bajo nivel oculto al usuario final.

Las clases definidas en el LPOO se convierten en tablas y las instancias de objeto en filas de la tabla. Para cada objeto se genera un identificador ID que sirve de llave primaria de cada objeto. Algunas relaciones se convierten en tablas y otras se almacenan como tablas objeto con apuntadores.

[Stonebraker86] Stonebraker M. Inclusion of New Types in Relational Database Systems, *Proceedings of the Second International Conference on Data Engineering,* pp. 262-269, IEEE Computer Society 1986. Disponible en http://citeseer.ist.psu.edu/stonebraker86inclusion.html.

En este artículo se propone un mecanismo para construir tipos de datos definidos por el usuario. La especificación de un tipo de dato nuevo incluye un registro de la existencia del nuevo tipo, la longitud de su representación interna y rutinas de conversión de entrada y salida, puesto que los nuevos valores son la entrada a un programa o una salida para el usuario. Se detalla también la sintaxis para definir nuevos tipos.

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

Capítulo 6

DATA MINING y *DATA WAREHOUSE*: una introducción

Los últimos avances tecnológicos han hecho que las capacidades para generar y almacenar datos se incrementen día a día. Entre los factores que contribuyen a este hecho están el uso extendido de los códigos de barras, la automatización de todo tipo de transacciones (comerciales, negocios, científicas o gubernamentales) y los avances en la recogida de datos, entre otros. Además, el uso de la Internet ha inundado las bases de datos de las organizaciones.

Por otra parte, la evolución de los dispositivos de almacenamiento masivo en relación con su capacidad de almacenamiento versus su costo, ha hecho posible a las organizaciones almacenar *gigabytes* de información a un precio reducido. Todo tipo de información se almacena, desde los datos de los clientes, sus transacciones hasta datos de telemetría, monitorización de pacientes y evolución de los precios en los mercados, entre otros.

Todo este explosivo crecimiento de datos generó, en la década de los ochenta, la aparición del nuevo campo de investigación KDD (*Knowledge Discovery in Databases)* representando el proceso de descubrimiento de conocimiento en grandes volúmenes de datos [Fayyad96]. El proceso KDD ha servido para unir investigadores de áreas como la Inteligencia Artificial, la Estadística, Técnicas de Visualización, el Aprendizaje Automático y las Bases de Datos en la búsqueda de técnicas eficientes y eficaces para encontrar el potencial conocimiento que se encuentra inmerso en los grandes volúmenes de datos almacenados por las organizaciones diariamente. Técnicas y herramientas para analizar estos datos y descubrir patrones ocultos en ellos se desarrollan paralelamente bajo el nombre de *data mining*.

El gran aumento de datos que requieren analizar las organizaciones dio lugar no sólo a la aparición del proceso de KDD sino que al mismo tiempo y

de manera paralela generó el concepto de *data warehouse,* como repositorio integrado y no volátil de datos para dar soporte a los procesos de toma de decisión en la organización.

Las áreas de *data mining* y *data warehousing* surgen entonces con el propósito de ofrecer soluciones diferentes a las tradicionales para gestión de datos con evidentes limitaciones de soporte a las decisiones. El desarrollo de nuevos sistemas de gestión de datos para soporte analítico extiende las capacidades de los SGBD tradicionales para tratar operaciones con millones de registros.

Para los investigadores en bases de datos estas dos nuevas áreas de investigación se han convertido en un gran reto al tener que buscar una nueva manera de pensar, diseñar e implementar bases de datos, lo que las ha convertido en una de las áreas más activas en la comunidad de bases de datos.

La aparición de la investigación en *data mining* surge de la confluencia de dos elementos: La necesidad, por un lado, y los medios necesarios, por el otro. La necesidad viene marcada fundamentalmente por un cambio en los entornos organizativos que han provocado contextos muy competitivos y la necesidad de las organizaciones de sobrevivir en tales entornos.

En este capítulo se introducen conceptos generales relacionados con los temas de *data warehousing* y *data mining*.

Data warehousing

Un *data warehouse* (*dwh*) es un repositorio centralizado de datos que sirve de soporte en la gestión y toma de decisiones, y que se caracteriza por ser orientado a temas, integrado, no volátil y variable en el tiempo [Inmon96]. El *dwh* almacena información, de múltiples fuentes, que se ha transformado para ofrecer un modelo común, eficiente y multidimensional para apoyar consulta y análisis. Su arquitectura se muestra en la Figura 6.1, tomada de http://www.datawarehouse4u.info/index_en.html.

Los datos en el *dwh* no se organizan de acuerdo con las aplicaciones que los usan, sino de acuerdo con su semántica (**por temas**), independientemente de qué aplicación los utilice. Por ejemplo, una compañía podría tener datos organizados por clientes, proveedores, productos, etc., independientemente de la aplicación que los vaya a utilizar.

Otra de las características de la definición anterior es, a juicio del propio autor, la más importante de un *dwh*, la **integración**. Un *dwh* se construye a partir de los datos de las diversas fuentes de datos de una organización, lo que hace necesario un esfuerzo para "poner en común" los datos de las diferentes fuentes.

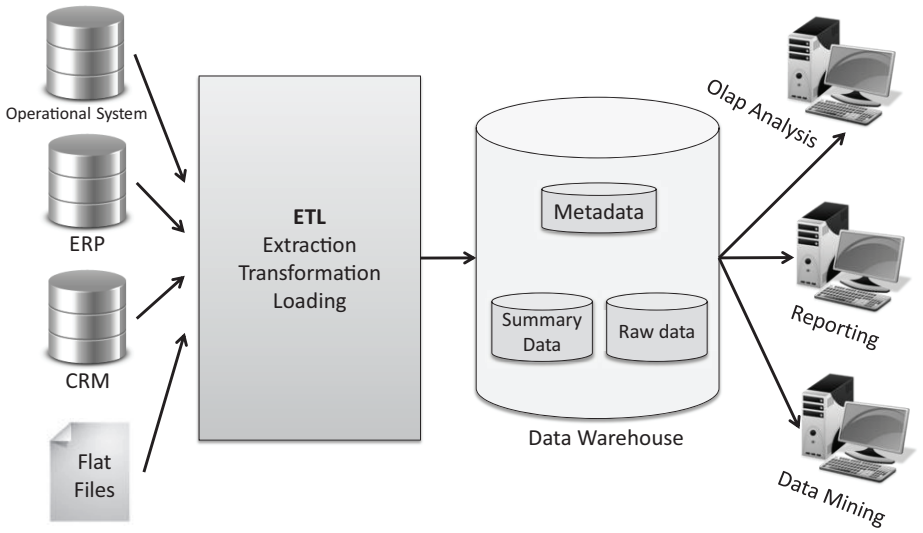

Figura 6.1 Arquitectura de un dwh

Cada una de las fuentes de datos de la organización tiene, por ejemplo, sus propios modelos de datos, sus propias políticas de asignación de nombres a campos y de codificación de valores. Por esta razón la integración de los datos en un esquema común implica un gran esfuerzo, tanto computacional como humano. El esfuerzo computacional proviene del hecho de que hay que recorrer todos los datos que se van a integrar, y transformarlos para que encajen con el esquema centralizado que se adopte para el *dwh*. El esfuerzo humano se requiere, por la necesidad de estudiar los modelos conceptuales, construir uno común, unificar todas las políticas de asignación y porque además esta tarea no se puede automatizar.

Otra característica importante es la **no volatilidad**. Existen varias razones por las que los datos de un *dwh* no son volátiles. Algunas de las más importantes son:

- Un *dwh* se construye para dar soporte a la toma de decisiones y este tipo de tareas puede requerir el análisis de datos en diferentes momentos del tiempo, para realizar análisis comparativos.
- Mantener diferentes versiones temporales de los datos permite recuperar el estado de los datos de la organización en cualquier instante, de modo que se pueden eliminar efectos indeseados de procesamientos erróneos.

Por tanto, los datos de un *dwh* no sufren actualizaciones. En él se mantienen diferentes versiones temporales de dichos datos y, por tal motivo, se requiere hacer inserción de los nuevos datos, a los que se añade una marca temporal que los distingue de las diferentes versiones ya existentes.

En relación con el tiempo, este es un factor diferenciador de los datos del *dwh*, es decir, que los datos en un *dwh* tienen un horizonte temporal de años, se pueden ver como una serie de "*snapshots*" o instantáneas tomadas en un momento en el tiempo, que no sufren actualizaciones. Adicionalmente, la fecha siempre forma parte de la llave de los datos en el *dwh*, para distinguir las diferentes versiones de los datos.

Finalmente, el *dwh* sirve de apoyo a la toma de decisiones. Esto hace referencia a que es necesario un modelo de almacenamiento de información que satisfaga las necesidades de las aplicaciones de análisis.

Esta definición hace énfasis en la naturaleza de los datos del *dwh*. Existen otras definiciones en las cuales se hace en mayor énfasis en los propósitos que se plantean con la construcción de un *dwh*. Por ejemplo, un *dwh* está organizado para servir de zona neutral de almacenamiento de datos, es la fuente de aplicaciones, especialmente de *data mining*, satisface una serie de requisitos específicos de la organización y usa datos que cumplen una serie de criterios predefinidos de la organización.

La primera de las características hace mención al hecho de que el *dwh* sirve de repositorio para información producida por otras aplicaciones, de ahí su "neutralidad". La segunda, hace referencia a la razón por la cual se construye el *dwh*. Las herramientas de consulta amigables juegan un importante papel para el proceso analítico en línea (*OLAP*, *On Line Analytical Processing*), y para los sistemas de soporte a la toma de decisiones (DSS, *Decision Support Systems*), también llamadas aplicaciones de *data mining*. El *data warehouse* no surge como soporte genérico de datos, sino específicamente para dar soporte de almacenamiento a estos procesos.

Una tercera definición tiene en cuenta la procedencia de los datos del *dwh* haciendo mas énfasis en ello: "un repositorio de información integrada proveniente de fuentes distribuidas, autónomas y posiblemente heterogéneas".

Diseño y construcción de un data warehouse

Un *dwh* recoge datos de diferentes fuentes, en un proceso denominado *adquisición*, los almacena en una base de datos relacional y posteriormente los ofrece a los usuarios en la fase de acceso a los datos, como se muestra en la Figura 6.2.

Los tres componentes de adquisición, almacenamiento y acceso, coinciden con los tres bloques principales de la figura.

El ***componente de adquisición*** sirve de interfaz con los sistemas operacionales, y tiene como función recoger los datos y hacerlos disponibles para el sistema de almacenamiento. Desde un punto de vista global, esta tarea consistente en identificar los datos que se quieren cargar en el *dwh* y cargarlos, parece simple. Sin embargo, no es tan sencillo, ya que surgen múltiples problemas cuando se trata de integrar datos de fuentes variadas.

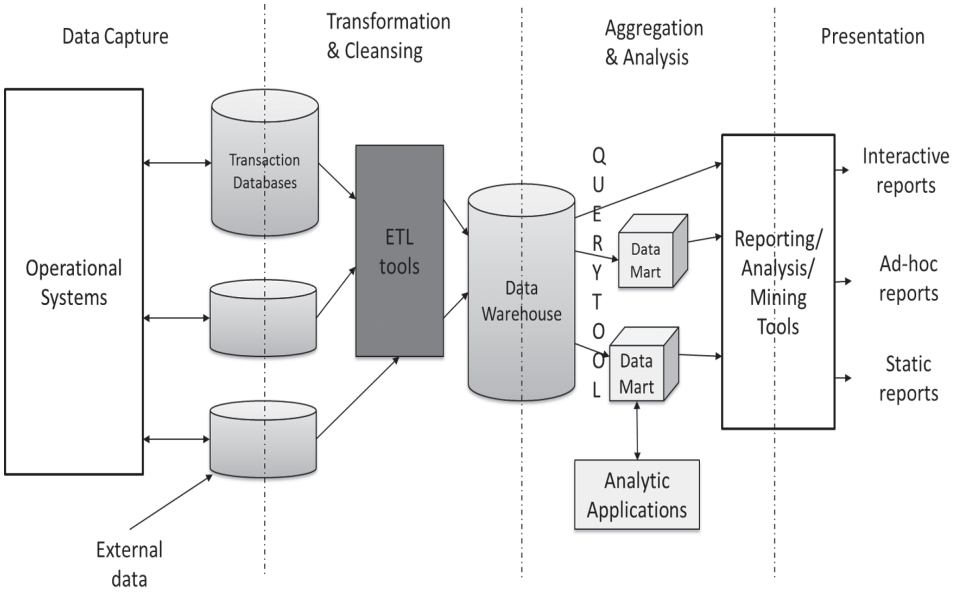

Figura 6.2 Componentes de un dwh

Tomada de http://www.orsoc.org.uk/about/topic/projects/elwood/images/Components.gif

Desde el momento en que se determinan cuáles datos formarán parte del *dwh* empieza la tarea de integración. Una vez se ha determinado cuáles datos se incluirán, se procede a buscarlos en las fuentes de datos de la integración.

