Introducción a las Bases de Datos

Jesús Vegas

Departamento de Informática
Universidad de Valladolid
jvegas@infor.uva.es

Prefacio

Estos apuntes surgen con el objetivo de satisfacer las necesidades de los alumnos que cursan la asignatura de Bases de Datos de la Diplomatura en Estadística de la Universidad de Valladolid. Y hay que entenderlos desde ese punto de vista, como una introducción a las Bases de Datos para alumnos del primer ciclo, con unos ciertos conocimientos previos de informática; aunque no los considero suficientemente consistentes como para servir a alumnos de Ingeniería Informática. A la hora de hacer públicos estos apuntes he considerado más importante el hecho de recopilar conocimientos que ayuden al seguimiento de la asignatura, que el riesgo de mostrar mis limitaciones en este área de conocimiento.

Al ser unos apuntes no han pretendido ser un trabajo original, y atribuyo el mérito de la mayoría del texto y de los ejemplos que se incluye en ellos a los autores de los libros reseñados en la bibliografía. Yo sólo me he dedicado a extraer de cada uno de ellos los apartados que más me han seducido, dándoles una continuidad argumental. También tengo que agradecer a Carmen Hernández la mayoría de los ejercicios del capítulo 3.

Como toda obra escrita, estos apuntes son subceptibles de contener errores, y a buen seguro que así es; por lo que encomiento al lector su descubrimiento y que me los señale para que con la ayuda de todos vayan desapareciendo poco a poco.

Estos apuntes se encuentran disponibles en la dirección `http://descartes.dcs.fi.uva.es/~jvegas`, o se pueden solicitar mandando un correo a `jvegas@infor.uva.es`.

Valladolid, 1998

Índice General

Capítulo 1

Conceptos y Objetivos de los Sistemas de Bases de Datos

En este capítulo analizaremos el concepto de bases de datos, examinando sus ventajas e inconvenientes en comparación con los sistemas tradicionales de ficheros, y finalizaremos exponiendo los distintos niveles de abstracción en una base de datos.

1.1 De los Sistemas Tradicionales de Ficheros a las Bases de Datos

Los sistemas informáticos tradicionales han sido llamados por algunos autores sistemas *orientados hacia el proceso*:

- se pone el énfasis en los tratamientos que reciben los datos, los cuales se almacenan en ficheros diseñados para una determinada aplicación, lo que determina el modo de almacenamiento.
- Las aplicaciones se analizan e implantan con entera independencia unas de otras, y
- los datos no se suelen compartir entre ellas, sino que se duplican siempre que los diferentes procesos los necesitan.

Los datos se recogen varias veces y se encuentran repetidos en los distintos archivos. Esta redundancia, además de malgastar recursos, origina a menudo divergencias en los resultados. Tambíen produce

- aumento en los tiempos de proceso, al repetirse los mismos controles y operaciones en los distintos ficheros.
- inconsistencias, debidas a que la actualización de los mismos datos, cuando se encuentran en más de un fichero, no se suele realizar de forma simultánea en todos los ficheros.

Por otra parte, la dependencia de los datos respecto al soporte físico y a los programas da lugar a una falta de flexibilidad y de adaptabilidad frente a los cambios, que repercute muy negativamente en el rendimiento de conjunto del sistema informático. (ver Figura 1.1).

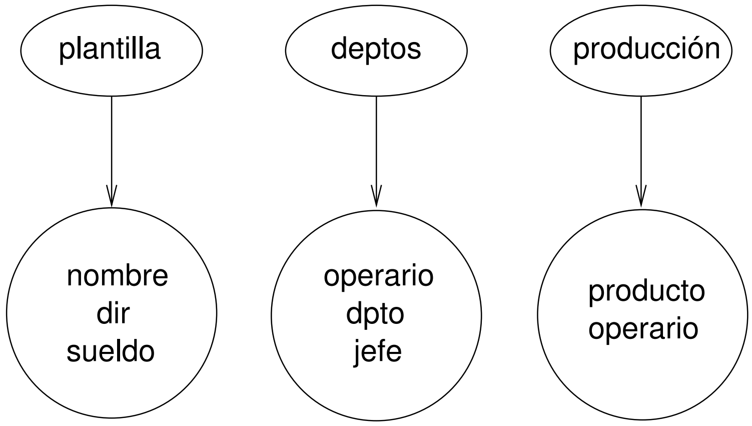


Figura 1.1: Sistema de información orientado al proceso

Los problemas son aún más acusados cuando se presentan demandas inesperadas de información.

De este análisis se deduce claramente la necesidad de una gestión más racional del conjunto de datos, surgiendo así un nuevo enfoque que se apoya sobre una base de datos en la cual los datos son recogidos y almacenados una sola vez, con independencia de los tratamientos: sistemas *orientados hacia los datos* (ver Figura 1.2).

1.2 Ventajas de las Bases de Datos Frente a los Ficheros Clásicos

Las bases de datos, presentan una multitud de ventajas frente a los sistemas clásicos de ficheros.

Debemos, sin embargo, señalar que las bases de datos no son la panacea universal que solucionará todos los problemas que la información plantea. Las bases de datos son un instrumento, y su éxito o su fracaso estará condicionado por el uso que de ellas sepamos hacer tanto usuarios, técnicos, y los responsables de los sistemas de bases de datos.

Las ventajas de los sistemas de bases de datos son, entre otras, las siguientes:

- Independencia de los datos respecto a los tratamientos y viceversa.

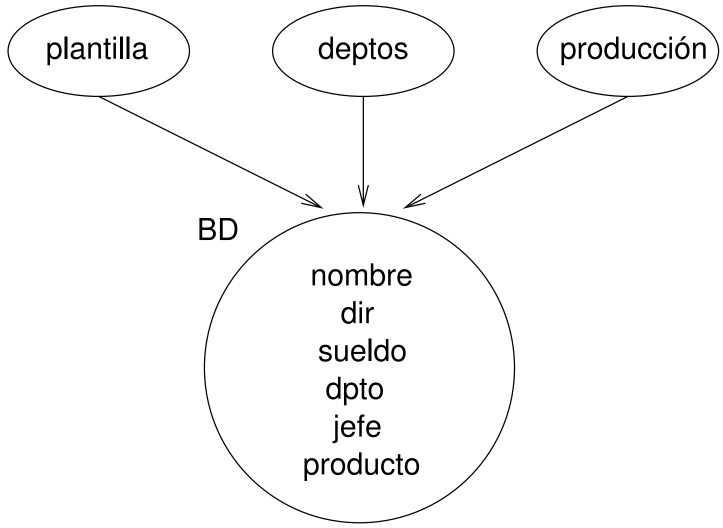


Figura 1.2: Sistema de información orientado a los datos

La mutua independencia de datos y tratamientos lleva a que un cambio de estos últimos no imponga un nuevo diseño lógico y/o físico de las bases de datos. Por otra parte, la inclusión de nuevas informaciones, desaparición de otras, cambios en la estructura física o en los caminos de acceso, etc., no deben obligar a alterar los programas.

- Coherencia de los datos.

  Debido a que la información de la base de datos se recoge y almacena una sola vez, en todos los tratamientos se utilizan los mismos datos, por lo que los resultados de todos ellos son coherentes y perfectamentes comparables.

- Mejor disponibilidad de los datos para el conjunto de los usuarios.

  Cuando se aplica la metodología de bases de datos, cada aplicación/usuario ya no es *propietario* de los datos, puesto que éstos se comparten entre el conjunto de aplicaciones, existiendo una mejor disponibilidad de los datos para todos los que tienen necesidad de ellos, siempre que estén autorizados para su acceso.

  Hay también una mayor *transparencia* respecto a la informacion existente, ya que todos los datos que se encuentran en la base se deben relacionar en un *catálogo* o diccionario, que puede ser ampliamente difundido y accedido por medios informáticos.

- Mayor valor informativo.

Puesto que la base de datos es un sistema reflejo del mundo real, donde los distintos elementos están relacionados, el valor informativo de su conjunto es superior a la suma del valor informativo de los elementos individuales que lo constituyen.

- Mejor y más normalizada documentación de la información, la cual está integrada con los datos.

  Este problema se atenúa en gran medida en las bases de datos, ya que en la misma base se incluyen no sólo los datos, sino también la semántica de los mismos.

- Mayor eficiencia en la recogida, validación y entrada de los datos al sistema.

  Al no existir apenas redundancias, los datos se recogen y validan una sóla vez, aumentando así el rendimiento de todo el proceso previo al almacenamiento.

- Reducción del espacio de almacenamiento.

  La desaparición (o disminución) de las redundancias, así como la aplicación de técnicas de compactacion, lleva en los sistemas de bases de datos a una menor ocupación de almacenamiento secundario. Se ha de tener presente, sin embargo, que los elementos del sistema (diccionarios, referencias, punteros, ficheros invertidos, etc.) ocupan bastante espacio.

1.3 Inconvenientes de las Bases de Datos

Como hemos señalado anteriormente, las bases de datos no sólo presentan ventajas, sino que también tienen posibles inconvenientes. Entre estos inconvenientes es preciso destacar:

- Instalación costosa.
- Personal especializado.
- Implantación larga y difícil.
- Falta de rentabilidad a corto plazo.
- Ausencia real de normas.
- Desfase entre teoría y práctica.

1.4 Concepto de Bases de Datos

La aparición de la expresión *base de datos* se produce a comienzos de los años setenta. En 1963 tuvo lugar en Santa Mónica (USA) un simposio en cuyo título se encontraba la expresión *Data Base.*

Nosotros vamos a empezar estudiando las características esenciales de una base de datos, que en algunos casos coinciden con las ventajas expuestas en el epígrafe anterior, para apoyar en ellas el concepto de base de datos.

En primer lugar, y en esto coinciden todas las definiciones, una base de datos es un *conjunto*, *colección* o *depósito de datos* almacenados en un soporte informático de acceso directo. Los datos deben estar *relacionados y estructurados* de acuerdo con un modelo capaz de recoger el máximo contenido semántico.

Dada la importancia que tienen en el mundo real las relaciones entre los datos, es imprescindible que la base de datos sea capaz de almacenar estas interrelaciones, al igual que hace con otros elementos (como las entidades y atributos), siendo ésta una diferencia esencial respecto a los ficheros donde no se almacenan las relaciones. En el mundo real existen, además, restricciones semánticas, a las que se está concediendo una importancia creciente y que, en los sistemas actuales, tienden a almacenarse junto con los datos, al igual que ocurre con las relaciones.