El primer problema que se debe enfrentar es la heterogeneidad de las fuentes, que hace más difícil la tarea de encontrar los datos. Esta dificultad se debe a que un mismo dato, en distintas bases de datos, puede tener diferente nombre, diferente tipo de almacenamiento, diferente asignación de valores y hasta diferente representación interna, si los datos están almacenados en distintos gestores.

El problema de asignación de nombres hace referencia a que en las diversas fuentes de datos, las políticas de asignación de nombres pueden ser diferentes. Esto produce que tablas, atributos y demás elementos de una base de datos no se puedan identificar de manera unívoca por su nombre. No se puede asumir que dos elementos de diferentes fuentes sean iguales por tener el mismo nombre, ni que son diferentes sólo por tener nombres distintos. Será necesario consultar las descripciones de los datos para comprobar qué datos son los buscados.

El tipo de dato tampoco ayuda, necesariamente, en la tarea de encontrar atributos iguales. Dos datos equivalentes no tienen por qué estar almacenados bajo el mismo tipo de representación. Adicionalmente, la misma información puede estar representada con diferentes valores de atributos, sean

o no del mismo tipo. Por tanto, automatizar la búsqueda de los datos en la base de datos de la organización no es una tarea sencilla.

Después de la integración los primeros datos en el *dwh* están disponibles para su uso.

Las funciones del componente de adquisición son, por tanto, las siguientes:

- **Extracción de datos**. Esta función contempla la recolección de datos de las fuentes seleccionadas, así como la planificación de futuras extracciones que se realizarán una y otra vez durante la vida del *dwh* para "refrescar" su contenido.
- **Limpieza de datos**. Existen múltiples causas por las que los datos pueden ser erróneos: pueden ser inexactos, o hacer referencia a datos inexistentes o tener valores fuera de rango. La limpieza de datos es una tarea ardua, que no se puede realizar de manera completa, ya que la cantidad de datos hace que no sea eficiente la comprobación de todos y cada uno de los valores. En cambio, automáticamente sí se puede evitar que ciertos errores lleguen al *dwh*. Por ejemplo, se puede comprobar de forma automática si los datos están fuera de rango, o si no contienen valor alguno, y definir políticas para tratar ambos casos.
- **Formato de los datos**. Una vez que los datos están limpios, es necesario adaptarlos al formato con el cual se almacenarán en el *dwh*, ya que éstos pueden diferir de los originales.
- **Procesamiento de la mezcla**. Si los datos provienen de una única fuente, no es necesaria esta función. Sin embargo, en muchos casos, los datos provienen de fuentes diversas, lo que hace necesario tratar las disparidades introducidas por cada fuente.
- **Tratamiento de llaves**. Una de las necesidades básicas de toda base de datos es la identificación de los datos por una llave. Esto no es diferente en un *dwh*, así que se deben tener llaves para todos los datos. El problema es que uno de los puntos en que más suelen diferir los diferentes gestores es en el método de identificación de sus datos, por lo que será necesario definir llaves que puedan ser traducidas a partir de todas las llaves origen. Este proceso puede parecer una particularización del anterior, pero por su importancia y complejidad suele requerir un tratamiento aparte.
- **Proceso de purga**. Cuando no se quiere almacenar todas las ocurrencias de datos en el *dwh*, y que aquellos datos que cumplan ciertas condiciones no estén presentes en éste, es necesario "filtrar" este tipo de información.
- **Carga de datos.** Por último, una vez que los datos han pasado por todo el proceso de adecuación, sólo resta almacenarlos en el *dwh*.

Adicionalmente, este componente monitorea los cambios que se produ-

cen en los datos fuente, para poder integrarlos al *dwh*, como se muestra en la Figura 6.3.

Cada vez que se requiera una carga de nuevos datos, se aplica sobre éstos el mismo proceso que se realizó la primera vez. Por esta razón se debe almacenar información necesaria para repetir el proceso de manera automática.

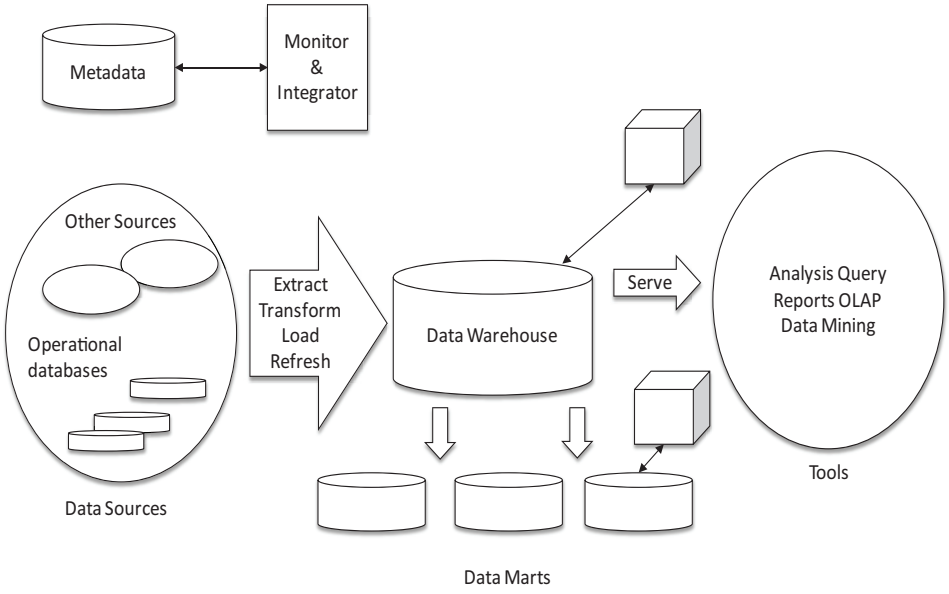

Figura 6.3 Componentes de un dwh

Tomada de http://www.codata.org/archives/2000/senegal-workshop/images/faye003.gif

El ***componente de almacenamiento*** gestiona el *dwh*. Cuando el *dwh* es una base de datos relacional, hay que tener en cuenta una serie de características especiales, como las siguientes:

- Existe un gran número de tablas, proveniente de las diferentes fuentes de datos de la organización.
- Las tablas son extremadamente grandes, ya que albergan datos provenientes de toda la organización.
- Hay un alto nivel de interdependencia.
- Los métodos de acceso no son predefinidos, ya que cada tipo de usuario hace peticiones bien diferenciadas.
- El acceso es en modo sólo lectura para usuarios, ya que el *dwh* se construye con propósitos de análisis. Los cambios se siguen produciendo en las fuentes originales de datos.
- Los datos se refrescan periódicamente de múltiples fuentes. Debido a que las fuentes de las que provienen los datos del *dwh* sufren cambios, sería necesario incluirlos en éste para que estén disponibles para las tareas de análisis.

- Existe un alto porcentaje de los datos históricos. Esto hace que, después de cierto tiempo de funcionamiento del sistema, existan numerosas versiones y que, por tanto, la mayoría de los datos sean históricos.

A medida que se construye el *dwh* hay que tener en cuenta que las anteriores características se pueden agrupar en tres categorías: alto volumen de datos y accesos no predefinidos, complejidad del entorno y manejo de la variable tiempo.

La primera característica de **alto volumen de datos y accesos no predefinidos** es "tóxica" para el desempeño de cualquier sistema. Surgirá, por tanto, un problema de desempeño/flexibilidad, que se podrá abordar intentando anticiparse al peor caso posible, lo cual puede tener excesivos requerimientos, evitando que los usuarios tengan total libertad de acción y estableciendo monitorización para establecer tiempos y cargas, precalculando parte de las consultas, lo cual puede hacer que se necesite mucho espacio adicional y estableciendo patrones fijos de consulta, a costa de perder flexibilidad en el sistema.

La segunda característica de **complejidad del entorno** es una de las trampas de la construcción, debido a que muchos desarrolladores no se dan cuenta de la complejidad que se puede llegar a tener debido principalmente al alto número de tablas. A medida que crece el número de tablas es más complicado saber qué contiene cada una, por lo que será necesario un catálogo de tablas más sofisticado que una simple lista de contenidos. Este nuevo catálogo se debe organizar de tal modo que los nuevos usuarios puedan saber qué contiene.

Por otra parte, además de saber qué contienen las tablas, será necesario conocer las relaciones entre ellas. Como consecuencia directa de esto, el mencionado catálogo se complica. Esto ha llevado a que la mayoría de productos se centren en ofrecer capacidades para gestionar catálogos de una manera más sofisticada.

Finalmente, la variable **tiempo** obliga a mantener cientos de tablas cuyos datos se deben refrescar en momentos diferentes, aspecto crucial para las consultas. Esto hace necesario contar con monitores de tiempo y de sincronismo.

El ***componente de acceso*** le da valor a los datos almacenados en el *dwh*. Se espera que pueda interpretar las peticiones de los usuarios mediante una interfaz sencilla, clara y potente, que les permita usar los datos de manera efectiva. Esto significa hacer peticiones al subsistema de almacenamiento e implica que:

- El sistema identifique el tipo de usuario para controlar el conjunto de acciones que puede realizar sobre el *dwh*, de acuerdo con su perfil. Se requiere entonces disponer de información sobre usuarios y permisos.

- El componente de acceso deberá conocer los datos con los que cuenta el *dwh*, para poder ofrecerle al usuario la información que solicite. Será necesario, entonces, que se tenga acceso a un catálogo de los datos disponibles para cada tipo de usuario.

Por otro lado, es necesario tener en cuenta que no son sólo los usuarios los que acceden al *dwh*, sino diversas aplicaciones. Aunque las aplicaciones de *data mining* son las más frecuentes, se debe considerar que al *dwh* se puede acceder mediante simples consultas de bases de datos, generadores de informes, aplicaciones personalizadas, paquetes de negocio o mediante facilidades incluidas en productos (hojas de cálculo, por ejemplo).

Implantación del data warehouse

Una vez se ha definido la infraestructura necesaria tanto de hardware como software para el *dwh*, se han determinado los métodos y reglas que lo gobernarán y se ha formado al personal de su mantenimiento, se tiene que analizar su funcionamiento.

En la Figura 6.4 se muestran los componentes de una aplicación de *dwh*.

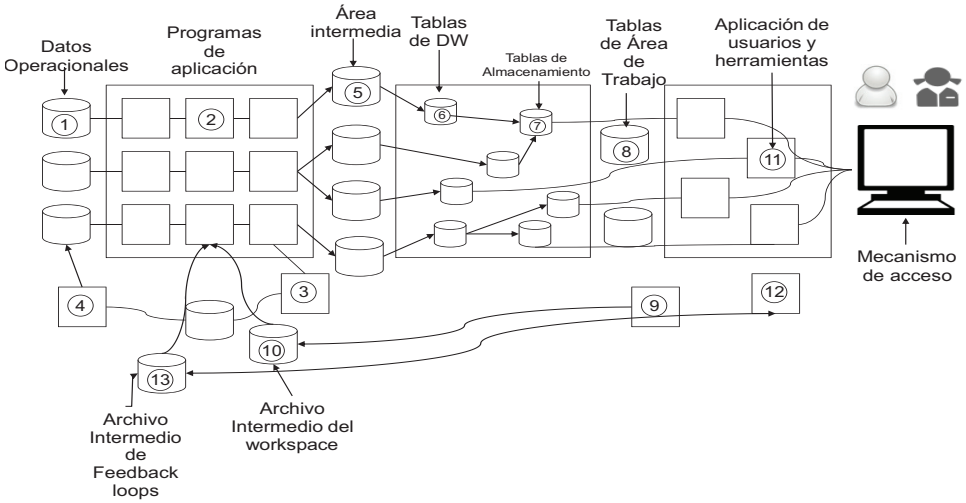

Figura 6.4 Componentes de una aplicación de dwh

El primer paso de análisis es la identificación de los datos y de sus fuentes (1). Los datos pasan por los programas de adquisición (2), para obtenerlos, limpiarlos y preprocesarlos.

Después de que los datos están limpios, los programas de *back-flush* los recogen para ponerlos en una zona de almacenamiento (3), de la que otros programas la tomarán para devolverlos a sus fuentes originales (4). La carga se realiza sobre las tablas relacionales diseñadas con este propósito (6). Se

construyen también resúmenes, cálculos y referencias cruzadas que requiera la aplicación que usará los datos (7).

Cuando sea necesario, se generan las tablas de *workspaces* (8), que periódicamente se exploran en busca de cambios (9), y que se llevan a una zona de almacenamiento para utilizarlos en posteriores integraciones de datos (10). Finalmente, las aplicaciones de usuario acceden a los datos que requieran (11). Se puede, adicionalmente, disponer de mecanismos para realización de bucles de retroalimentación (12), similar al de las áreas de trabajo, cuyos cambios se almacenarán en otro archivo (13) que también forma parte del futuro proceso de carga de datos en el *dwh*.

Diseño multidimensional

Ejemplos típicos de consultas hechas sobre un *dwh* son, entre otros, calcular el número promedio de ventas de computadores por sucursal en el último mes, los 10 programas de televisión con más audiencia la semana pasada y comparar las ventas de arequipe en Navidad contra cualquier otro periodo del año.

Estas consultas, aunque son diferentes, tienen en común que todas analizan un hecho básico objetivo de una organización: ventas en la primera y tercera y audiencias en la segunda. El contenido y presentación de los resultados podrá variar, pero la naturaleza de las consultas es la misma en todos los casos.

La Figura 6.5 muestra la arquitectura de un *dwh* en la cual se puede observar la interacción entre las bases de datos operacionales y los usuarios finales.

Los datos sobre los hechos que ocurren en una organización tienen ciertas características que se pueden explotar cuando se diseña la base de datos. Los hechos son transacciones que han ocurrido en algún tiempo pasado y que no cambiarán en el futuro. Cada hecho se puede analizar con respecto a distintos factores. Por ejemplo, las ventas se pueden analizar por sucursal, por períodos temporales, por tipos de clientes o por zonas demográficas.

Uno de los mayores retos que se debe enfrentar cuando se diseña una base de datos de soporte a la decisión es estructurar una solución que sea efectiva a lo largo del tiempo, es decir, que no tenga que cambiar cuando el perfil de consulta cambie.

En este contexto y para diseñar la base de datos del *dwh* se requiere del modelo multidimensional.

Modelo multidimensional: principios básicos

El modelado multidimensional es una técnica de diseño lógico que busca presentar los datos en un marco estándar intuitivo y que permita accesos de alto desempeño. Es inherentemente dimensional y los modelos se transfor-

man en relacionales con algunas restricciones importantes. Cada modelo dimensional se compone de una tabla, denominada la tabla de hechos, con una llave compuesta y un conjunto de tablas más pequeñas denominadas tablas de dimensión. Cada tabla de dimensión tiene una llave primaria simple que corresponde con una parte de la llave compuesta de la tabla de hechos.

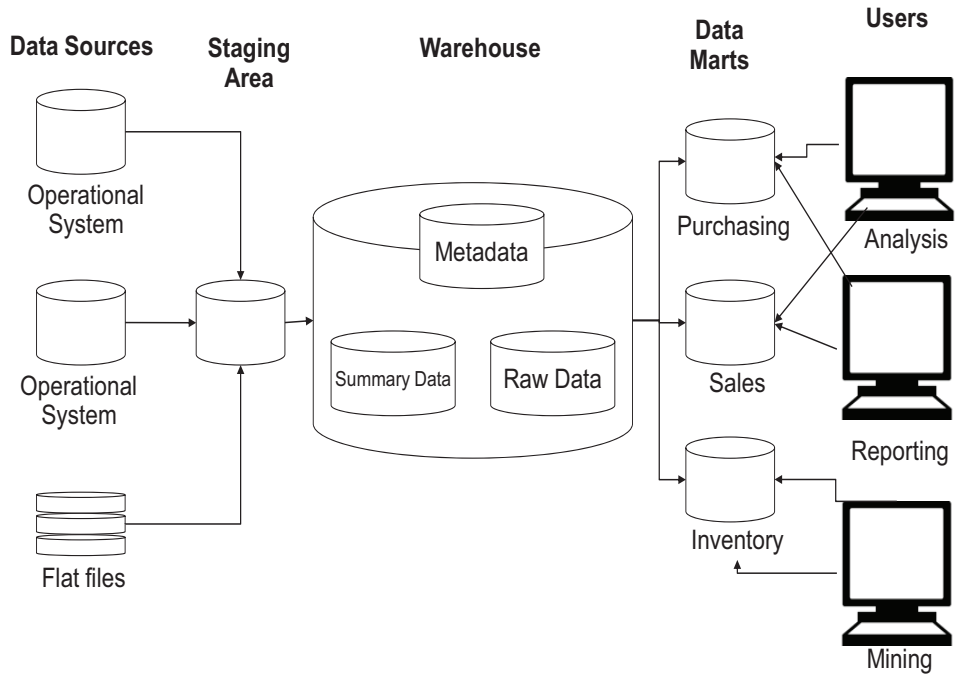

Figura 6.5 Arquitectura de un dwh

Tomada de http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/img/cncpt172.gif

En la Figura 6.6 se muestra un modelo, llamado por su apariencia ***diseño en "estrella" (STAR)***. La tabla de hechos almacena la información sobre un hecho del negocio y las tablas de dimensión almacenan información sobre los factores que influyen y con base en los cuales se puede analizar el hecho.

En el ejemplo, la tabla de hechos representa las ventas que se pueden analizar desde el punto de vista de clientes, fechas, sucursales o productos.

Tablas de hechos y de dimensiones

La idea subyacente en el modelado multidimensional es que la organización se puede representar como un cubo cuyas celdas contienen valores medidos y cuyas aristas representan las dimensiones naturales de los datos. En la práctica se permiten más de tres dimensiones, por lo que no se trataría exactamente de un cubo sino de un hipercubo, aunque sean los términos de cubo y cubo de datos (Figura 6.7) los más utilizados en la literatura.

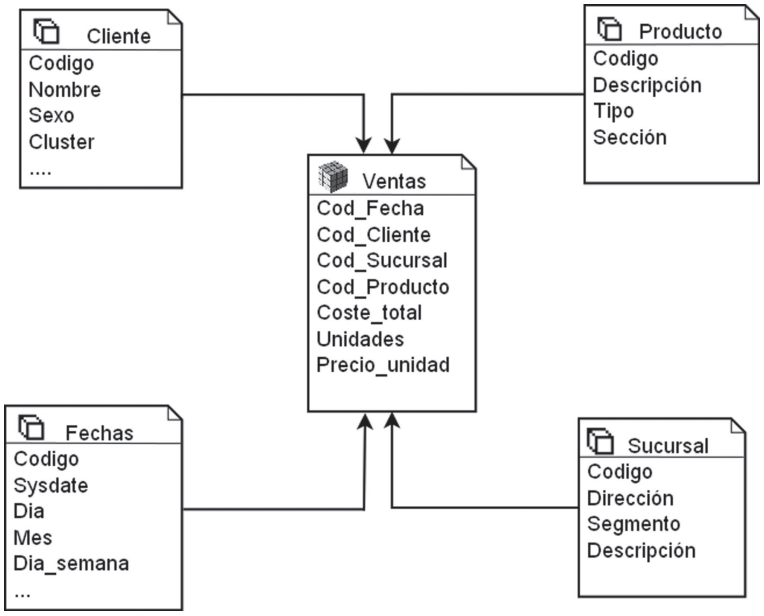

Figura 6.6 Estrella de ventas

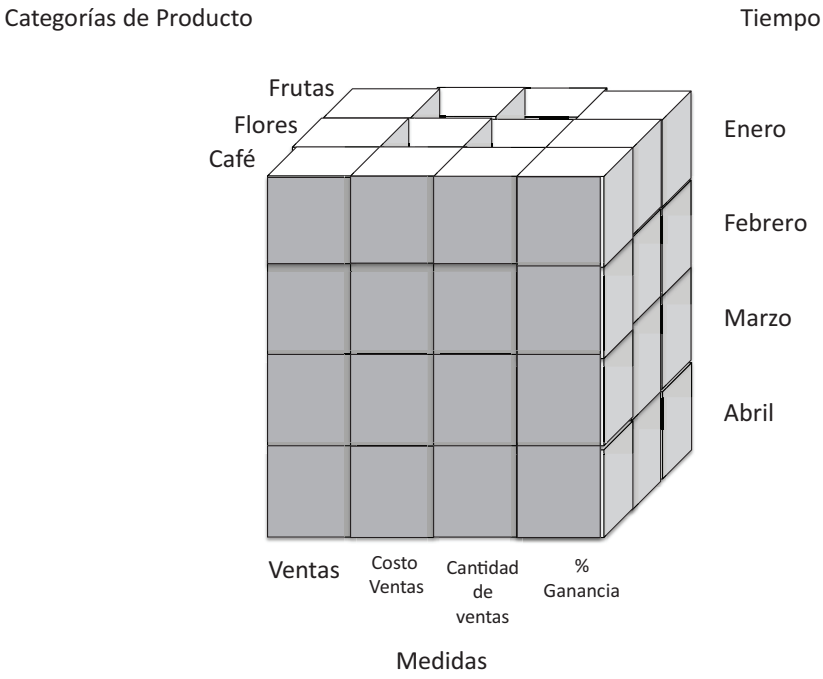

Figura 6.7 Cubo de datos

Los modelos dimensionales en el mundo real tienden a tener entre 4 y 15 dimensiones. Los modelos con 2 o 3 dimensiones son raros y llevan a sospechar que se deberían añadir más dimensiones. Diseños con más de 20 dimensiones, por otra parte, llevan a pensar que hay dimensiones superfluas o dimensiones que se podrían unir.

Atributos

En un modelo dimensional se distinguen hechos y atributos. Los atributos suelen ser campos textuales que describen características de las entidades u objetos y generalmente se utilizan para designar propiedades de las dimensiones. Un ejemplo claro de atributos son las características de los productos. Es fácil ver en este caso que el sabor o la marca de un producto no se mide sino que se conoce de antemano.

Hechos

Un hecho, generalmente, no se conoce de antemano, es una observación del mundo real. La mayoría de los hechos son numéricos, aunque se puede dar el caso de que tengan valores textuales. Cualquier valor numérico es candidato a ser un hecho y no un atributo.

En ocasiones se duda de si un determinado valor es un hecho o un atributo, sin embargo, si es un valor no se conoce de antemano sino que depende del momento, ese valor debería ser un hecho. De esta manera, en el ejemplo del producto, su precio es un atributo. Sin embargo, si el precio depende del momento de la transacción, éste dejaría de ser un atributo para ser un hecho.

Tablas de hechos

Una tabla de hechos contiene observaciones sobre el hecho que se analiza a lo largo del tiempo. Una tabla de hechos tiene una llave primaria compuesta y un conjunto de atributos (generalmente numéricos) que se denominan hechos. Cada uno de los componentes de la llave compuesta (llave foránea) es la llave primaria de una dimensión. Las tablas de hechos crecen cuando se refresca el *dwh*, puesto que contienen el histórico a lo largo del tiempo. En este sentido, es importante, cuando se diseñan, tener en cuenta algunos aspectos relacionados con la elección de las llaves primarias de las dimensiones, con la aditividad de los hechos que contiene y con el tamaño de la tabla.

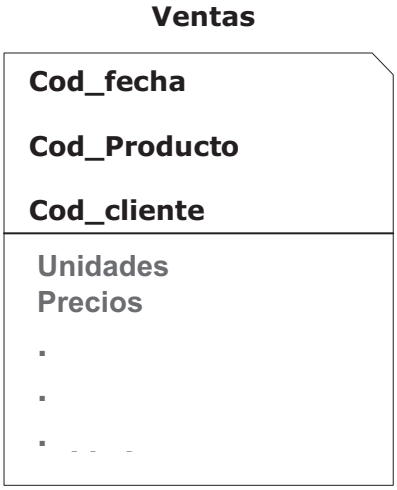

Figura 6.8 Tabla de hechos de ventas

En la tabla de ventas de la Figura 6.8, los primeros tres valores (Cod_fecha, Cod_Producto y Cod_cliente) forman la llave de la tabla de hechos y son llaves foráneas con respecto a las tablas de dimensión fecha, producto y cliente. Los valores restantes (Unidades y Precios) representan los hechos de la tabla y se miden en cada transacción. Esta tabla de hechos representa las unidades que de un producto adquiere un cliente en una determinada fecha y el precio al que las adquiere.