En las bases de datos no debe existir *redundancia lógica*, aunque sí se admite cierta *redundancia física* por motivos de eficiencia. Desde el punto de vista del usuario, los datos sólo están almacenados una vez, aunque el sistema los duplique para facilitar su acceso. Por tanto, un dato se atualizará *lógicamente* por el usuario de forma única, el sistema se preocupará de cambiar físicamente todos aquellos campos en los que el datos estuviese repetido, en caso de existir redundancia física.

Las bases de datos han de atender a *múltiples usuarios* y a *diferentes aplicaciones.*

Otro aspecto importante de las bases de datos es la *independencia*, tanto física como lógica, entre datos y tratamientos. Esta independencia, objetivo fundamental de las bases de datos es una característica esencial que distingue a las bases de datos de los ficheros y que ha tenido una enorme influencia en la arquitectura de los SGBD, como se verá más adelante.

La *definición* y la *descripción* del conjunto de datos contenidos en la base deben ser *únicas* y estar *integradas* con los mismos datos. En los sistemas basados en ficheros, los datos se encuentran almacenados en soporte magnético, mientras su descripción está separada de los mismos formando parte de los programas. En las bases de datos, la descripción y, en algunos casos, también una definición y documentación completas (metadatos) se almacenan junto con los datos, de modo que éstos estén *autodocumentados*, y cualquier cambio que se produzca en dicha documentación se ha de reflejar y quedará recogido en el sistema, con todas las ventajas que de este hecho se derivan.

La actualización y recuperación en las bases de datos debe realizarse mediante *procesos bien determinados*, incluidos en el SGBD; procedimientos que han de estar diseñados de modo que se mantenga la *integridad*, *seguridad* y *confidencialidad* de la base de datos.

El concepto de base de datos ha ido cambiando y configurándose a lo largo del tiempo, en la actualidad, y de acuerdo con estas características que acabamos de analizar, podemos definir la base de datos como sigue:

> "Colección o depósito de datos integrados, con redundancia controlada y con una estructura que refleje las relaciones y resticciones existentes en el mundo real; los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de éstas, y su definición y descripción, únicas para cada tipo de datos, han de estar almacenadas junto con los mismos. Los procedimientos de actualización y recuperación, comunes y bien determinados, habrán de ser capaces de conservar la integridad, seguridad y confidencialidad del conjunto de los datos"

1.5 Distintos Niveles de Abstracción en una Base de Datos

Se puede observar en los sistemas de información la existencia de dos estructuras distintas, la lógica (vista del usuario) y la física (forma en que se encuentran los datos en el almacenamiento). En las bases de datos aparece un nuevo nivel de abstracción que se ha denominado de diferentes maneras: nivel conceptual, estructura lógico global, etc. Esta estructura intermedia pretende ser una representación global de los datos que se interponga entre las estructuras lógica y física, y que sea independiente, tanto del equipo como del usuario en particular (ver Figura 1.3).

- El esquema externo es la visión que cada usuario particular tiene de la base de datos,
- El esquema conceptual responde al enfoque del conjunto de la realidad representada en la base de datos (o visión del administrador), y
- El esquema interno es la forma en que se organizan los datos en el almacenamiento físico.

Estructura Lógica de Usuario: esquema externo. En él deberán encontrarse sólo aquellos datos y relaciones que necesite cada usuario. También habrán de especificarse las restricciones de uso (ej. el derecho a insertar

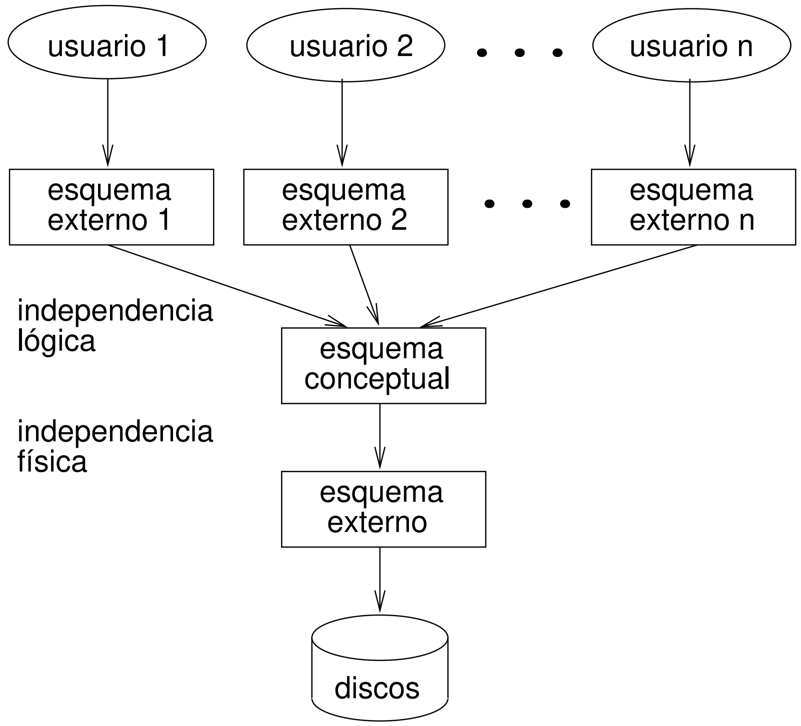


Figura 1.3: Los tres niveles de abstracción de datos.

o a borrar determinados datos o el acceso a los mismo, etc.). Habrá tantos esquemas externos como exijan las diferentes aplicaciones. Un mismo esquema externo podrá ser utilizado por varias aplicaciones.

Estructura Lógica Global: esquema conceptual. En el esquema debe incluirse la descripción de todos los datos y relaciones entre éstos, las restricciones de integridad y confidencialidad

Estructura Física: esquema interno. Aunque el contenido del esquema interno depende de cada SGBD, se pueden distinguir tres clases de aspectos a especificar dentro de él:

- Estrategia de almacenamiento.
- Caminos de acceso.
- Miscelanea (encriptación, compresión, etc.)

Uno de los principales objetivos de las bases de datos es conseguir la independencia entre las estructuras lógica y física de los datos, que tiene como consecuencia la independencia entre datos y aplicaciones, de modo que los cambios en la estructura de los datos tenga una repercusión mínima en los programas de aplicación, y viceversa (Figura 1.3).

La flexibilidad y versatilidad de que ha de estar dotado todo sistema de información es muy difícil, sino imposible, de alcanzar con la orientación hacia ficheros propia de los primitivos sistemas de gestión de datos.

El concepto de independencia de datos implica la separación entre el almacenamiento y la organización lógica de los datos tal como éstos son contemplados por los distintos programas de aplicación que hacen uso de la base de datos, con lo que se consigue una doble finalidad:

- Unos mismos datos se presentarán de distintas formas, según las necesidades de los usuarios.

- El almacenamiento de los datos, su estructura lógica y los programas de aplicación serán independientes unos de otros, de modo que un cambio en uno de ellos no obligue a modificar los demás.

Este objetivo de independencia, aunque no alcanzado plenamente, ha tenido una fuerte repercusión en la arquitectura de los sistemas de bases de datos.

En algunas ocasiones se enfoca la independencia en el sentido de la estandarización y los sistemas abiertos, lo que permitiría que una vez desarrollado un sistema e implantado en un determinado SGBD, el cambio de éste a otro producto comercial no implicaría tener que diseñar de nuevo la base de datos ni tampoco tener que reescribir los programas que acceden a los mismos.

Capítulo 2

El Sistema de Gestión de la Base de Datos

Una vez examinado el concepto de base de datos, analizaremos el *sistema de gestión de base de datos*, SGBD, o *Data Base Management System*, DBMS, junto con sus funciones, lenguajes, facilidades, interfaces, etc.

2.1 El SGBD como Interfaz entre el Usuario y la Base de Datos

La base de datos, como único depósito de datos, debe ser capaz de integrar las distintas aplicaciones que trabajan con los datos, siendo el SGBD el encargado de suministrar el interfaz entre el usuario y los datos.

Ademas de estos usuario finales, existen otros tipos de usuarios, analistas y programadores, cuyas exigencias son muy diferentes y también han de ser tenidas en cuenta por el SGBD. El administrador de la base de datos también necesitará que el SGBD le facilite sus tareas, Figura 2.1.

2.2 Concepto y Principales Funciones del SGBD

Se puede definir el sistema de gestión de la base de datos, SGBD, como *un conjunto coordinado de programas, procedimientos, lenguajes, etc. que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la base, manteniendo su integridad, confidencialidad y seguridad.*

Las operaciones típicas que debe realizar un SGBD pueden clasificarse en aquellas que afecten a la totalidad de los datos o a todos los registros de un determinado tipo y las que tienen lugar sobre registros concretos.

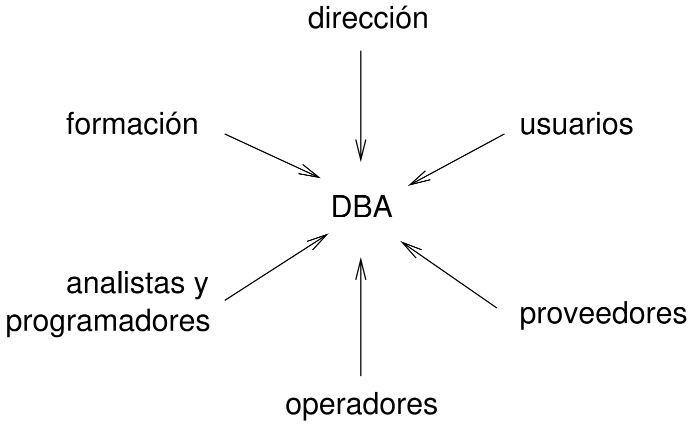


Figura 2.1: El administrador de la BD.

Las funciones esenciales de un SGBD son las de descripción, manipulación y utilización.

- **Descripción o Definición**

  Permite al administrador de una base de datos especificar los elementos de datos que la integran, su estructura y las relaciones que existen entre ellos, las reglas de integridad, los controles de acceso y las características de tipo físico.

  Todo esto se realizará con la ayuda del *lenguaje de definición de datos*, *Data Definition Language* DDL.

- **Manipulación**

  Permite a los usuarios de la base de datos, técnicos o no, buscar, añadir, suprimir o modificar los datos de la misma, siempre de acuerdo con las especificaciones y las normas de seguridad dictadas por el administrador.