Algunas de las consultas posibles que se podrían hacer sobre esta tabla de hechos serían:

- Producto más vendido en una determinada fecha.
- Comparativo de ventas de un determinado mes de este año con el equivalente de años anteriores.
- Análisis de los productos vendidos por tipos de clientes.

Tamaño de la tabla de hechos

"Llenar" el *dwh* es hacer concatenaciones sobre las tablas de hechos. Por lo tanto, son las tablas de hechos las que más crecen. Por otra parte y como consecuencia de lo anterior, las modificaciones a esta tabla se deberían evitar en la medida de lo posible. Sobre todo, en lo que se refiere a modificaciones en cascada sobre los valores de las llaves foráneas porque, debido al volumen de datos que se requiere tratar, la mayoría de estas tablas se particionan. Por esta razón es recomendable:

- Nunca incluir campos de más en la tabla de hechos y solamente los necesarios.
- Medir el tamaño de cada campo y nunca añadir bits más allá de lo considerado necesario en cada caso.

- No utilizar llaves de producción y generar llaves ficticias o sintéticas.

Las llaves de las tablas en un *dwh* son un aspecto diferenciador con respecto a las de las bases de datos tradicionales. En los diseños de bases de datos tradicionales las llaves corresponden con valores del mundo real. En el *dwh* se aconseja no utilizar nunca llaves de producción y generar llaves ficticias o sintéticas propias para el *dwh*. Esta política trata de evitar que los cambios de las bases de datos de producción se tengan que propagar al *dwh*, lo que implicaría cambios en el *dwh* (e.g. re-estructuración de índices, modificación de tablas particionadas).

Otra razón para utilizar llaves sintéticas es que éstas facilitan almacenar datos en situaciones tales como cuando "no se sabe de momento", "es desconocido en este caso", etc. El hacer uso de llaves generadas permite solucionar el problema de las dimensiones con datos que pueden cambiar en el tiempo.

Tablas de dimensión

Una dimensión es una colección de atributos textuales que están altamente correlacionados unos con otros. En el modelo multidimensional, los atributos de las tablas de dimensión juegan un papel muy importante porque sobre ellos se aplican restricciones para las consultas.

Los atributos textuales que describen entidades se organizan en dimensiones. En el ejemplo de ventas, existe una dimensión para la fecha, el producto, el cliente y posiblemente otra para el vendedor o la sucursal, dependiendo del nivel de detalle.

Calidad de los atributos

Los atributos de las dimensiones son la entrada para las consultas al *dwh*. Sobre ellos se aplican las restricciones más interesantes en las consultas. Por esta razón, se ha llegado a afirmar que la calidad del *dwh* se mide por la calidad de los atributos de las dimensiones.

Una tabla de dimensión ideal contiene muchos campos de texto que describen a todos y cada uno de los miembros de la dimensión. Estos campos textuales deben ser palabras completas o frases enteras, nunca códigos o abreviaturas. Es responsabilidad del equipo de desarrollo del *dwh* mantener la consistencia y la calidad en los valores de estos campos. Campos incompletos o pobremente administrados llevarán a resultados de consultas pobres e inconsistentes.

La dimensión fecha juega un papel muy importante en los sistemas de *dwh* porque una tabla de hechos es básicamente una serie de observaciones en el tiempo. Siempre se tiene una o más tablas de fechas en cualquier diseño. Dentro de la tabla de fechas se distingue entre la tabla de fechas con granularidad de día y la tabla de fecha con granularidad minuto/segundo.

Cuando se requiere tener el detalle del segundo o minuto en que ocurre una transacción se aconseja mantener dos tablas de fechas. La primera, con una granularidad hasta el día y la segunda en la que se mantiene información de los segundos y minutos de cada día. En la Figura 6.9 se muestra el diseño de la tabla de fechas con granularidad en días. Esta tabla está detallada para permitir tener la información de periodos fiscales, fiestas nacionales y locales, meses, semanas, etc.

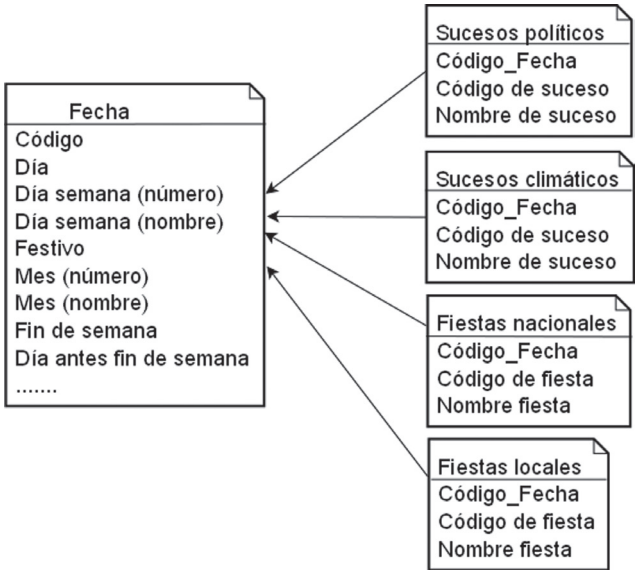

Figura 6.9 Diseño de las fechas con granularidad a nivel de días

Jerarquías en las dimensiones

Algunos de los atributos de una tabla dimensión definen jerarquías que facilitan el tratamiento de los datos a distinto nivel de granularidad. Una misma dimensión puede contener atributos que no incluyen jerarquías, y atributos que sí las generan. Regularmente, es muy común el caso de que los atributos de una dimensión definan más de una jerarquía para poder adaptarse a descripciones de datos provenientes de distintas partes de la organización.

Las jerarquías permiten las consultas OLAP. Cuando existen jerarquías un usuario puede hacer consultas con distintos niveles de granularidad o de generalidad. Las consultas de OLAP pueden ser de los siguientes tipos:

- **Roll up (drill-up),** para resumir los datos subiendo o reduciendo las dimensiones,
- **Drill down (roll down),** para bajar en la jerarquía o aumentar dimensiones,
- **Slice and dice** si se quiere hacer proyección y selección,

- **Pivot (rotar)** para reorientar el cubo, y
- **Drill across,** cuando las consultas involucran más de una tabla de hechos.

Normalización de las tablas de dimensión

Otro aspecto muy importante a tener en cuenta en el diseño de las tablas de dimensión está relacionado con la normalización de estas tablas.

La normalización evita problemas de inconsistencias cuando se actualiza una base de datos. En un *dwh* sólo se insertan datos y el objetivo fundamental es conseguir que las consultas se ejecuten eficientemente. Como consecuencia, con frecuencia en el diseño de estas tablas se rompe la normalización buscando reducir el tiempo que toma resolver una consulta. Por tanto, una dimensión está "*snowflaked*" cuando los campos de baja cardinalidad están en sus tablas originales y no se han incluido directamente en la tabla de dimensión; esto es, cuando se encuentran normalizadas.

Sin embargo, esta no es una técnica recomendable en *data warehousing*, puesto que al romper la normalización, en este entorno, no se pierde consistencia ya que los procesos de carga del *dwh* están controlados y la ganancia en espacio de almacenamiento es irrelevante si se tiene en cuenta el tamaño que tendrá la tabla de hechos.

A pesar de la prohibición de normalizar tablas con atributos de baja cardinalidad, existen ocasiones en las cuales se aconseja normalizar. Diferentes situaciones se muestra en las Figuras 6.10, 6.11 y 6.12.

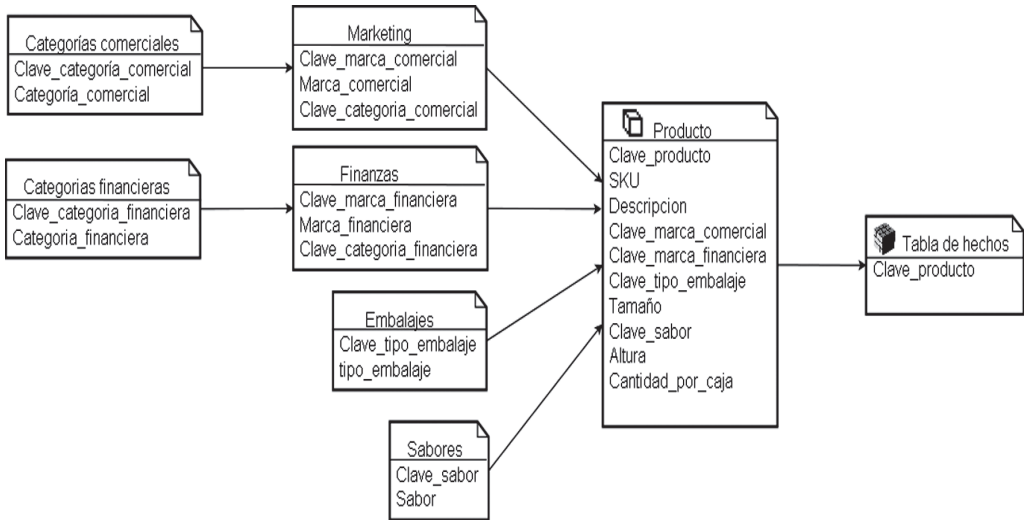

Figura 6.10 Dimensión producto normalizada

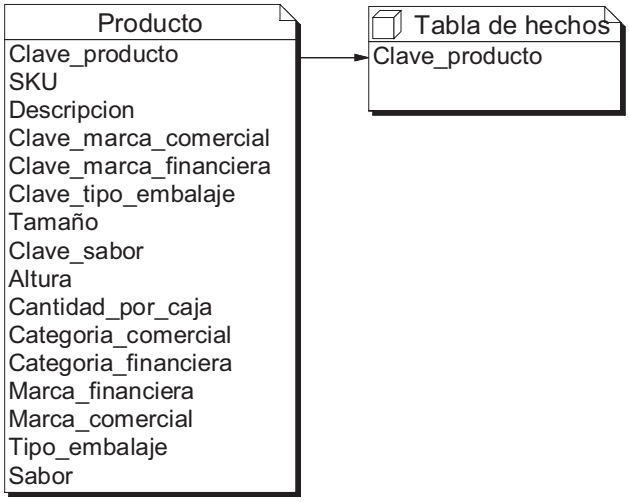

Figura 6.11 Dimensión producto no normalizada

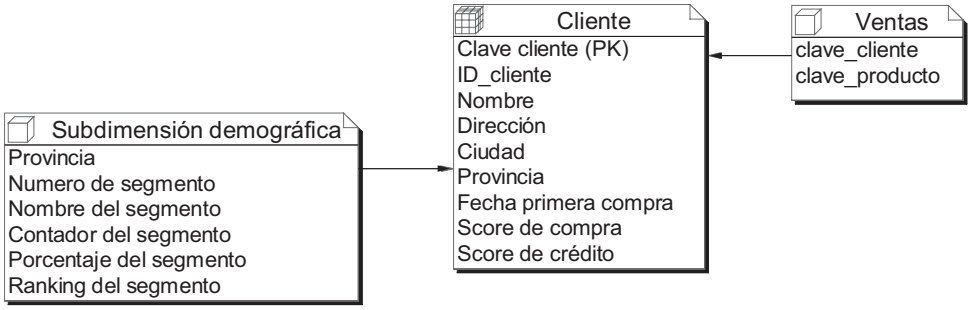

Figura 6.12 Dimensión cliente no normalizada

Data mining

Introducción

Data mining es la etapa central del proceso de descubrimiento de conocimiento (*Knowledge Discovery Process - kdd*), aunque comercialmente se asocia con el proceso mismo de *kdd*. Aunque no hay una única definición de *data mining* o *kdd*, la siguiente es la más conocida: *Proceso de extracción de información desconocida con anterioridad, válida y potencialmente útil a partir de grandes bases de datos para usarla con posterioridad para tomar decisiones importantes de negocio* [Fayyad96].

El proceso de descubrimiento de conocimiento está integrado por varias etapas como las que aparecen en la Figura 6.13.