  Para realizar estas operaciones se cuenta con el *lenguaje de manipulación de datos*, *Data Manipulation Language* DML. Muchas veces se trata de un conjunto de instrucciones, *lenguaje huesped*, que son admitidas por un lenguaje de programación, *lenguaje anfitrión*, mientras que otras veces se trata de un *lenguaje autocontenido* que no precisa apoyarse en ningún otro lenguaje.

- **Utilización**

  Reune todos los interfaces que necesitan los diferentes usuarios para comunicarse con la base de datos y proporciona un conjunto de procedimientos para el administrador.

En la mayoría de los SGBD existen funciones de servicio, como la obtención de estadísticas de utilización, cargas de archivos, copias de seguridad, rearranque en caso de caída del sistema, protección frente a accesos no autorizados, etc.

2.3 Lenguajes de los SGBD

Las distintas funciones que ha de cumplir un SGBD hacen necesario disponer de diferentes lenguajes y procedimientos que permitan la comunicación con la base de datos.

Además, dado el conjunto de potenciales usuarios de la base de datos y las diferentes necesidades de cada uno se tendrá que disponer de una variedad de lenguajes distintos, no sólo en la forma de manejarse, sino en las posibilidades de cada uno de ellos.

- **Lenguajes de Definición de Datos**

  Deben permitir al administrador de la base de datos definir los datos con facilidad y precisión, especificando sus distintas estructuras.

- **Lenguajes para Manipulación de Datos**

  Darán a los usuarios la posibilidad de referirse a determinados conjuntos de datos que cumplan ciertas condiciones, *criterios de selección*, como que un atributo tenga un determinado valor.

2.4 Otras Facilidades Proporcionadas por los SGBD

Además de las facilidades suministradas por los lenguajes, los distintos SGBD proporcionan otros medios suplementarios para simplificar al administrador las tareas de mantenimiento y salvaguarda de la base, y ayudar a los distintos usuarios a obtener el máximo provecho de los datos contenidos en la misma.

2.5 Interrelación del SGBD con el Sistema Operativo

El funcionamiento del SGBD está muy relacionado con el Sistema Operativo. Cuando, utilizando un DML, se accede a un dato en concreto, el SGBD está accediendo a un fichero dado, utilizando para ello las facilidades que le aporta el Sistema Operativo sobre el que se ejecuta.

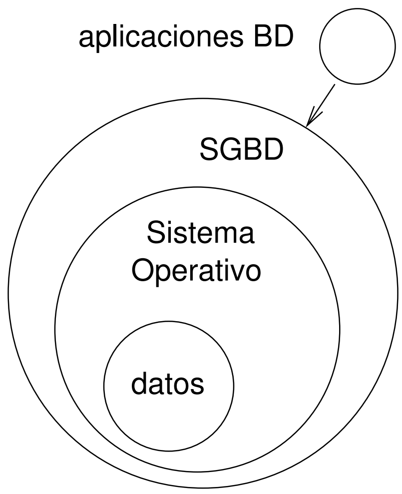


Figura 2.2: Interrelación del SGBD con el sistema operativo.

Capítulo 3

Modelo de Datos Entidad/Interrelación, ER

3.1 Concepto de Modelos de Datos

Llamamos *modelo* al instrumento que se aplica a una parcela del mundo real para obtener una estructura de datos a la que denominamos *esquema*.

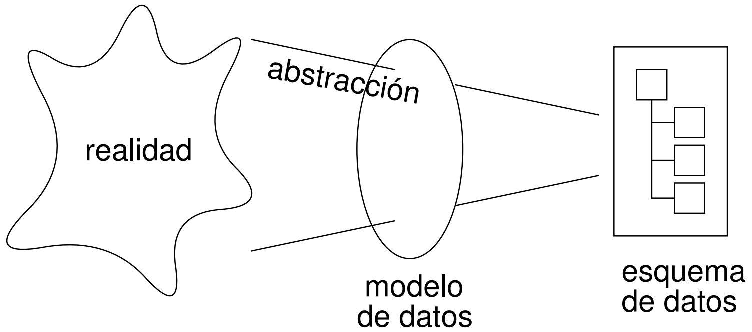


Figura 3.1: Proceso de abstracción y aplicación de un modelo de datos.

Los objetivos que persigue todo modelo de datos son:

- *Formalización*, ya que el modelo de datos permite definir formalmente las estructuras permitidas y las resticciones a fin de representar los datos de un Sistema de Información, y también porque el modelo de datos establece la base para la definición de un lenguaje de datos.

- *Diseño*, ya que el modelo de datos es uno de los elementos básicos para el desarrollo de una metodología de diseño de BD, permitiendo, además, prever el impacto de los cambios en un Sistema de Información.

Un modelo de datos se compone de dos partes: una *estática*, o relativamente invariante en el tiempo, que responde a lo que se suele entender como estructuras, y otra *dinámica*, que varía en el transcurso del tiempo, son los datos o valores que se almacenan en las estructuras.

3.2 Presentación del Modelos ER

El modelo ER está centrado en dos elementos fundamentales: el de *entidad* y el de *interrelación*[1]. Se entiende por entidad cualquier objeto sobre el cual queremos tener información en la BD y que tiene existencia por sí mismo. Relación es la asociación o correspondencia entre entidades.

3.3 Estática del Modelo ER

En el modelo ER se puede distinguir como elementos fundamentales las entidades, los atributos y las relaciones, además del conjunto de valores o dominio.

3.3.1 Entidad

Se puede definir la entidad como aquel objeto (real o abstracto) acerca del cual queremos almacenar información en la BD. Se puede distingir un *tipo de entidad* de cada una de sus ocurrencias o *entidades*.

La representación de este tipo de objeto es un rectángulo etiquetado con el nombre del tipo de entidad, como podemos ver en la Figura 3.2.

Figura 3.2: Representación de tipos de entidad.

Existen dos clases de entidades: *regulares*, que son alquellas que tienen existencia por sí mismas (como Libro y Autor), y *débiles*, cuya existencia depende de otro tipo de entidad(Familiar depende de Empleado, y la desaparición de un empleado de la BD hace que desaparezcan también todos los familiares que estaban a su cargo). Los tipos de entidad débil se representan con dos rectángulos concéntricos, con su nombre en el interior (Figura 3.3).

[1]En adelante *relacion*, aunque pueda confundirse con las relaciones del modelo Relacional que se verá más adelante, es fácil diferenciarlas fijándose en el contexto.

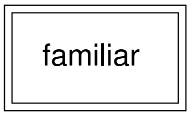


Figura 3.3: Representación de tipos de entidad débiles.

3.3.2 Relación

Se entiende por *relación* aquella asociación o correspondencia existente entre entidades. Denominamos *tipo de relación* a la estructura genérica del conjunto de relaciones existentes entre dos o más tipos de entidad.

Se denomina *grado* del tipo de relación al número de tipos de entidades que participan en él.

Representaremos el tipo de relación mediante un rombo etiquetado con el nombre de la relación, unido mediante arcos a los tipos de entidades que asocia (Figura 3.4).

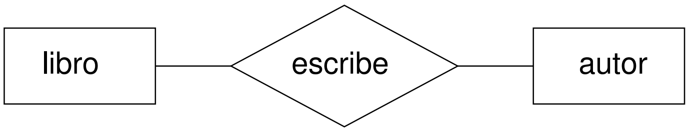


Figura 3.4: Representación de la relación escribe entre entidades libro y autor.

Entre dos tipos de entidad puede existir más de un tipo de relación (Figura 3.5).

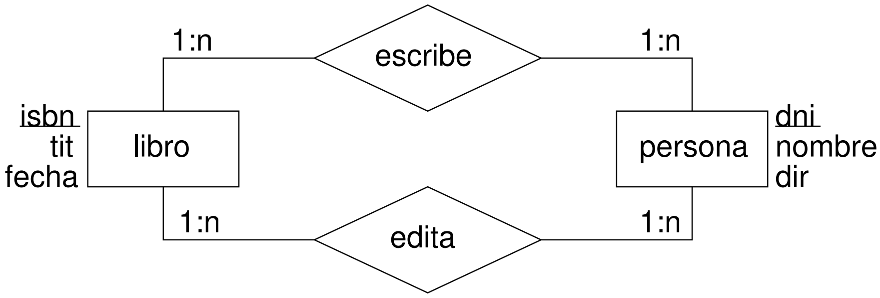


Figura 3.5: Varias relaciones entre entidades libro y persona.

Elementos de un Tipo de Relación

En un tipo de relación existen los siguientes elementos:

- **Nombre**: que lo distingue del resto de los elementos del modelo ER.
- **Grado**: el número de tipos de entidad que participan en el tipo de relación. Por ejemplo, la relación escribe de la Figura 3.4 es de grado 2, mientras que la relación saluda de la Figura 3.6 relaciona a una entidad de tipo persona con otra entidad del mismo tipo, grado 1 o *reflexiva*.

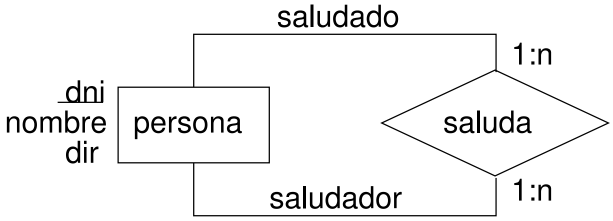


Figura 3.6: Relación de grado 1.

- **Cardinalidad**: el número mínimo y máximo de ocurrencias de cada tipo de entidad que pueden intervenir en el tipo de relación tratada. Para representarlo gráficamente se pone la etiqueta con $1:1$, $1:n$ o $n:m$, según corresponda a ocurrencias uno a uno, uno a muchos, o muchos a muchos, respectivamente (Figura 3.7).

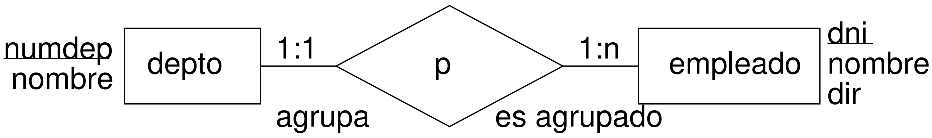


Figura 3.7: Elementos de una relación.

- **Papel** o *rol*: la función que cada uno de los tipos de entidad realiza en el tipo de relación. El nombre del papel se suele colocar junto al arco que une cada tipo de entidad con el tipo de relación, (Figura 3.7).

3.3.3 Atributo, Dominio, Valor, Clave

Un tipo de entidad o de relación tiene distintas propiedades o características, *atributos*, que pueden tomar diferentes *valores* para cada ocurrencia de un tipo de entidad o relación. El conjunto de todos los posibles valores que puede tomar una cierta característica se denomina *dominio*.

Debemos tener alguna manera de distinguir una entidad o relación dada entre todas las entidades o relaciones pertenecientes a un tipo concreto de entidad o relación. Desde el punto del esquema conceptual esta distinción se hace en función de sus atributos.

Entre todos los atributos de un tipo de entidad debemos elegir uno o varios que identifiquen unívocamente cada una de las ocurrencias de ese tipo de entidad. Ese atributo o conjunto de atributo se denomina *clave principal* o *clave primaria*, y los atributos que lo componen deben ser mínimos en el sentido de que la eliminación de cualquiera de ellos le haría perder su carácter identificador. Puede ocurrir que exista más de un conjunto de atributos que sean un identificador unívoco y mínimo para un tipo de entidad, por lo que denominaremos a cada uno de ellos como una *clave candidata*. De todos modos, elegiremos sólo a uno de ellos como clave principal.

Una representación gráfica de un atributo ámpliamente extendida consiste en un círculo u óvalo etiquetado con el nombre del atributo. Pero en un esfuerzo para conseguir gráficos de un tamaño contenido vamos a representar los atributos alineados al lado del tipo de entidad o de relación a los que afectan, subrayando los atributos que formen la clave principal.

Las relaciones también pueden tener atributos asociados. Éstos atributos propios de las relaciones se denominan *atributos locales* a la relación. Estos atributos suelen corresponder con informaciones dependientes de todos los tipos de entidad que relaciona el tipo de relación.

3.4 Control de Redundancias en los Esquemas ER

Una vez construido el esquema ER, hay que analizar si se presentan redundancias, ya que pueden acarrear problemas, como se ha visto en los capítulos precedentes.

Una primera señal que puede indicar la existencia de redundancias son los ciclos en los diagramas ER, ya que pueden estar formados por relaciones redundantes.

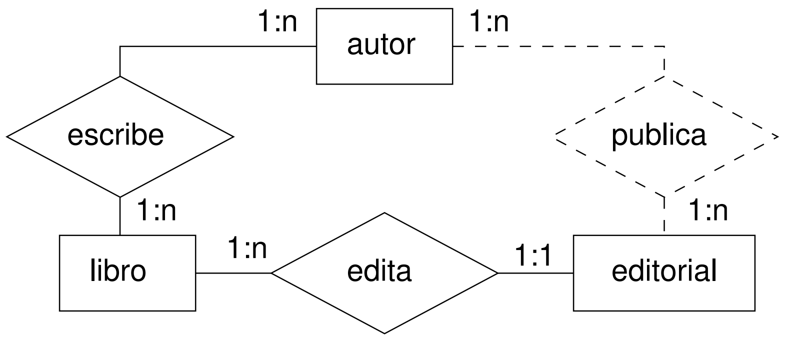


Figura 3.8: Ciclo donde aparece una relación redundante.

En el ejemplo de la Figura 3.8 se muestra el diagrama ER correspondiente al caso de que los autores escriben libros que son publicados por editoriales, y que los escritores publican en ciertas editoriales. En dicho diagrama ER se da un ciclo entre autor, libro y editorial, por lo que en principio es posible que aparezca alguna relación redundante. Y así es. Si se conocen los libros de un autor y las editoriales que los han editado, se puede deducir fácilmente en qué editoriales ha publicado dicho autor; de la misma manera, dada una editorial, si sabemos qué libros ha publicado, podemos deducir fácilmente qué autores han escrito para ella, por lo que la relación publica entre las entidades autor y editorial es redundante, lo que se muestra en el ejemplo mediante un trazo punteado.

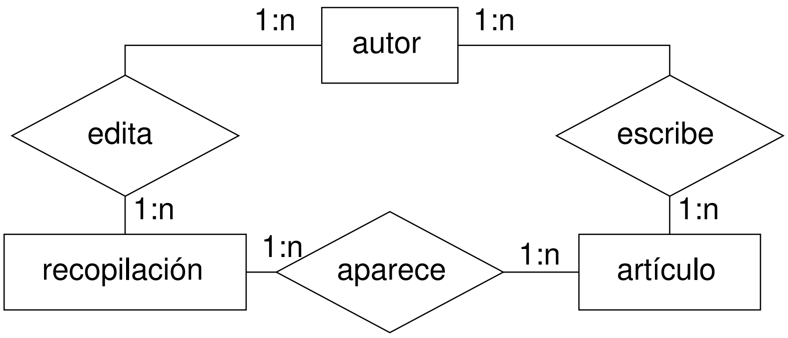


Figura 3.9: Ciclo donde no aparece una relación redundante.

Sin embargo, en la Figura 3.9, a pesar de existir un ciclo, no hay ninguna relación redundante. En dicha figura se muestra el caso de la elaboración de recopilaciones de artículos, en los que interesa conocer los autores que editan las recopilaciones; los articulos que escribe cada autor; y los artículos que forman cada recopilación. En lo que respecta a la relación escribe, no puede deducirse de las otras dos, ya que aunque sepamos las recopilaciones que ha editado un autor y los artículos aparecidos en ellas, no podemos saber los artículos que ha escrito dicho autor. La relación edita tampoco es redundante, ya que un autor escribe varios artículos que pueden aparecer en recopilaciones sin que eso implique que dicho autor edite esas recopilaciones. Por último, la relación aparece tampoco es redundante, ya que un artículo escrito por un autor no tiene porqué aparecer necesariamente en las recopilaciones que éste edite: puede aparecer en otras recopilaciones o incluso en revistas.

Existe, además, otros casos en los que la relación, a pesar de poder ser deducida a partir de otras presentes en el esquema, no se puede eliminar porque posee atributos.

Se puede decir como norma general, que la existencia de un ciclo no implica la existencia de relaciones redundantes. Habrá que analizar si al eliminar una relación es siempre posible el paso, tanto en un sentido como en el inverso, entre las dos entidades que relaciona, y habrá que comprobar también que no se pierden atributos al eliminar dicha relación.

3.5 Semántica de las Relaciones

El contenido semántico de las relaciones se ha completado con conceptos tales como las cardinalidades, la dependencia en existencia y la abstracción de generalización, que pasamos a estudiar a continuación.

3.5.1 Cardinalidades de un Tipo de Entidad

Se define como el número mínimo y máximo de ocurrencias de un tipo de entidad que pueden estar interrelacionadas con una ocurrencia del otro, u otros tipos de entidad que participan en el tipo de relación. Su representación gráfica es una etiqueta del tipo $1:1$, $1:n$, o $n:m$, según corresponda.

Para calcular la cardinalidad de un tipo de entidad en un tipo de relación hay que hacerse la pregunta siguiente: suponiendo que en la relación participa una entidad de cada uno de los otros tipos de entidad, ¿cuántas entidades van a participar, como mínimo y como máximo, de este tipo de entidad?

3.5.2 Dependencia en Existencia y en Identificación

Como en el caso de los tipos de entidad, los tipos de relación se clasifican tambén en *regulares* y en *débiles*, según estén asociando dos tipos de entidad regulares, o un tipo de entidad débil con un tipo de entidad regular, respectivamente.

Parece interesante distinguir dentro del tipo de relación débil la dependencia en *existencia* y la dependencia en *identificación*.

- Dependencia en existencia: cuando las ocurrencias de un tipo de entidad, entidad débil, no pueden existir si desaparece la ocurrencia de la entidad regular de la cual dependen (Figura 3.10).

- Dependencia en identificación: cuando además de cumplirse la condición anterior, las ocurrencias del tipo de entidad débil no se pueden identificar únicamente mediante los atributos propios de la misma y exigen añadir la clave del tipo de entidad regular del cual dependen (Figura 3.11).

Si la dependencia es en identificación, el rombo que representa la relación va etiquetado con **id**, y con una **e** (o sin etiqueta) en caso de que la dependencia sea sólo en existencia.

En la Figura 3.10 se puede observar que los datos acerca de los familiares de un empleado sólo tendrán sentido si éste permanece en la base de datos, con lo que hay una dependencia en existencia. Por otro lado, en la Figura 3.11 tenemos que un ejemplar está identificado con la clave del libro del cual depende más un código própio, habiendo, por tanto, una dependencia en identificacion.

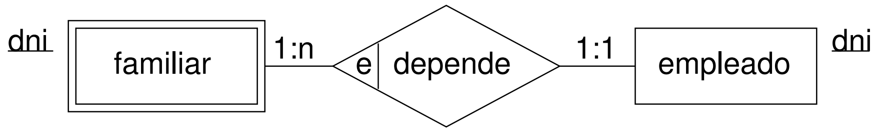


Figura 3.10: Dependencia en existencia.

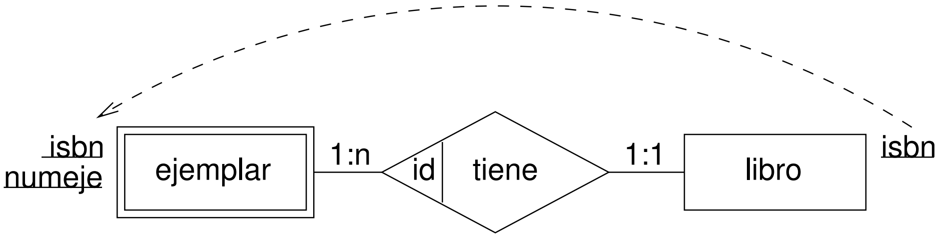


Figura 3.11: Dependencia en identificacion.

3.5.3 Relaciones Exclusivas

Decimos que dos o más tipos de relación son exclusivos cuando cada ocurrencia de un tipo de entidad sólo puede pertenecer a un tipo de relación. Por ejemplo, en la Figura 3.12 se observan las relaciones Publica y Aparece como exclusivas, ya que se ha supuesto que en una determinada biblioteca, los artículos se publican en revistas o en recopilaciones, pero no en ambos.