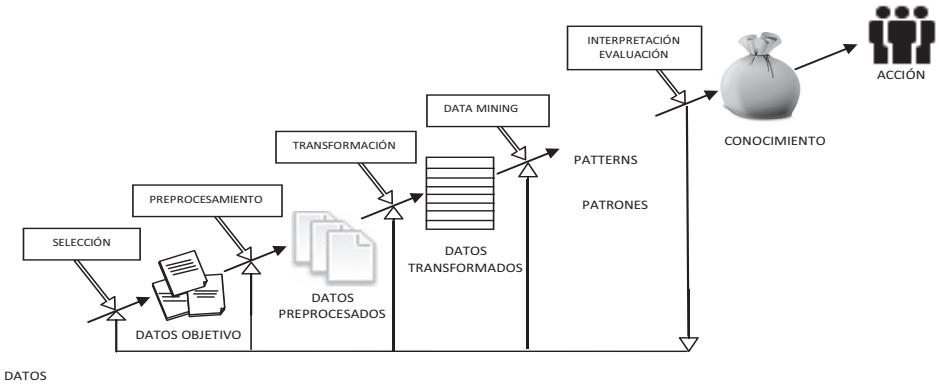

Figura 6.13 Etapas del proceso de descubrimiento de conocimiento

Tomada de http://www.kmining.com/

Una vez se conoce el dominio de la aplicación, las metas que tiene el usuario y el conocimiento relevante, se construye un conjunto de datos objetivo con base en el cual se van a extraer patrones novedosos.

Las siguientes dos etapas requieren del mayor esfuerzo en relación con las otras etapas. El conjunto de datos objetivo se analiza con dos propósitos fundamentales: entender su significado y detectar errores generados por la integración de los datos. Estos problemas surgen porque los datos pueden provenir de diferentes fuentes y cada fuente puede almacenarlos de distinta manera y formato. El conjunto de datos se debe, además, preprocesar para remover ruido y valores fuera de rango (*outliers*), entre otros. Es necesario también, en esta etapa, tomar decisiones en relación con el tratamiento de los datos faltantes (*missing and empty data*).

A partir del conjunto de datos preprocesados, se requiere identificar las variables significativas y transformar los datos para reducir su volumen y para adecuarlos al formato de entrada de los algoritmos de *data mining* que se deben aplicar.

La etapa de minería de datos (*data mining*) implica escoger la tarea de minería que se va a aplicar: clasificación, asociación, *clustering*, entre otras. Una vez seleccionada la tarea se escogen el(los) algoritmo(s) que se van a aplicar al conjunto de datos, definir los parámetros del algoritmo y adaptar los datos a la entrada del algoritmo. Su aplicación puede producir resultados en forma de reglas, árboles o agrupamientos, entre otros.

En la etapa de interpretación y evaluación de resultados se utilizan técnicas de visualización para ver mejor los resultados obtenidos. Una vez presentados los resultados el usuario debe interpretarlos. Si los resultados no son satisfactorios para el usuario es posible volver a cualquiera de las etapas anteriores, aplicar de nuevo los algoritmos con otros parámetros e incluso ejecutar otro algoritmo de *data mining*. En este sentido, el proceso

es iterativo e interactivo puesto que siempre requiere la intervención de un experto en el dominio.

Cuando los resultados son satisfactorios, el conocimiento obtenido se puede aplicar en el contexto. En esta fase también se debe determinar cómo utilizar los resultados obtenidos.

A partir de bases de datos de suficiente tamaño y calidad la aplicación de procesos de *data mining* puede generar nuevas oportunidades de negocio facilitando, entre otras, la ***predicción*** automática de tendencias y comportamientos. Un ejemplo típico de un problema predictivo es el *marketing* dirigido. *Data mining* utiliza los datos generados en campañas anteriores para identificar los objetivos que con mayor probabilidad harán rentables futuras campañas.

Es posible también aplicar el proceso de *data mining* para ***descubrir*** patrones previamente desconocidos. Las herramientas de *data mining* filtran los datos contenidos en bases de datos muy grandes e identifican patrones previamente ocultos. Un ejemplo de descubrimiento de patrones es el análisis de datos de ventas para identificar productos aparentemente no relacionados que a menudo se adquieren juntos. Otros problemas de descubrimiento de patrones incluyen la detección de transacciones fraudulentas con tarjetas de crédito y la identificación de datos anómalos que podrían representar errores de entrada.

El estándar CRISP-DM

El estándar CRISP-DM (www.crisp-dm.org), *Cross-Industry Standard Process for Data Mining* [Chapman00], se propuso en 1996 como resultado de la cooperación de *DaimlerChrysler, SPSS y NCR* con el propósito de apoyar y facilitar el diseño y puesta en marcha de proyectos de desarrollo de soluciones de *data mining*. Una de sus ventajas es que no está ligado a ningún producto comercial ni a ningún tipo específico de aplicación.

El estándar describe la metodología de *data mining CRISP-DM* como un modelo de proceso jerárquico. Este modelo se divide en cuatro niveles de abstracción. El primero, el más general y el último, el más específico. Dentro del primer nivel, el proceso de *data mining* se estructura en fases o etapas, integradas por tareas genéricas de segundo nivel, que se pretende, cubran todas las posibles situaciones que se puedan encontrar. En el tercer nivel se describe cómo se llevarán a cabo las acciones asociadas con las tareas genéricas en situaciones específicas. Finalmente, en el cuarto nivel se registran las acciones, decisiones y resultados asociados con el proceso de *data mining* objetivo.

En CRISP-DM se distingue entre el **modelo de referencia** y la **guía de usuario**. El primero ofrece una vista general de las fases, tareas y salidas. La guía de usuario ofrece información detallada, consejos y recomendaciones que se deben tener en cuenta en cada fase.

Por otra parte, se distinguen también cuatro dimensiones de contexto de *data mining*:

- El **dominio de aplicación,** es el área específica en que tiene lugar el proyecto de *data mining*.
- El **tipo de problema** de *data mining,* describe los objetivos del proyecto particular.
- El **aspecto técnico,** que cubre aspectos específicos relacionados con los desafíos que suelen tener lugar durante el proyecto.
- La dimensión de **herramientas y técnicas** especifica las herramientas y/o técnicas de *data mining* que se aplican durante el proyecto.

CRISP-DM identifica las siguientes etapas, integrando un proceso de *data mining*, como se muestra en la Figura 6.14:

1. Entendimiento del negocio (*Business understanding)*
2. Entendimiento de los datos (*Data understanding*)
3. Preparación de los datos (*Data preparation*)
4. Modelamiento (*Data mining modeling*)
5. Evaluación e interpretación de resultados (*Evaluation and interpretation of results*)
6. Aplicación (*Deployment*)

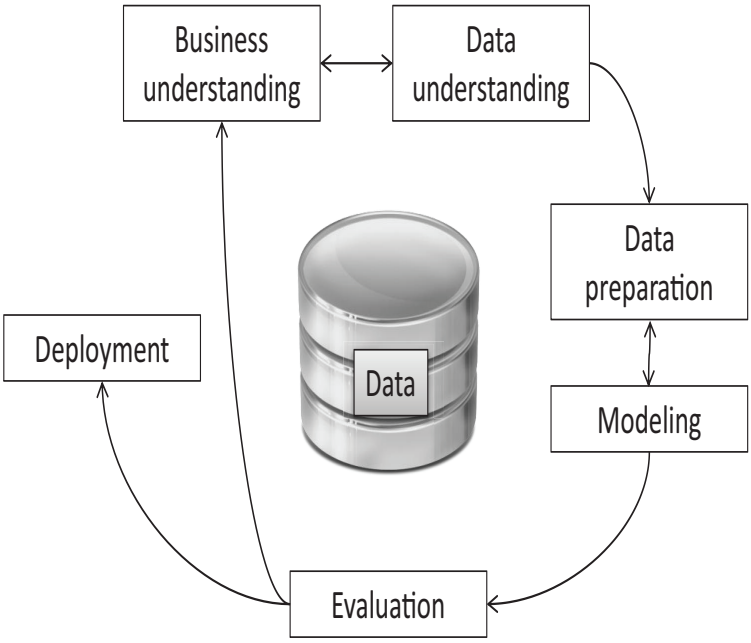

Figura 6.14 Etapas del modelo de proceso CRISP-DM

La etapa de entendimiento del negocio está centrada en entender los objetivos del proyecto de descubrimiento de conocimiento y en los requerimientos, desde el punto de vista del negocio. A partir de este conocimiento se puede definir el problema de minería y proponer un plan para cumplir con los objetivos propuestos.

La etapa de entendimiento de los datos se empieza con un conjunto inicial de datos para tratar de entenderlos, de identificar problemas de calidad de los datos y para detectar subconjuntos de estos que permitan emitir algunas hipótesis sobre información oculta.

En la etapa de preparación de los datos, se seleccionan atributos, registros, tablas y se limpian y transforman los datos. Todas estas tareas tienen como propósito construir el conjunto de datos que serán la entrada de las herramientas de modelamiento.

Las técnicas que se van a aplicar se seleccionan en la etapa de modelamiento. Es necesario también, en esta etapa, ajustar los parámetros para las técnicas seleccionadas. Es posible que de nuevo se deba llevar a cabo tareas de preparación de datos, en aquellos casos en los cuales éstos se deban transformar para ajustarse a la especificación de la entrada de las técnicas.

En la etapa de evaluación, ya se cuenta con el o los modelos construidos que tengan la más alta calidad desde la perspectiva del análisis de datos. Se requiere evaluar si algún aspecto importante no se ha tenido en cuenta o no se le ha dado la importancia necesaria.

Finalmente, en la etapa de utilización y aplicación, el conocimiento obtenido se debe organizar y presentar al usuario en una forma en que éste lo pueda usar. Los resultados de esta fase, dependiendo de los requerimientos, pueden ser simples como un reporte o complejos de manera que impliquen implementar un proceso de minería a través de la organización.

Para cada una de estas etapas o fases, la metodología CRISP-DM propone un conjunto de tareas genéricas asociadas con salidas que se detallan en [Chapman00].

La etapa de data mining: Tipos de problemas y enfoques

En la etapa de *data mining* se aplican los algoritmos de búsqueda de patrones sobre los datos previamente preprocesados.

Debido a que es en esta fase en la que se ejecutan los algoritmos de *data mining*, es fundamental el rol del analista de datos. Es necesario que el analista disponga del conjunto de datos preprocesado, los correspondientes metadatos y toda la información que de los datos se haya previamente extraído en las fases anteriores de análisis. Los resultados de esta etapa dependen del tipo de meta que se desea obtener. El análisis de los resultados es uno de los pasos más importantes del proceso.

Tipos de problemas de data mining

Los problemas de *data mining* se pueden clasificar en *descriptivos*, o de

descubrimiento indirecto puesto que no hay una meta a predecir, y *predictivos* o de descubrimiento directo. La meta de un problema **descriptivo** es encontrar una descripción de los datos de análisis.

Ejemplos de este tipo de problemas son conocer las características de los clientes de una organización, encontrar los productos que frecuentemente se compran juntos o los síntomas de enfermedades que se presentan juntos. Los problemas descriptivos pueden requerir la aplicación de análisis de segmentación o de asociación.

El análisis de segmentación (aprendizaje no supervisado) hace referencia a los problemas donde la meta es encontrar grupos homogéneos en la población objetivo.

El análisis de asociaciones a los problemas, cuyo objetivo es obtener relaciones entre los valores de los atributos del conjunto de datos objetivo.

Por otra parte, los problemas **predictivos** tienen como meta obtener un modelo que en un futuro se pueda aplicar para predecir comportamientos. A pesar de que en el proceso de *data mining* las técnicas aplicadas para la obtención del modelo son técnicas de inducción sobre los datos de origen, el resultado (modelo) se aplica para predecir. Dependiendo del tipo de variable que se quiere predecir (categórica o numérica) los problemas predictivos pueden ser de clasificación o de predicción de valores.

Los ***problemas de clasificación*** son aquellos en los que la variable que se va a predecir toma un número finito de valores, es decir, la variable es categórica. Encontrar un modelo que a partir de datos históricos de clientes clasificados como “buenos”, “regulares” y “malos”, establezca el tipo de un nuevo cliente es un ejemplo de problema de clasificación.

Los ***problemas de predicción de valores*** son aquellos en los cuales la variable que se va a predecir es numérica. Encontrar un modelo para determinar la probabilidad de que un cliente que está pidiendo un préstamo lo pague no es un ejemplo de este tipo de problemas.