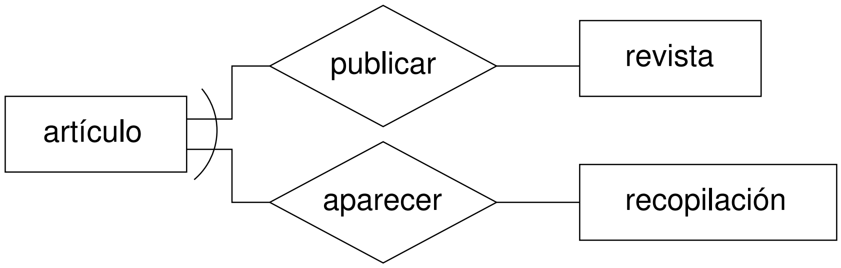


Figura 3.12: Tipo de relación exclusiva.

3.6 Generalización y Herencia

La descomposición de tipos de entidad en varios subtipos es una necesidad muy habitual en el modelado de bases de datos. En el mundo real se pueden identificar varias *jerarquias* de entidades, en las que se plasma la idea de “es un”, (*is a* en inglés).

En el modelo ER se pueden representar este tipo de relación, utilizando para ello un triangulo con el vértice apuntando al tipo de entidad más general, o *supertipo*, y conectado por la base a los tipos de entidad más particulares, o *subtipos*, (Figura 3.13).

Esta relación tiene la característica de que toda ocurrencia de un subtipo es una ocurrencia del supertipo, aunque no sucede lo contrario, con lo que las cardinalidades serán siempre $(1,1)$ en el supertipo y $(0,1)$ o $(1,1)$ en los subtipos.

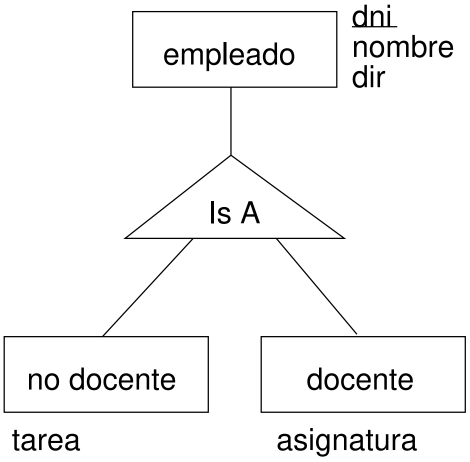


Figura 3.13: Relación IS-A.

Otra característica muy importante de esta clase de interrelaciones es la herencia, ya que en principio todo atributo del supertipo pasa a ser un atributo de los subtipos; así, por ejemplo podemos establecer una asociación de este tipo entre la entidad `empleado` y las entidades `docente` y `no docente`, en el sentido de que tanto los docentes como los no docentes *son* empleados, por lo que heredarán todas las características de la entidad empleado: `dni`, `nombre` y `dir`.

En este tipo de abstracción los atributos comunes a todos los subtipos se asignan al supertipo, mientras que los atributos específicos se asocian al subtipo correspondiente. Del mismo modo, las relaciones que afectan a todos los subtipos se asocian al supertipo, dejándose para los subtipos las relaciones específicas para cada uno de ellos.

La división en subtipos (*especialización*) puede venir determinada por una condición predefinida (por ejemplo, en función de los valores de un atributo), en cuyo caso se representará la condición (o el atributo *discriminate*) asociada al triángulo que representa la relación (Figura 3.14).

Se pueden distinguir cuatro clases de generalización, atendiendo a si los subtipos se solapan o son disjuntos, y a si la unión de los subtipos recubre o no el supertipo. Representaremos por un arco el hecho de que los subtipos sean disjuntos y con un círculo la presencia de una jerarquía total (los subtipos recubren el supertipo).

Puede suceder también que un subtipo tenga más de un supertipo, formandose una *red* de generalización. En este caso la herencia se convierte en múltiple, pudiéndose presentar conflictos a la hora de heredar atributos iguales de dos supertipos distintos. En este caso se podrá optar por renombrar los atributos de los supertipos que entren en conflicto o por inhibir la herencia de dichos atributos.

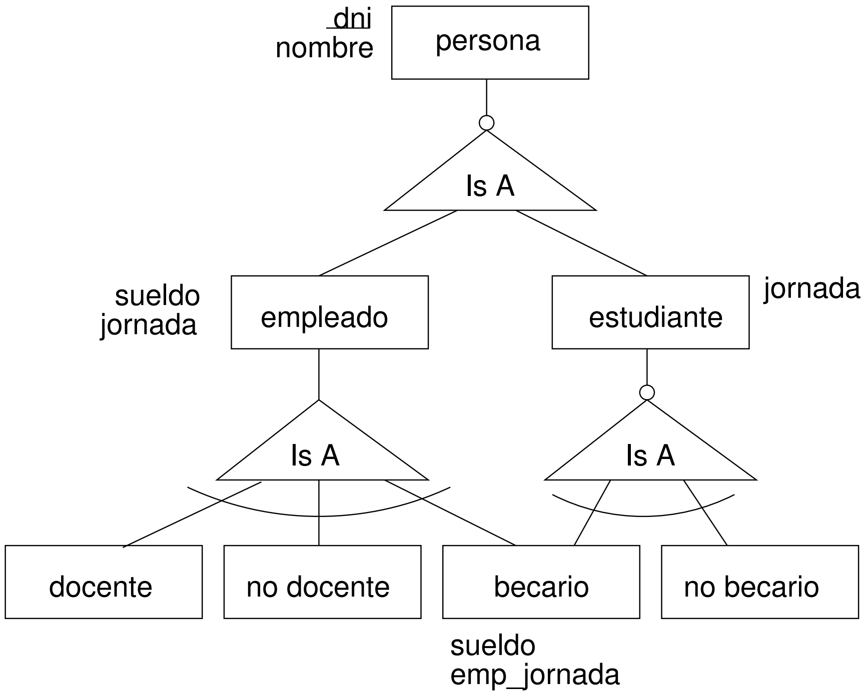


Figura 3.14: Ejemplo de red de generalización.

En la Figura 3.14 se muestra un ejemplo de red de generalización donde se tiene un supertipo persona que posee como atributos un código y el nombre (que serán heredados por todos los demás subtipos de entidad), que tiene dos subtipos empleado y estudiante. Ambos tienen como atributo jornada (laboral o de estudio), que, al tratarse de atributos distintos, hemos definido en cada tipo de entidad. Además, la entidad empleado tiene como atributo el sueldo, que heredarán tanto los docentes como los no docentes y los becarios. En este último tipo de entidad se presenta un conflicto en la herencia del atributo jornada, por lo que tenemos que especificar qué jornada hereda el tipo de entidad becario o si hereda las dos. En el ejemplo se ha supuesto que sólo hereda como jornada la que proviene de la entidad empleado.

3.7 Ejercicios

1. Considerar las siguientes características de la compañía aérea *Ala2*, e intentar proponer un diagrama ER (ver Figura 3.15).

   La compañía trata con pasajeros que son registrados en diferentes vuelos en una clase y en un asiento determinados. De los pasajeros interesa el nombre, la dirección y el número de teléfono. De los vuelos es necesario almacenar el número de vuelo, la fecha, el origen, el destino, la hora de salida y la de llegada.

   La compañía dispone de un conjunto de empleados de los cuales almacena información referente al número de empleado, el nombre, dirección y el salario que recibe cada uno, Una parte de estos empleados son pilotos y sólo ellos pueden pilotar aparatos.

   La companía dispone también de aviones de diferentes tipos. Para cada tipo de aparato necesita registrar el constructor, el número de modelo, el número de asientos que posee y la velocidad de crucero que alcanza. A los aviones se les asigna un número de serie e interesa conocer el número de horas de vuelo que ham cumplido cada uno. Cada vuelo tiene asignado un avión concreto y varios empleados.

2. Consideremos una base de datos que registra los jugadores de fútbol, los equipos para los que han jugado, su promedio de goles por temporada y las posiciones en que han jugado. Las entidades son:

   - jugadores con atributos nombre, lugarNacimiento y fechaNacimiento.
   - posiciones con atributos nombre, número (cualquiera podría servir para clave).
   - equipos con atributos denominación, ciudad, año.
   - pg, el conjunto de promedio de goles. Este conjunto entidad tiene un único atributo numérico.

   Tendremos las siguientes relaciones: temporada entre jugadores, equipos y pg. El jugador j, el equipo e y el promedio de goles g están relacionados por temporada si j jugó en el equipo e y su promedio de goles fue g. Esta es una relación ternaria. juega entre jugadores y posiciones. Es varios a varios, indicando las posiciones en que ha jugado un jugador a lo largo de su carrera. Dibujar el diagrama ER asociada a esta descripción.

3. Supongamos que definimos informalmente los datos de una base de datos de un almacén como sigue:

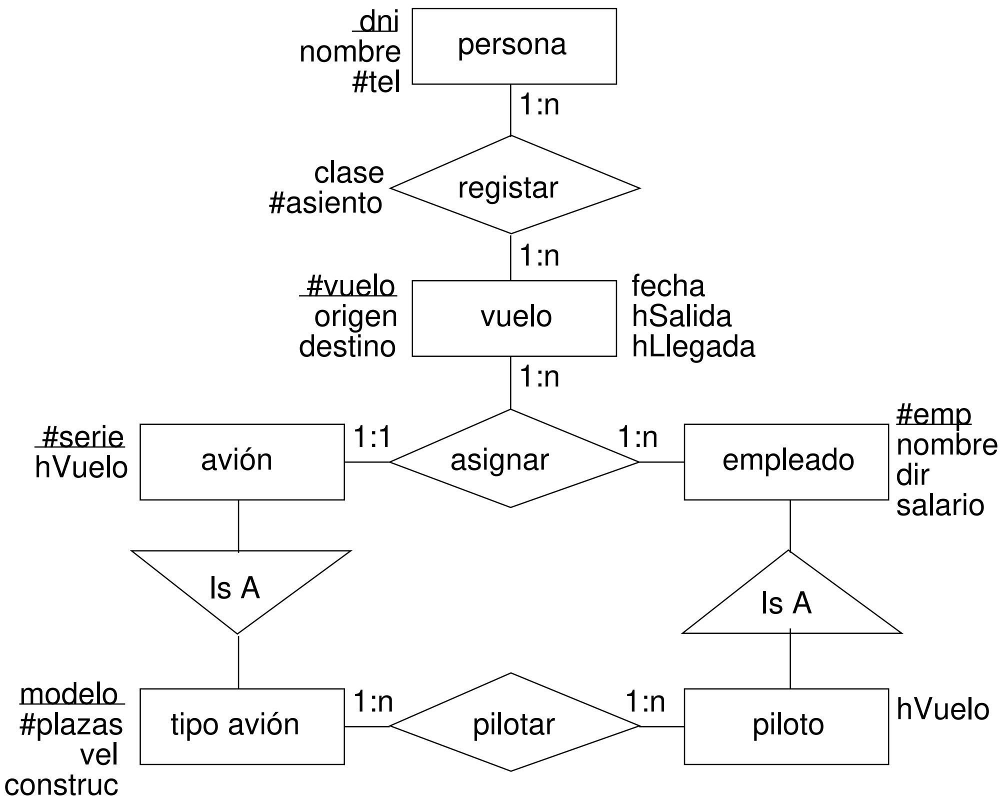


Figura 3.15: Compania aerea Ala2.