Tareas de data mining

Una técnica se implementa en un algoritmo para llevar a cabo tareas de *data mining*. Rara es la ocasión en la que un único algoritmo es la solución de un problema. El éxito de un proceso de *data mining* se basa, fundamentalmente, en el conocimiento de los datos que se están analizando y de las técnicas que se estén aplicando.

En tareas de clasificación predictiva se suelen aplicar modelos de aprendizaje supervisado como los árboles de decisión, árboles de inducción y las redes neuronales, entre otros. En ellos se utiliza un conjunto de datos de entrenamiento para crear el modelo que, posteriormente, se usa para clasificar individuos desconocidos. Para la predicción de valores se emplean modelos de regresión.

Si la tarea de clasificación es no supervisada, se aplican técnicas de clus-

tering demográfico. Cada registro de la base de datos se asigna a uno de los segmentos creados mediante una estrategia de creación de segmentos.

Cuando se trata de la tarea de asociación, el objetivo es encontrar elementos que implican la presencia de otros. El resultado de esta técnica son reglas del tipo "if X then Y". Uno de los algoritmos de asociación más utilizados es Apriori [AS94]. En Apriori se cuentan las ocurrencias de todas las posibles combinaciones de elementos. Muchas implementaciones de Apriori se han propuesto [AIS93], [AS94], [BMUT97], [Bodon03], [Borgelt05], [EZ03], [HPY00] intentando reducir el número de lecturas de la base de datos.

Tareas de *data mining*: Clasificación y predicción

Tanto la clasificación como la predicción buscan determinar el valor de cierto atributo (etiqueta de clase) a partir de un conjunto de datos de prueba y con base en los valores de otros atributos.

Esta tarea requiere, por tanto, conocer el valor del atributo etiqueta de clase, para el conjunto de datos con base en el cual se extrae el modelo. Este modelo permite estimar los valores del atributo etiqueta con base en el resto de valores de atributos.

Una forma de representar un modelo de clasificación, ampliamente utilizado, es el basado en reglas, como aparece en el siguiente ejemplo, en el que el atributo etiqueta de clase es ABANDONO:

SI MODELO = SP02 Y GASTO_MENSUAL > 105
    ENTONCES ABANDONO = si
SI EDAD < 18 Y GASTO_MENSUAL > 35
    ENTONCES ABANDONO = si
SI OFERTA = TT12 Y RESIDENCIA = CALI Y CASADO = si
    ENTONCES ABANDONO = no

Una vez construido el modelo, éste se utiliza para predecir el valor del atributo etiqueta en aquellos casos en los cuales este valor se desconoce. El requerimiento de conocer la etiqueta de clase para el conjunto de entrenamiento es lo que diferencia las técnicas de aprendizaje supervisado de las de aprendizaje no supervisado.

El conjunto que se utiliza para extraer el modelo se divide en dos subconjuntos: el **conjunto de entrenamiento** (*training data set*), usado para generar el modelo, y el **conjunto de prueba** (*testing data set*), empleado para contrastar la predicción del modelo con datos no usados en su entrenamiento.

La diferencia entre clasificación y predicción es únicamente la naturaleza del atributo etiqueta. Cuando el atributo etiqueta es nominal (o discreto) (e.g. si hay o no abandono, modelo de carro que se compra, tipo de hipoteca que se va a contratar) se habla de **clasificación**. Cuando el valor que se quiere estimar es numérico o continuo (e.g. gasto en llamadas del próximo

mes, beneficios de producción de un determinado producto) se habla de **predicción**.

Una buena parte de los algoritmos usados en la actualidad para problemas de clasificación se derivan del campo de la inteligencia artificial denominado *Machine Learning*.

Un algoritmo de clasificación consiste de dos etapas: construcción y explotación del modelo. En la etapa de **construcción del modelo** se integran las fases de entrenamiento y validación que permiten determinar la fiabilidad del mismo.

En la etapa de **explotación,** el modelo obtenido se aplica para estimar los valores del atributo etiqueta en aquellos casos en los cuales ésta no se conoce.

En los algoritmos de clasificación surge el concepto de **clase**. A una misma clase pertenecen todas las instancias (tuplas o registros de datos) que tienen el mismo valor en el atributo etiqueta. El algoritmo de clasificación construye un modelo que sea capaz de discernir cada una de las clases en las cuales está dividido el conjunto de datos de entrada.

Para evaluar la adecuación de los modelos y para desechar técnicas o algoritmos cuyas soluciones no son adecuadas, se utilizan los siguientes criterios:

- **Capacidad predictiva**: representa el grado de acierto que tiene el modelo a la hora de estimar el valor del atributo etiqueta para casos no usados en su entrenamiento.
- **Eficiencia**: hace referencia al tiempo requerido, tanto en la construcción del modelo como en la evaluación de cada una de las instancias por medio del mismo. En aplicaciones de tiempo real, el tiempo de evaluación del modelo es crítico, puesto que la respuesta está limitada en el tiempo.
- **Robustez**: indica lo sensible que es el algoritmo, y por tanto el modelo extraído de él, a datos con *ruido*. Se denomina datos con ruido a cualquier tipo de error en la medición, almacenamiento o recuperación de los datos de carácter puntual.
- **Escalabilidad**: mide la posibilidad de aplicar una determinada técnica sobre un conjunto de datos de gran tamaño (número de instancias y atributos muy alto). En cierto tipo de entornos los problemas que se van a tratar pueden incluir millones de instancias y centenares o miles de atributos.
- **Interpretabilidad de los modelos**: significa que el modelo obtenido se pueda entender. Los modelos basados en reglas son fácilmente comprensibles por usuarios humanos, mientras que modelos compuestos por complejas funciones estadísticas sólo son interpretables por usuarios con avanzados conocimientos en probabilidad y estadística. Esta característica no es tan importante si lo que se pretende es aplicar el modelo, puesto que éste se puede tratar como *caja negra*.

Técnicas de clasificación son, entre otras, los árboles y las reglas de decisión y las funciones de probabilidad. Entre los algoritmos de clasificación mas ampliamente utilizados están ID3 (Induction Tree)[BS02], [Quinlan96], [SL91]. C4.5 es una versión modificada de ID3 que soporta diferentes distribuciones de datos numéricos (continuos), discretización interna y eliminación de ruido o poda, entre otras.

Otras técnicas basadas en conceptos probabilísticos son el clasificador ingenuo Naive-Bayes [HF08] y las redes neuronales, entre otras.

Algunos aspectos sobre los modelos extraídos

• *Calidad*

Uno de los aspectos fundamentales, relacionados con los modelos de clasificación o predicción, es la evaluación de su calidad. Dos peligros pueden surgir a la hora de hacer esta validación a la ligera o por medio de mecanismos poco estrictos.

Por una parte, si los casos utilizados en la validación no se eligen convenientemente o incluso no se diferencian de los usados en el proceso de entrenamiento se obtendrán resultados excesivamente optimistas. Evaluar la calidad de un modelo de clasificación con los mismos datos usados en su entrenamiento produciría resultados poco ajustados a la realidad.

Por otra, y en el otro extremo, está la validación de los modelos con datos que no tiene ninguna relación con los utilizados en el entrenamiento. Por ejemplo, si se trata de evaluar perfiles de compra de clientes de una multinacional, no es muy lógico usar para entrenar los datos de clientes europeos y contrastarlo con los datos provenientes de clientes americanos o asiáticos. Es muy probable que los gustos o necesidades de los clientes tengan una estrecha relación con su cultura y con la sociedad en la que viven. Los resultados obtenidos en estos casos podrían ser excesivamente pesimistas.

• *Sobreentrenamiento (overfitting)*

Uno de los problemas más habituales de los modelos generados es el sobre-ajuste u *overfitting*. Este problema aparece cuando el modelo está excesivamente ajustado a los datos de entrenamiento haciendo que cualquier registro no usado en el entrenamiento se clasifique erróneamente. Este problema se debe a una mala elección de atributos en los datos de entrada, un conjunto de entrenamiento que no representa todas las instancias del problema bajo estudio, o a un error en la definición de los parámetros de convergencia o del límite en el número de iteraciones del algoritmo.

Un ejemplo extremo de este problema es el que se produciría si como atributo de entrada en una lista de clientes aparece el número de cédula/pasaporte. Un mecanismo de clasificación que no descarte este atributo debido a su alta cardinalidad (número de valores) lo seleccionaría como mejor atributo de clasificación. Generando reglas del estilo "*Si el Pasaporte=E345199*

entonces compra=si". Estas reglas pueden clasificar perfectamente el conjunto de datos de entrenamiento pero con cualquier registro diferente no utilizado, la validación produciría resultados erróneos.

Sin llegar a casos tan extremos sí es muy probable que un modelo muy ajustado a los datos de entrenamiento funcione de forma poco acertada sobre los datos de validación. Este problema es fácil de detectar cuando el modelo se representa mediante reglas puesto que el número y la complejidad de las reglas son elevados. Otra característica de los modelos con este problema es el bajo soporte (número de instancias) de cada una de las reglas. Las reglas más generales son preferibles aunque su porcentaje de acierto sea menor a reglas más específicas con un 100% de acierto.

Soluciones al problema de *overfitting* se basan en atacar las dos posibles causas: los atributos utilizados y los parámetros de funcionamiento del algoritmo. En el primer caso es necesario hacer un análisis previo y es muy aconsejable analizar el dominio para saber cuáles atributos o combinación de atributos identifica unívocamente a cada instancia.

El problema de *overfitting* tiene como consecuencia la falta de generalidad del modelo. Es decir, cada vez que cambia el conjunto de datos se requiere ajustar nuevamente el modelo. La falta de generalidad del modelo es una característica de las técnicas para construir modelos basados en conjuntos de prueba y entrenamiento.

Técnicas de validación de modelos de clasificación

Validar un modelo de clasificación o predicción implica llevar a cabo dos tareas: aplicar el algoritmo sobre un subconjunto de los datos para obtener un modelo y validar el modelo extraído con otro conjunto de datos independiente. Después de aplicar el modelo sobre los datos de validación se pueden calcular algunas medidas de error. Un aspecto importante en el proceso de validación es la división de los datos entre conjunto de validación y prueba. Las siguientes son algunas de las técnicas que se pueden utilizar:

• *Holdout*

Los datos se dividen aleatoriamente en dos conjuntos, uno de entrenamiento y otro de prueba. Típicamente la división que se aplica es de dos tercios para entrenamiento y un tercio para validación.

Este método tiene dos grandes limitaciones: La primera es que la división aleatoria puede hacer que todos los datos de una determinada clase (valor del atributo etiqueta) se encuentren, exclusivamente, o en el conjunto de validación o en el de entrenamiento. Cuando sólo están en el de validación el modelo que se construye genera reglas que predigan una clase diferente a la de los datos de entrenamiento y en consecuencia todos los elementos de dicha clase no se podrán clasificar correctamente.

Esta limitación se resuelve aplicando la estratificación como estrategia

de división de los conjuntos validación y entrenamiento (Figura 6.15). La estratificación consiste en mantener, en ambos conjuntos, la relación entre las diferentes clases que hay en los datos originales. De esta forma, todas las clases están igualmente representadas en todas las divisiones.

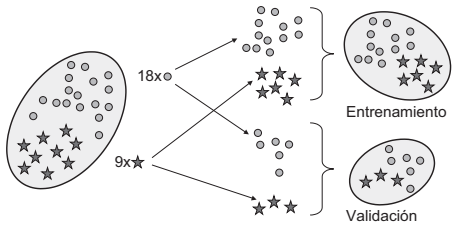

Figura 6.15 Selección de conjuntos de entrenamiento y prueba

El segundo problema es un poco más sutil. Si los datos de una de las clases son en realidad instancias de distinta naturaleza y las representativas de alguno de estos tipos se agrupan en un mismo subconjunto se estaría ante el mismo problema.

Suponga, por ejemplo, que se quiere predecir los clientes que comprarían un equipo electrónico de gama alta (por ejemplo, un equipo de reproducción de video de alta definición). De esta manera, se tendrían dos clases de equivalencia *compra=no* y *compra=si*. Dentro de los que sí lo compran puede haber clientes que lo compren porque sean profesionales de fotografía o imagen que lo utilicen para su trabajo y clientes con un nivel adquisitivo alto que lo compren como equipo personal sin fines profesionales. Los perfiles de ambos clientes son diferentes y, sin embargo, a nivel de clasificación pertenecen a la misma clase de equivalencia. Un modelo correcto debería clasificarlos a los dos, pero si, por ejemplo, los usuarios profesionales del equipo son aleatoriamente incluidos en el conjunto de validación, el modelo no los tendrá en cuenta.