- Cada empleado está representado. Los datos a considerar de un empleado son su número de empleado, nombre, dirección y el departamento para el que trabaja.
- Cada departamento está representado. Los datos a considerar de los departamentos son su nombre, empleados, director, y productos que tiene asignados.
- Cada producto con que se trabaja está representado. Los datos a considerar de los productos son su nombre, fabricante, precio, número de modelo asignado por el fabricante y número de producto interno asignado por el almacén.
- Cada fabricante se representa. Los datos son su nombre, dirección, productos que suministra al almacén y su precio.

Construir un diagrama ER para esta base de datos. (Nótese que alguna información puede ser representada por atributos y otra por relaciones).

4. Interesa mantener la siguiente información acerca de una Escuela Universitaria:

   - Sobre los alumnos: su número de expediente, nombre, dirección, asignaturas en que se haya matriculado y su calificación por asignaturas.
   - Sobre cada asignatura que se imparte: código de la asignatura, nombre, sección, especialidad, profesores que la imparten, carácter y número de créditos.
   - Sobre los departamentos: nombre, extensión telefónica, profesores adscritos y director del departamento. El director es también un profesor.
   - Sobre cada profesor: nombre, dirección, teléfono, departamento al que pertenece y asignaturas que imparte.
   - El horario de cada profesor: día, hora y asignatura para cada sesión docente.

   Se pide construir el diagrama ER que modele convenientemente esta información.

5. Hacer un diagrama ER para una base de datos que muestre las relaciones entre hombres y mujeres de padre, madre y cónyuge.

6. Nos interesa modelizar la siguiente información histórica sobre los equipos de la liga de béisbol americano:

   - Cada equipo que se localiza en una determinada ciudad tiene un número único, un nombre, un presidente y un conjunto de entrenadores actuales de los que se desea mantener su nombre, su número de teléfono y la experiencia profesional que cuentan en su haber. Nos basta con los años de experiencia de cada cargo de entrenador que haya desempeñado. Por ejemplo, Adams fue entrenador de Universidad cinco años y entrenador de liga japonesa dos años.
   - A lo largo de la historia del equipo han pasado por sus filas una serie de jugadores. Además del nombre del jugador, su edad actual y un número único que lo distingue, nos interesa mantener una relación de los años que ha figurado en la plantilla de cada equipo por el que haya fichado y el promedio de bateo que obtuvieron en esos años.
   - Por último, también ha de figurar el modelo y el fabricante de los bates que ha utilizado cada equipo.

   Diseñar el diagrama Entidad-Relación correspondiente.

7. Nos interesa modelar la siguiente información acerca de los libros departamentales de un Centro Universitario:

   - Cada departamento puede adquirir anualmente y a un cierto precio cierto número de ejemplares de un libro que pasan a ser de su propiedad y a los que inmediatamente se les asigna un número de registro general de biblioteca. De cada libro interesa su título, autor, materia, editorial y año de publicación, así como su número de código (ISBN) que lo identifica unívocamente.
   - Los ejemplares pueden ser cogidos en calidad de préstamo (sin fecha límite de devolución) por los profesores del centro pertenezcan o no al departamento propietario del ejemplar.
   - Se guardará información sobre los profesores usuarios potenciales de los ejempla res de algún departamento: su nombre, dirección, teléfono y departamento al que pertenecen.
   - Cada vez que se haya prestado un ejemplar queremos saber a quién, la fecha de retirada y la fecha de devolución.

   Diseñar el diagrama Entidad-Relación correspondiente.

8. Una empresa de reparación de aparatos domésticos mantiene información acerca de sus técnicos: número de empleado, nombre y título/s que posee. Además, recoge información acerca de los tipos de aparato a que da servicio junto con la cuota por hora para reparar cada uno de los aparatos, los aparatos específicos (que se identifican por el número de serie) de los que se han vendido contratos de reparación y qué técnicos están cualificados para dar servicio a qué tipos de aparatos, incluyendo el número de años en que un técnico ha estado cualificado para arreglar un tipo específico de aparato.

   Diseñar el diagrama ER correspondiente.

9. Queremos diseñar una base de datos para almacenar información sobre nuestros Clientes y sus Pedidos. Los artículos comercializados por nuestra organización tienen un código que los identifica unívocamente y una descripción. Algunos son fabricados en factorías propias y otros hay que adquirirlos en proveedores externos.

   Cada cliente dispone de un crédito con un límite de crédito, un descuento por compras masivas y un saldo, cuyo importe va variando conforme va pagando los pedidos entregados. Los pedidos se entregan en una dirección de entre varias disponibles para cada cliente. Son las direcciones de entrega del cliente. Además, un pedido puede servirse en varias entregas, por lo que habrá que llevar un control de lo que queda pendiente de entregar en cada momento. Los pedidos se componen de dos partes: una cabecera de pedido y varias líneas de detalle.

El la cabecera figuran la fecha de pedido y el cliente que lo hace. Las líneas de detalle contienen el artículo pedido y la cantidad pedida.

Se desea llevar, también, un control de las necesidades de fabricación. Para ello, por cada fábrica propia se guarda la cantidad disponible de cada artículo en el almacén de la fábrica y el nivel mínimo de existencias a partir del cual hay riesgo de quedarse sin existencias.

Obtener el diagrama ER de acuerdo con esta descripción, completándola si hace falta con las hipótesis adicionales que se crean razonables.

10. Se desea diseñar una base de datos que contenga la información relativa a todas las carreteras de un determinado país. Se pide realizar el diseño del modelo ER y del modelo relacional, sabiendo que:

    - En dicho país las carreteras se hayan divididas en tramos.
    - Un tramo siempre pertenece a una única carretera y no puede cambiar de carretera.
    - Un tramo puede pasar por varios términos municipales, siendo un dato de interés el km del tramo por el que entra en dicho término municipal y el km por el que sale.
    - Los tramos tienen una determinada densidad de tráfico media (vehículos/h), una velocidad media de los vehículos y un estado del firme.
    - También se ha de mantener información acerca de las características por defecto de la carretera (arcenes, velocidad máxima y mínima y carriles por cada sentido), en función de su categoría (autopista, autovia, nacional, etc).

11. El hotel BD tiene 400 habitaciones. Los tipos de habitaciones que tiene son individuales, dobles y suites; algunas habitaciones tienen teléfono, televisión y/o aire acondicionado. El hotel, a menudo, reserva habitaciones con un año de anticipación. El precio de la habitación depende del tipo de huésped y se considera que hay tres categorías de huésped: normal, comercial y gobierno. Cuando un cliente hace una reserva para una fecha determinada, el conserje le pide los siguientes datos: fecha de inicio de la estancia, nombre y dirección, tipo de la habitación deseada, duración de la estancia, tarifa a la que se puede acoger. Si se confirma la reserva, se abre una factura donde se anotan todos los servicios que el huésped utiliza durante su estancia y que tendrá que abonar junto con su habitación. Cada servicio tiene una única tarifa y queda claramente consignado en la factura. El hotel quiere instalar un sistema que maneje las reservas de las habitaciones y que contemple las siguientes facilidades:

- control del inventario de las habitaciones. El hotel debe conocer en cada momento qué habitaciones están reservadas y cuáles están disponibles.
- mantenimiento y control de facturas. Se debe poder controlar todos los gastos originados por el cliente y facturárselos debidamente.
- control de los clientes habituales. El hotel desea mejorar el trato dado a ese tipo de clientes.

Con la información ofrecida, detállese el diagrama ER para los datos del hotel.

12. Se desea crear una base de datos para almacenar los menús de un hospital.

    - Cada menú tiene un único nombre y fue creado en una fecha determinada por un dietista. Cada menú puede estar formado por cualquier número de platos, cada uno de ellos con un único nombre. El orden de consumo del plato varía para cada menú.
    - Se debe mantener una descripción del plato para cada plato almacenado. Cada plato está hecho con cualquier cantidad de cualquier número de alimentos (por ejemplo, el plato PLATO1 está hecho con 10 gr. de ternera, 20 gr. de patatas y 5 gr. de tomates).
    - Cada alimento está perfectamente identificado y descrito en la base de datos.
    - Para poder obtener el valor nutricional de cada menú, los valores nutrientes se asocian con cada alimento como sigue:
      - Cada nutriente se identifica con una clave, almacenándose la descripción del nutriente y la unidad de medida.
      - Se conoce el contenido de nutriente en una cantidad estándar de alimento, para cada nutriente en cada alimento (por ejemplo, 100 UI de Vitamina A en un litro de leche).
    - A cada paciente del hospital se le asigna un menú cada día de su estancia en el mismo. Sólo se asigna un menú a cada paciente por día, pero un paciente puede tomar diferentes menús en diferentes días. Cada paciente está identificado y se guarda cierta información sobre él.

    Con la información que se ofrece, detallar el esquema Entidad-Relación para este supuesto.

    Modificar el esquema obtenido en el apartado anterior para:

- Permitir que los platos puedan servirse directamente al paciente, llevando cuenta del número de veces que un paciente toma un determinado plato y el día en que lo hace.
- Permitir que los alimentos puedan servirse directamente al paciente, llevando cuenta del número de cantidades estándar que toma del alimento y el día en que lo hace.

13. Se desea estudiar el *Cursus Honorum* de los perros que participan en concursos y su árbol genealógico. Cada perro tiene un nombre y se identifica por un número, aunque varios perros pueden tener el mismo nombre. Interesa conocer la fecha de nacimiento, el sexo y la raza del perro, junto con su DNI. Un perro puede tener varios dueños sucesivos y un único propietario actual. Los concursos de perros se celebran en una ciudad y en unas fechas establecidas, son de algún tipo (belleza, destreza, etc.) y cada convocatoria del concurso se identifica por un número. Un perro puede ser presentado a distintos concursos, obteniendo un resultado en cada uno de ellos. Con respecto al árbol genealógico de un perro, hay que tener en cuenta que cada perro tiene una madre cierta y un presunto padre; además, en el 10% de los casos existe un segundo padre presunto. Con la información detallada, obtener el modelo ER para el *Cursus Honorum* de los perros.