• *Cross-Validation*

Esta técnica de *validación cruzada* consiste en crear varias divisiones aleatorias del mismo tamaño (o *folds*), generalmente 10, de las cuales 9 se utilizan para entrenar y una para validar. El proceso se repite 10 veces seleccionando un conjunto distinto para la validación. Al final del proceso cada conjunto se ha utilizado 9 veces en entrenamiento y una vez en validación. La ventaja de este mecanismo es que la aleatoriedad debida al proceso de división en *folds* se atenúa al repetirse equitativamente su utilización en validación y entrenamiento. Adicionalmente, el número de datos usado en el entrenamiento (90%) es superior al caso de *holdout* (66%), lo que tiende a mejorar el modelo.

El mecanismo de evaluación más comúnmente utilizado es de 10 *folds*,

10-fold cross-validation. Cuando el proceso de división en subconjuntos se hace de forma estratificada se habla de *10-fold stratified cross-validation.*

• *Bootstrap*

Este método se basa también en la selección aleatoria de elementos para conformar los dos conjuntos de validación y prueba. A diferencia de las anteriores, el proceso aleatorio de selección se realiza con remplazamiento, haciendo que una misma instancia pueda ser elegida múltiples veces. La selección aleatoria con reemplazo busca mantener constante la probabilidad de seleccionar una instancia y sólo en raras ocasiones —con conjuntos de datos pequeños— una instancia aparece múltiples veces. Las instancias que no han sido elegidas son las que conforman el conjunto de datos de validación y el conjunto de entrenamiento puede tener datos repetidos.

• *Leave-One-Out*

Este método se puede ver como una variante del *cross-validation* cuando el número de *folds* es igual al número de datos disponibles. La idea es usar todos los datos para entrenar dejando uno que es el que se usa para validar. El proceso se repite tantas veces como datos se tengan.

Este método elimina el factor aleatorio de la división que los métodos anteriores tienen, pero no es aplicable para conjuntos de datos de gran tamaño debido al alto número de iteraciones que habría que realizar.

Tareas de *data mining*: Asociación

En esta tarea se trata de encontrar asociaciones interesantes en un conjunto de datos. Formalmente, el problema se define como sigue [AS94]:

Definición del problema

Sea $I = i_1, i_2, ..., i_m$ un conjunto de literales o *items*. Sea D un conjunto de transacciones T, donde cada transacción T es un conjunto de *items* tales que $T \subseteq I$. Asociado a cada transacción existe un único identificador *TID*.

Se dice que una transacción T contiene a X (X es un subconjunto de I), si $X \subseteq T$.

Una **regla de asociación** es una implicación de la forma $X \rightarrow Y$. La regla $X \rightarrow Y$ tiene una confianza *c* en el conjunto de transacciones D, si el *c%* de las transacciones de D que contiene a X también contienen a Y.

La regla $X \rightarrow Y$ tiene un soporte *s* en el conjunto de transacciones D si el *s%* de las transacciones de D contienen a $X \cup Y$. Es decir, $soporte(A \rightarrow B) = P(A \cup B)$ y $confianza(A \rightarrow B) = P(B \mid A)$

Las reglas de asociación cuyo valor de soporte y confianza estén por encima de un mínimo establecido por el usuario se denominan reglas fuertes.

Dado un conjunto de transacciones D, el problema de extraer reglas de asociación consiste en generar todas las reglas cuyo soporte y confianza es

mayor que los mínimos especificados *minsup (minimum support)* y *minconf (minimum confidence)*, respectivamente.

Calculando reglas de asociación

El problema de descubrir todas las reglas de asociación se puede descomponer en dos sub-problemas:

1. Encontrar todos los conjuntos de *ítems*, *itemsets*, que aparecen en el conjunto de transacciones D un número de veces superior al *minsup*. Los *itemsets* que aparecen en un número de transacciones superior al mínimo requerido, se denominan *large itemsets*.
2. Utilizar los *large itemsets* para generar las reglas de asociación cuya confianza sea al menos *minconf*.

Obteniendo *large itemsets*

Los algoritmos para el cálculo de *large itemsets* hacen numerosas pasadas sobre los datos. En la primera pasada se calculan los *1-itemsets,* conjuntos de ítems de tamaño 1 cuyo soporte es al menos el *minsup*. En las subsiguientes pasadas, se toman estos *large itemsets* obtenidos en la pasada anterior y se utilizan como semilla para generar potenciales *large itemsets* llamados candidatos. Durante esta pasada se calcula su soporte. Al final de cada pasada se determina cuáles de los candidatos son realmente *large itemsets* que, de nuevo, serán la semilla para la pasada siguiente. Este proceso continúa hasta que no se puedan generar *itemsets* nuevos.

El algoritmo más básico para calcular asociaciones frecuentes es *Apriori* [Agrawal94], cuyo nombre se basa en el hecho de que el algoritmo utiliza información previa de los *itemsets*.

El algoritmo *Apriori* y sus variantes *Apriori* y *AprioriTid* se detallan en [AS94].

Pseudocódigo de Apriori

1) L_1 = {1-large itemsets};
2) **for** (k = 2; $L_{k-1} \neq \emptyset$; k++) **do begin**
C_k = apriori-gen (L_{k-1}); // Nuevos candidatos
forall transacciones $t \in D$ **do begin**
    C_t = subset (C_k , t); // Candidatos contenidos en t
    forall candidatos $c \in C_t$ **do**
        c.contador++;
end
L_k = { $c \in C_k$ | c.contador >minsup}
10) **end**
11) respuesta = $\cup_k L_k$;

Generación de candidatos (apriori_gen)
// función apriori-gen

insert int C_k
select $p.\text{item}_1, p.\text{item}_2, \ldots, p.\text{item}_{k-1}, q.\text{item}_{k-1}$
from $L_{k-1}\, p, L_{k-1}\, q$
where $p.\text{item}_1 = q.\text{item}_1, \ldots, p.\text{item}_{k-2} = q.\text{item}_{k-2}, p.\text{item}_{k-1} < q.\text{item}_{k-1}$;

A continuación, en la fase de poda, se eliminan todas las asociaciones $c \in C_k$ que contengan subconjuntos de c de (k - 1*) items* que no estén en L_{k-1} :

forall asociaciones $c \in C_k$ **do**
forall (k - 1)-subconjuntos s de c **do**
if (s $\notin L_{k-1}$) **then**
delete c de C_k;

Descubrimiento de reglas
Para generar las reglas, para cada *large itemset l,* se calculan todos los subconjuntos no vacíos de *l*. Para cada subconjunto *a*, una regla de la forma *a->*(*l-a*) se genera si (soporte(*l*)/soporte(*a*)) es superior al valor de la mínima confianza (*minconf*). Todos los subconjuntos de *l* se consideran para generar reglas con múltiples consecuentes.

Referencias

[AS94] Agrawal R., Srikant R. Fast Algorithms for Mining Association Rules. Proceeding of the 20th Int'l Conference on Very Large Databases, *Santiago, Chile, 1994. Expanded version available as IBM Research Report RJ9839, June 1994*. Disponible en http://rakesh.agrawal-family.com/papers/vldb94apriori.pdf.

[AIS93] Agrawal R., Imielinski T., Swami A. Mining Association Rules between Sets of Items in Large Databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data,* Washington, D.C., United States, pp. 207-216, 1993.

[Bodon03] Bodon F. A Fast Apriori Implementation. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* Melbourne, USA, 2003. Disponible en http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-90/bodon.pdf.

[Borgelt05] Borgelt C. An Implementation of FP-Growth Algorithm.

Proceedings of the 1st International Workshop on Open Source Data Mining OSDM '05, Chicago, Illinois pp. 1-5, 2005. Disponible en http://www.borgelt.net/papers/fpgrowth.pdf.

[BC04] Baralis E., Chiusano S. Essential Classification Rule Sets. *ACM Transaction on Database Systems* (29)4:635-674, 2004.

[BMUT97] Brin S. , Motwani R., Ullman J.D., Tsur S., Dynamic Itemset Counting and Implication Rules for Market Basket Data. *ACM SIGMOD Record(6),* 2:255-264, 1997. Disponible en http://w3.umh.ac.be/~ssi/tpdm/brin97.pdf.

[BS02] Blockeel H., Struyf J. Efficient Algorithm for Decision Tree Cross-Validation. *Journal of Machine Learning Research* (3):621-650, 2002.

[Chapman00] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R. CRISP-DM 1.0: Step-by-step data mining guide, CRISP-DM consortium, 2000. Disponible en http://www.crisp-dm.org.

[CK02] Cios K., Kurgan L. Trends in Data Mining and Knowledge Discovery. *En: Pal N. R., Jain, L. C. and Teoderesku, N. (Eds.), Knowledge Discovery in Advanced Information Systems, 2002.* Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.5317.

[CLNP06] Calders T., Lakshmanan L., Ng R., Paredaens J. Expressive Power of an Algebra for Data Mining. *ACM Transaction on Database Systems* (31)4:1169-1214, 2006.

[Dzeroski03] Dzeroski S. Multi-Relational Data Mining: An Introduction. *ACM SIGKDD Exploration Newsletter* (5)1:1-16, 2003. Disponible en www.sigkdd.org/explorations/issue5-1/Dzeroski.pdf.

[EZ04] El-Hajj M., Zaiane O. COFI Approach for Mining Frequent Itemsets Revisited.

Proceedings of the 9th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery DMKD '04, pp. 70-75, Paris, France, 2004.

[EM07] Esmeir S., Markovitch S. Anytime Learning of Decision Trees. *Journal of Machine Learning Research* (8):891-933, 2007.

[EZ03] El-Hajj M., Zaiane O. COFI-Tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* Melbourne, USA, 2003. Disponible en http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-90/elhajj.pdf.

[FPS96] Fayyad, U., Piatestky-Shapiro, G., Smyth, P. From Data Mining to Knowledge Discovery: An Overview. In (Fayyad, U., Piatestky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.)) Advances in Knowledge Discovery and Data Mining, AAAI Press, The MIT Press, Menlo Park, CA, 1996, pp. 1-34.

[GH06] Geng L., Hamilton H.J. Interestingness Measures for Data Mining: A Survey. *ACM Computing Surveys* (38)3, Article 9, 2006.

[Goethals03] Goethals B. Survey on Frequent Pattern Mining, Technical Report (2003), University of Helsinki, Finland. Disponible en http://www.adrem.ua.ac.be/~goethals/software/survey.pdf.

[GZ03] Goethals B, Zaki M. Advances in Frequent Itemset Mining Implementation: Introduction to FIMI03. *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* Melbourne, USA, 2003. Disponible en http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-90/intro.pdf.

[HF08] Hall M., Frank E. Combining Naïve Bayes and Decision Tables. *Disponible en* www.cs.waikato.ac.nz/~ml/publications/2008/HallAndFrankFLAIRS08.pdf.

[HFWKZ96] Han J., Fu Y., Wang W., Koperski K., Zaiane O. DMQL: A Data Mining Query Language for Relational Databases. *1996 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery KMKD'96,* pp.27-33, Montreal, Canadá, 1996. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.4217.

[HK2006] Han J., Kamber M. Data mining: Concepts and Techniques. Morgan Kaufmann Publishers, ELSEVIER 2006 http://www.cs.sfu.ca/~han/dmbook. Data mining: practical machine learning tools w/ java implementations.

[HPYM04] Han J., Pei J., Yin Y., Mao R. Mining frequent patterns without candidate generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery* (8): 53-87, 2004. Disponible en http://www-faculty.cs.uiuc.edu/~hanj/pdf/dami04_fptree.pdf.

[HPY00] Han J., Pei J., Yin Y. Mining frequent patterns without candidate generation. *ACM SIGMOD Record* (29)2:1-12, 2000. Disponible en www.cs.uiuc.edu/~hanj/pdf/dami04_fptree.pdf.

[HPYM04] Han J., Pei J., Yin Y. Mining frequent patterns without candidate generation. *Data Mining and Knowledge Discovery* (8):53-87, 2004.

[HCC92] Han J., Cai Y., Cercone N. Knowledge Discovery in Databases: An Attribute-Oriented Approach. *ACM SIGMOD Record* (29)2:1-12, 2000.

[HL05] Hu H., Li J. Using Association Rules to Make Rule-based Classifiers Robust. *Proceedings of the 16th Australasian Database Conference ADC '05,* pp. 47-54, Newcastle, Australia, 2005.

[JFA08] Jin R., Fuhry D., Alali A. Cost-Based Query Optimization for Complex Pattern Mining on Multiple Databases. *Proceedings of the 11th International Conference on Extending Database Technology, EDBT '08,* pp. 380-391, Nantes, France, 2008.

[JG06] Jiang N., Gruenwald L. Research Issues in Data Stream Association Rule Mining. *SIGMOD Record* (35)1:14-19, 2006.

[JLS08] Jen T., Laurent D., Spyratos N. Mining All Frequent Projection-Selection Queries from a Relational Table. *Proceedings of the 11th International Conference on Extending Database Technology, EDBT '08,* pp. 368-379, Nantes, France, 2008.

[KJC04] Kantarcioglu M., Jin J., Clifton C. When Do Data Mining Results Violate Privacy?. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '04,* pp. 599-604, Seattle, WA, USA, 2004.

[LCH+07] Liu Y., Cai J., Huang Z., Yu J., Yin J. Fast Detection of Database System Abuse Behaviors Based on Data Mining Approach. *Proceedings of the 2nd International Conference on Scalable Information Systems InfoScale '07,* pp. 1-7, Suzhou, China 2007.

[LPWH02] Liu J., Pan Y., Wang K., Han J. Mining frequent item sets by opportunistic projection. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge Discovery and Data Mining, KDD '02,* pp. 229-237, Edmonton, Alberta, Canada, 2002. Disponible en http://www.cs.sfu.ca/~wangk/pub/KDD-Junqaing-Arial.pdf.

[Quinlan96] Quinlan J.R. Learning Decision Tree Classifiers. *ACM Computing Surveys* (28)1:71-71 , 1996.

[SL91] Safavian S.R., Landgrebe D. A Survey of Decision Tree Classifier Methodology. *IEEETransaction on Systems, Man and Cybernetics* (21)3:660-674, 1991. Disponible en http://cobweb.ecn.purdue.edu/~landgreb/SMC91.pdf.

[SS05] Shang X., Sattler K.U. Depth-First Frequent Itemset Mining in Relational Databases. *Proceedings of the 2005 ACM Symposium on Applied Computing SAC '05,* pp. 1112-1117, Santa Fe, New Mexico, 2005.

[SSG04] Shang X., Sattler K.U., Geist I. SQL Based Frequent Pattern Mining Without Candidate Generation. *Proceedings of the 2004 ACM Symposium on Applied Computing, SAC '04,* pp 618-619, Nicosia, Cyprus, 2004.

[STA98] Sarawagi S., Thomas S., Agrawal R. Integrating association rule mining with relational database systems: alternatives and implications. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, SIGMOD'98,* pp. 343-354. Disponible en http://rakesh.agrawal-family.com/papers/sigmod98dbi.pdf.

[YPK00] Yoshizawa T., Pramudiono I., Kitsuregawa M. SQL Based Association Rule Mining using commercial RDBMS (IMB DB2 UDB EEE). *In Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery,* pp. 301-306, 2000. Disponible en http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.9604.

[YWY07] Yuan J., Wu Y., Yang M. From Frequent Itemsets to Semantically Meaningful Visual Patterns. *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '07,* pp. 864-873, San José, California, USA, 2007.

Bibliografía básica anotada

[FPS96] Fayyad, U., Piatetstky-Shapiro, G., Smyth, P. From Data Mining to Knowledge Discovery: An Overview. In (Fayyad, U., Piatetstky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.)) Advances in Knowledge Discovery and Data Mining, AAAI Press, The MIT Press, Menlo Park, CA, 1996, pp. 1-34.

En este artículo se describe el proceso de descubrimiento de conocimiento en bases de datos (*KDD- Knowledge Discovery in Databases*) como un

"proceso no trivial de identificación de patrones válidos, novedosos, potencialmente útiles y entendibles a partir de los datos". El proceso propuesto, interactivo e iterativo, está integrado por nueve etapas. Las etapas que se propones son: entendimiento del dominio, creación de un conjunto de datos objetivo, limpieza y pre-procesamiento de datos, reducción y proyección de datos, selección de tareas de *data mining*, selección de los algoritmos de *data mining*, *data mining* o búsqueda de patrones interesantes, interpretación de patrones obtenidos y incorporación del conocimiento descubierto a la organización.

La Figura 6.16 muestra las etapas que integran el proceso.

Se describen la clasificación, regresión, clustering, sumarización, modelamiento de dependencias y cambio y detección de desviación como las tareas principales de *data mining*.

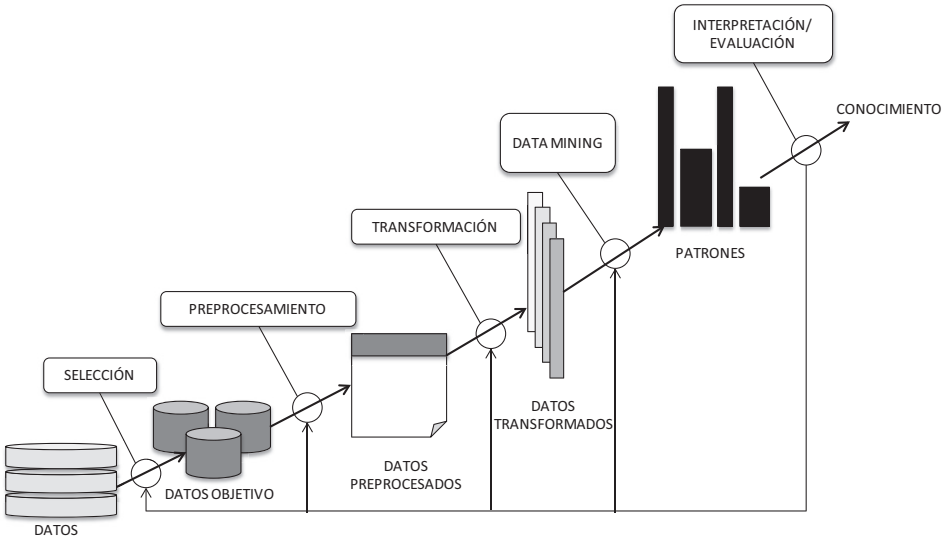

Figura 6.16 Etapas del proceso KDD

Tomada de http://www.infoviswiki.net/index.php?title=Knowledge_Discovery_in_Databases_(KDD)

[Chapman00] Chapman, P., Clinton, J., Kerber, R.,Khabaza, T., Reinartz, T., Shearer, C., Wirth, R. CRISP-DM 1.0: Step-by-step data mining guide, CRISP-DM consortium, 2000. *Disponible en http://www.crisp-dm.org.*

Se describe de manera detallada el estándar para proyectos de minería de datos, CRISP-DM. Cada una de las tareas asociadas con cada una de las etapas que integran el proceso: entendimiento del negocio, entendimiento de los datos, preparación de datos, evaluación, utilización y aplicación, se describe. Un conjunto de entregables de cada una de las tareas que forman parte de una etapa se presentan también en el estándar.

[HPY00] Han J., Pei J., Yin Y. Mining frequent patterns without candidate generation. *ACM SIGMOD Record* (29)2:1-12, 2000. Disponible en www.cs.uiuc.edu/~hanj/pdf/dami04_fptree.pdf.

En este artículo se presenta un algoritmo para generación de *itemsets* frecuentes que no requiere de la etapa de generación de candidatos propia del algoritmo *Apriori*.

Una estructura compacta denominada *FP-tree* se construye para almacenar conjuntos de ítems. El nodo raíz del árbol se etiqueta con *"null*". Cada una de las transacciones se procesa inicialmente para calcular el soporte de los conjuntos de ítems de tamaño 1. A partir de este cálculo se almacenan en el árbol prefijo, las transacciones. Los nodos hoja del árbol son conjuntos de ítems frecuentes de tamaño 1. Las transacciones que comparten un prefijo común de conjuntos de ítems frecuentes (en un orden de frecuencia), se combinan en la estructura de árbol. Después de que el árbol de patrones frecuentes (FP) se ha construido no se requiere volver a acceder a la base de datos de transacciones.

El algoritmo para construir el *FP-Tree* se presenta en el artículo para el cual se analizan varias propiedades. La estructura se explora para obtener todos los patrones frecuentes.

Resultados experimentales de desempeño se presentan con respecto a los algoritmos Apriori [AS94] y TreeProjection [LPWH02].

PÁGINA EN BLANCO
EN LA EDICIÓN IMPRESA

BIBLIOGRAFÍA GENERAL

[AA93] Atzeni P., De Antonellis V. Relational Database Theory. The Benjamin/Cummings Publishing Company, Inc., 1993.

[AHV95] Abiteboul S., Hull R., Vianu V. Foundations of Databases. Addison-Wesley Publishing Company, 1995.

[Cattell94] Cattell R. G. G. Object Data Management. Object-oriented and extended relational database systems. Addison-Wesley Publishing Company 1994.

[CBB+97] Cattell R. G. G., Barry D. K., Bartels D., Eastman J., Gamerman S., Jordan D., Springer A., Strickland H., Wade D. The Object Data Standard: ODMG 2.0. Morgan Kuffmann, San Francisco, California, 1997.

[CBB+00] CattellR.G.G., Barry D.K., Berler M., Eastman J., Jordan D., Russell C., Schadow O., Stanienda T., Velez F. The Object Data Standard:ODMG 3.0. Morgan Kuffmann, San Francisco, California, 2000.

[CHS+98] Cabena P., Hadjinian P., Stadler R., Verhees J., Zanasi A. . Discovering Data Mining from concept to implementation. Prentice Hall. 1998.

[CBS98] Connolly T., Begg C., Strachan A. Database Systems. A practical Approach to design, Implementation and Management. Second Edition, Addison-Wesley, 1998.

[Codd90] Codd Edgard. The Relational Model for Database Management, Version 2. Addison-Wesley Publishing Company, 1990.

[Date98] Date C. J. Relational DATABASE. Writings 1994-1997. Addison-Wesley, 1998.

[Date2005] Date C. J. Database in depth. Relational Theory for Practitioners. O'Reilly, 2005.

[Fayyad96] Fayyad, U., Grinstein, G. G., Wierse, A. (eds.) Information Visualization in Data Mining and Knowledge Discovery, Morgan Kaufmann 2001.

[FPSU96] Fayyad, U., Piatetstky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.)) Advances in Knowledge Discovery and Data Mining, AAAI Press, The MIT Press, Menlo Park, CA, 1996.

[GUW00] Garcia-Molina H., Ullman J., Widom J. Database System Implementation. Prentice Hall, 2000.

[HMS2001] Hand D., Mannila H., Smyth P. Principles of Data Mining. The MIT Press, 2001.

[Inmon96] Inmon W.H. Building the DataWarehouse. Second Edition. John Willey and Sons Inc. 1996.

[IWG97] Inmon H., Welch J., Glassey K. Managing the Data Warehouse. John Willey and Sons Inc., 1997.

[Maier83] Maier David. The Theory of Relational Databases. Disponible en http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html.

[Mattison97] Mattison R. Data Warehouse Strategies Technologies and Techniques. McGraw Hill, 1997.

[ML+2003] Data Mining and Decision Support. Integration and Collaboration. Edited by Dunja Mladenic, Nada Lavrac, Marko Bohanec, Steve Moyl. Kluwer Academic Publishers, 2003.

[Pyle99] Pyle Dorian. Data Preparation for Data Mining. Morgan Kaufmann, 1999.

[SM96] Stonebraker M., Moore D. Object-relational DBMSs. The Next Great Wave. Morgan Kaufmann Publishers, Inc, 1996.

[SW2005] Simsion G. C., Witt G. C. Data Modeling Essentials. Morgan Kaufmann Publishers, Third Edition, 2005.

[Ullman89] Ullman, J. D. Principles of Database and Knowledge Base Systems, vol. II: The New Technologies. Computer Science Press, Rockville, MD, 1989.

[WF05] Witten I., Frank E. Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). Morgan Kaufmann, 2005. *Documentación de WEKA (Waikato Environment for Knowledge Analisis) Data Mining Software in Java.* Disponible en http://www.cs.waikato.ac.nz/~ml/weka/index.html.

Programa Editorial