14. Los conductores de una empresa de reparto de mercancías se identifican por un número, tienen un nombre, una dirección y una fecha de nacimiento. Su trabajo consiste en la realización de repartos en los vehículos de la empresa.

    Cada vehículo se identifica por un número, es de una determinada marca y se fabricó en un año específico. Un vehículo puede ser utilizado siempre que esté disponible y puede estar de servicio durante cualquier periodo de tiempo (desde una o dos horas hasta cualquier número de días). También es posible que un vehículo esté de servicio más de una vez en el mismo día. Cualquier vehículo puede ser utilizado por cualquier conductor, cualquier número de veces cada día.

    Cada vez que un conductor utiliza un vehículo, lo hace para repartir cualquier cantidad de cualquier tipo de producto (los tipos de producto se identifican por un número y tienen un color, un peso y una descripción). Cada vez que se utiliza un vehículo, se producen una serie de gastos de varios tipos (por ejemplo, gasto de gasolina). Cada tipo de gasto tiene un número de código. Se lleva control del importe y del código, cada vez que se realiza un gasto. Uno o más gastos del mismo tipo pueden producirse durante el mismo viaje. Se pueden realizar cualquier número de entregas de mercancía durante el viaje. Para cada parada, se lleva control de la dirección en que se entrega mercancía, junto con la cantidad de cada tipo de producto entregado.

Un conductor entrega mercancía en una dirección sólo una vez durante el viaje. Sin embargo, se puede entregar en diferentes viajes.

Con la información que se ofrece, detallar el modelo ER para esta empresa.

15. Se describe a continuación una base de datos que contiene la información relativa a una empresa que se dedica a la fabricación de motores. Dicha empresa tiene interés en mantener información sobre los distintos motores que fabrica. Se pide construir el diagrama ER correspondiente.

    La empresa está dividida en secciones, con un código de identificación, un nombre y un presupuesto, a las que están asignados obligatoriamente los empleados, los cuales tienen que pertenecer a una sola sección. Sobre cada empleado se guarda información relativa a su código de empleado, nombre, dirección y año de ingreso en la empresa. A cada sección se le puede asignar la fabricación de uno o varios tipos de motores, existiendo secciones generales (administración, proceso de datos, etc.) que no tienen asignado ningún tipo de motor en concreto; nunca un tipo de motor puede ser fabricado por más de una sección, ni ésta puede cambiar una vez que se le ha asignado la fabricación de un determinado motor. Sobre cada motor interesa su código de identificación, descripción y peso. Entre los empleados que trabajan en una tipo de motor siempre hay un responsable, siendo esta información de interés para la empresa. Un empleado puede estar trabajando en más de un tipo de motor.

    Los empleados pueden tener familiares a su cargo, cuyos datos (nombre, edad, parentesco, etc.) conviene guardar en la base.

    La empresa utiliza piezas, con un código y una descripción, para fabricar los motores y desea conocer la estructura del motor (piezas que lo componen); a estos efectos, el motor es considerado una pieza.

16. Se desea mantener en una base de datos información referente a la marcha de una carrera ciclista.

    - De cada corredor que participa en ella hay que conocer su dorsal, su nombre, su edad, su nacionalidad, el equipo con el que corre, el nombre del director y la nacionalidad del equipo.
    - La carrera consta de varias etapas que están numeradas consecutivamente; cada etapa tiene un lugar de salida y otro de llegada y recorre un número dado de kilómetros. Hay algunas etapas en las que los corredores deben subir puertos de montaña; en una etapa pueden subirse varios puertos, un puerto puede ascenderse en diferentes etapas y varias veces en la misma etapa. Cada

puerto tiene un nombre y ocupa una posición relativa dentro de la etapa en la que se asciende, asignándole en esta posición una categoría, que indica la dificultad de la ascensión.

- El objetivo fundamental del sistema a construir es conocer la clasificación general y de la montaña de los corredores. La general se obtiene conociendo el tiem po que invierte cada corredor en recorrer las etapas; ordenando a los corredores de menor a mayor tiempo utilizado se obtiene su puesto en la carrera. La clasificación de la montaña se obtiene adjudicando un número determinado de puntos a los corredores, según el orden de paso por el puerto; ordenando a éstos de mayor a menor número de puntos conseguidos se obtiene su puesto en la montaña.

Con la información que se ofrece, detallar el esquema Entidad-Relación para este supuesto.

17. Suponer que, al diseñar una base de datos, se quiere recoger la información referente a toda la educación universitaria de los estudiantes.

    - Para cada uno de ellos, hay que tener en cuenta todas las universidades a las que ha asistido, recogiendo el nombre de la universidad, las fechas de inicio y fin de la estancia, los títulos obtenidos en cada una de ellas y las asignaturas que ha completado.
    - Para cada título interesa conocer su nombre, y el mes y el año en que se obtuvo.
    - Para cada asignatura, el nombre, el mes y el año en que se completó, el curso al que pertenece y el título en el que se cursa.

    Con la información que se ofrece, detallar el diagrama Entidad-Relación para este supuesto.

18. Se trata de diseñar una base de datos que contenga información referente a la planificación de las operaciones de los cirujanos de un hospital.

    - El hospital comprende un cierto número de servicios (con un código y un nombre) a los que están adscritos los cirujanos.
    - Cada servicio dispone de habitaciones para la hospitalización de los enfermos. Estas habitaciones, que tienen un número único sobre el conjunto del hospital, pueden contener de una a cuatro camas, según el caso. Cada cama está numerada de uno a cuatro en la habitación.
    - Los informes concernientes a la hospitalización de los enfermos (Número de enfermo, nombre, fecha de nacimiento, fecha de hospitalización, ...) se archivan durante su estancia y no se conservan una vez que abandonan el hospital.

- Cada cirujano tiene una especialidad, pero varios de ellos pueden tener la misma. Las especialidades están codificadas y tiene un nombre.
- Puede suceder que un enfermo deba sufrir varias operaciones durante una misma hospitalización, en diferentes fechas y realizadas por cirujanos que pueden pertenecer a un servicio diferente a aquel en el que están hospitalizados.

Con la información que se ofrece, detallar el esquema Entidad/Relación para este supuesto.

19. Una agencia de colocaciones dedicada a la búsqueda de directivos para empresas desea diseñar una base de datos que incorpore datos de las personas de las que tiene información, así como de los trabajos que oferta.

    - De cada persona se desea poder relacionarla con la ciudad en la que vive y con todas las ciudades en las que estaría dispuesta a vivir, dándoles un orden de preferencia.
    - Además, cada persona debe estar relacionada con aquellas experiencias (tanto académicas como profesionales) que posee, indicando los años durante los cuales las ha adquirido.
    - Deben incluirse, también, las empresas (tanto las que ofertan trabajo, como aquellas a las que pertenecen o han pertenecido las personas de las que se tiene información) y deberá relacionarse cada persona con su empresa actual y con todas a quellas en las que ha trabajado, indicando, en este caso, la fecha de entrada, salida, motivo de la salida y último puesto ocupado.
    - Cada trabajo ofertado requiere de una o varias experiencias y es para una sola empresa y en una única ciudad.
    - Por último, al comenzar el proceso de selección para un trabajo, se establece una relación entre el trabajo y las personas que optan a él, de forma que en cada selección aparecen varias personas y una persona puede estar implicada en varias selecciones a la vez.

    Con la información que se ofrece, detallar el esquema Entidad/Relación para este supuesto, añadiendo los atributos que se consideren necesarios.

20. Se trata de diseñar una base de datos que permita realizar la gestión de los préstamos de una biblioteca. Actualmente se vienen utilizando dos tipos de fichas:

    - Fichas para recoger las características de los libros con la información: Título, autor o autores, editorial, año de publicación y número de ejemplares de que se disponen.

- Fichas para los préstamos, que recogen: Persona a la que se ha prestado, fecha de inicio del préstamo y fecha de devolución.

Además, se pretende considerar los siguientes nuevos aspectos:

- En la biblioteca existen un conjunto de artículos que también pueden ser prestados. Interesa recoger las características de estos artículos (título del artículo, y, para cada revista en la que se ha publicado, nombre de la revista, volumen, número, fecha de publicación, num. de página inicial y num. de página final) y los préstamos que de ellos se realicen.
- Las recopilaciones son un tipo de libro formado por la reunión de varios artículos hecha por un editor o varios acerca de un tema, un congreso, etc... Además de poder prestar los artículos de una recopilación, se podrá prestar también la recopilación completa como si fuera un libro.
- También se prestan las revistas, de las que habrá que conocer su nombre y la editorial que las publica.
- Hay que conocer el idioma en que están escritos los libros.
- Cada artículo, libro, etc... trata de uno o varios temas. Los temas se pueden dividir en subtemas y así sucesivamente.
- De los autores interesa conocer el nombre, la nacionalidad y las instituciones en las que trabajan.
- De las editoriales, su nombre, dirección, ciudad y país.
- Existe un conjunto de documentos no publicados (no son, por tanto, ni libros ni artículos) y que también pueden prestarse: informes técnicos, trabajos, ...
- De los socios interesa conocer su número de socio, el DNI, el nombre y el teléfono. Además, se quiere distinguir a tres tipos de socios: Alumnos, Alumnos de doctorado o Proyectos Fin de Carrera, Profesores y otras bibliotecas.

Con la información que se ofrece, detallar el esquema Entidad-Relación para este supuesto.

21. Se trata de diseñar una base de datos para un entorno de apoyo a la programación.

    - En este entorno los programadores producen programas, que se escriben en los diferentes lenguajes de programación soportados por el sistema.
    - Cada programa es escrito por un único programador, puede llamar a otros programas y puede ser utilizado por determinados usuarios.

- Los usuarios se reconocen por su nombre de entrada al sistema; los programadores se reconocen por su nombre de entrada al sistema o por su código de programador; los programas se identifican por el nombre del programa, la extensión y el código del programador que los ha escrito.
- Los programas tienen un número de versión, una fecha y una descripción breve; algunos programas interactúan con las bases de datos del sistema.
- Cada base de datos la define un único administrador de bases de datos, que es un programador especializado en la administración de datos.

Diseñar el esquema E-R asociado a este supuesto.

22. Una organización quiere construir una base de datos que contenga información sobre su personal. La estructura de la organización es:

    - La compañía tiene un conjunto de departamentos.
    - Cada departamento tiene un conjunto de empleados, un conjunto de proyectos, y un conjunto de oficinas.
    - Cada empleado tiene una historia de empleos (conjunto de trabajos que ha tenido el empleado).
    - Para cada empleo, el empleado tiene también una historia de salario (conjunto de salarios recibidos mientras tenía ese empleo).
    - Cada oficina tiene un conjunto de teléfonos.

    La base de datos debe contener la siguiente información:

    - Para cada departamento: número de departamento (único), presupuesto, y el número de empleado del gerente del departamento (único).
    - Para cada empleado: número del empleado (único), número del proyecto en el que trabaja actualmente, número de la oficina en la que trabaja y número de teléfono del mismo; además, el título de cada trabajo que ha tenido el empleado, junto con la fecha y el salario para cada salario distinto recibido en ese trabajo.
    - Para cada proyecto: número de proyecto (único) y presupuesto.
    - Para cada oficina: número de oficina (único), área en metros cuadrados, y número de todos los teléfonos de esa oficina.

    Con la información que se ofrece, detallar el esquema Entidad-Relación para esta organización, indicando todas las suposiciones que se realicen.

23. Se pretende mantener información sobre músicos, composiciones y sus representaciones.

    - Todos los músicos están identificados por un número, tienen un nombre, nacieron en una fecha y tienen un teléfono de contacto. Algunos músicos componen piezas de música.
    - Cada composición musical tiene un número que la identifica, un título y ha sido compuesta por un solo músico en una fecha determinada.
    - Otros músicos son instrumentistas y saben tocar un instrumento con cierto grado de destreza (Muy bien, Bien, Regular). Los instrumentistas tiene un número único que los identifica como tales y pueden saber tocar varios instrumentos, cada uno de ellos con un grado de destreza diferente.
    - Los músicos que son instrumentistas se agrupan formado bandas, orquestas, cuartetos... Un instrumentista puede formar parte de varios grupos. Cada grupo tiene un número que lo identifica, un nombre y una ciudad donde reside, además siempre hay un músico representante del grupo que puede ser o no instrumentista.
    - Las composiciones musicales se representan en distintas localidades. Hay que controlar la fecha de representación, la localidad y provincia donde se representa, el grupo que la representa y el músico que ejerce como director. Una composición puede ser representada en muchos lugares, en fechas distintas y por grupos distintos.
    - Un músico puede ser instrumentista, y/o compositor, y/o director, y/o representante.

    Obtener un esquema E-R asociado a este supuesto, detallando todas las suposiciones que se realicen.

24. Se trata de diseñar un sistema de bases de datos para una Facultad de Farmacia, División de Farmacocinética Clínica.

    - La división tiene proyectos de investigación en varias etapas de desarrollo: en curso, pendientes y completos.
    - Cada proyecto de investigación está identificado por un código, tiene un presupuesto para su ejecución y ha de cumplir unos objetivos iniciales.
    - Un proyecto estudia los efectos de un fármaco en varios sujetos: a cada sujeto se le administra una cantidad de fármaco en unas fechas determinadas y se anotan los resultados obtenidos. La información de los sujetos que han participado en un proyecto es utilizable para futuros proyectos.

- Los diversos fármacos usados en los experimentos suelen ser proporcionados por una o más compañías famacéuticas.
- Varios de los empleados de la Facultad de Farmacia trabajan en cada proyecto, dirigido por un investigador principal; cada investigador principal puede controlar varios proyectos.
- Cuando se completa un estudio, se hace un informe de la investigación, donde se describen los resultados del estudio.

Construir el esquema E-R asociado a este supuesto.

Capítulo 4

El Modelo Relacional: Introducción

En este capítulo analizaremos el modelo relacional. Empezaremos exponiendo la historia y objetivos del modelo, para pasar a continuación a presentar los aspectos estáticos del mismo. Finalizaremos el capítulo estudiando el modelo relacional en el contexto de la arquitectura ANSI.

4.1 Presentación y Objetivos

A finales de los años sesenta Codd introdujo la teoría de las relaciones en el campo de las bases de datos. Propuso un modelo de datos basado en la teoría de las relaciones, en donde los datos se estructuran lógicamente en forma de *relaciones*, tablas, siendo un objetivo fundamental del modelo el mantener la independencia de esta estructura lógica respecto al modo de almacenamiento y a otras características de tipo físico.

Con respecto a la parte dinámica del modelo, se propone un conjunto de operadores que se aplican a las relaciones. Algunos de estos operadores son clásicos de la teoría de conjuntos, —no hay que olvidar que una relación es un conjunto–, mientras que otros fueron introducidos específicamente para el modelo relacional. Todos ellos conforman el álgebra relacional.

Las indiscutibles ventajas del modelo relacional no le han librado de críticas, especialmente relativas a la poca eficiencia de los primeros prototipos y productos comerciales y a su falta de semántica. Probablemente, la teoría relacional nació cuando la tecnología existente en aquellos momentos no podía todavía ofrecer la base adecuada para que las instrumentaciones repondiesen eficientemente a las necesidades de los usuarios.

A pesar de ello, el modelo de datos relacional ha tenido un auge espectacular desde finales de los años setenta y, sobre todo, en los ochenta, una vez que empezaron a vencerse las dificultades que presentaba su instrumentación y gracias al desarrollo tecnológico que ha permitido una mayor eficiencia en los productos relacionales.

4.2 Estructura del Modelo Relacional

El elemento básico del modelo relacional es la relación, que se representa mediante una tabla (Figura 4.1).

relación	atributo 1	atributo 2	....	atributo n
	xxxx	xxxx	....	xxxx
	xxxx	xxxx	....	xxxx
	xxxx	xxxx	....	xxxx

Figura 4.1: Representacion de una relacion en forma de tabla.

En ella podemos distinguir un conjunto de columnas, denominadas *atributos* (A), que representan propiedades de la misma y que están caracterizadas por un nombre, y un conjunto de filas llamadas *tuplas* (μ), que son las ocurrencias de la relación. El número de filas de una relación se llama *cardinalidad*, mientras que el número de columnas es el *grado*. Existen también *dominios* de donde los atributos toman sus valores.

Un *dominio* es un conjunto finito de valores $V_1, V_2, \ldots, V_n$ homogéneos y atómicos:

- homogéneos: porque son todos del mismo tipo,
- atómicos: porque son indivisibles.

Los atributos toman sus valores de un determinado dominio. Si A es un atributo, el dominio de A se denota por $dom(A)$. Asi, $A \in dom(A)$

Dentro de la representación de las relaciones como tablas se pueden hacer las siguientes consideraciones:

- No puede haber filas duplicadas. Todas las tuplas tienen que ser distintas.
- El orden de las filas es irrelevante.
- El orden de los atributos no es significante.
- La tabla es plana, es decir, en el cruce de una fila y una columna sólo puede haber un valor (no se admiten atributos multivaluados.

Una relación siempre tiene un nombre, y en ella es posible distinguir una cabecera o *formato de la relación*, que define la estructura de la tabla, y un cuerpo, que está formado por un conjunto de tuplas que varían en el tiempo.

4.2.1 Relación

Para dar una definición de relación es preciso distinguir entre dos conceptos en la noción de relación (Figura 4.2):

- *Formato de relación*: denotado $X = \{A_1, A_2, \ldots, A_n\}$, es un conjunto de n atributos, donde n es el *grado* del esquema de relación. El formato es la parte definitoria y estática de la relación. En condiciones normales, los dominios se obviarán en la definiciones de los esquemas de relación, excepto cuando sea preciso hacerlos explícitos para aclarar dichas definiciones. Podemos representar por $Tup(X)$ al conjunto de todas las tuplas μ con el formato X.

- *Instancia de relación, o relación*: r es un conjunto finito de tuplas sobre X, $r \subseteq Tup(X)$. Si $Rel(X)$ representa el conjunto de todas las posibles relaciones sobre X, se puede decir $r \in Rel(X)$. El número de tuplas m es la cardinalidad. La relación $r \in Rel(X)$ es, por tanto, el conjunto de tuplas que, en un instante dado, satisfacen el correspondiente formato de relación X; y así como el formato de la relación es 'invariante', su instancia varía en el transcurso del tiempo.

Formato de relación: $LIB\{sig,\ aut,\ tit,\ edi\}$

Instancia de relación:

LIB	sig	aut	tit	edi
	1	Date	Intro DB I	AW
	2	Date	Intro DB II	AW
lib	3	de Miguel & Piattini	Concep y Diseño BD	RAMA
	4	Korth & Silberschatz	Fundament BD	MGH
	5	Ullman	Princ DB	CSP
	6	Vossen	Data Models, DB Lang	AW

Figura 4.2: Formato e instancia de la relación libro.

4.2.2 Restricciones de Integridad y Claves

De entre todas las posibles combinaciones de tuplas que pueden formar una relación, sólo interesarán aquellas que tengan sentido con respecto a las aplicaciones en las que se definen:

No serían válidos en una aplicación de biblioteca los siguientes sucesos:

- libros con la misma signatura,

- un libro prestado a varios lectores,
- libros con el mismo título pero de diferentes autores,
- préstamo de libros que no estén en la biblioteca.

Sólo interesan aquellas relaciones que cumplen ciertas condiciones semanticas. Anteriormente se han concretado bajo el concepto de clave, pero esto no es suficiente para modelar algunas observaciones del mundo real. En general, las observaciones o condiciones de este tipo se denominan *dependencias de datos*, y hacen posible establecer con precisión cuales de las relaciones de un conjunto dado $Rel(X)$ pueden ser consideradas 'razonables' o 'válidas' (y pueden ocurrir por tanto en la base de datos correspondiente), y cuáles no.

Se pueden establecer dos clases de dependencias de datos:

- Dependencia *estática* sobre X: tiene que ser satisfecha por todo estado de la relacion $r \in Rel(X)$(*restricción intrarelacional*).
- Dependencia *dinámica*: afecta a la transición que, se da en una actualización desde una relación actual $r_1 \in Rel(X_1)$ hacia otra relación $r_2 \in Rel(X_2)$ (*restricción interrelacional*).

Un ejemplo muy común de restricción intrarelacional es el de clave primaria. Y las claves ajenas serán un ejemplo de restricción interrelacional.

Una *clave candidata* de una relación es un conjunto no vacío de atributos que identifican unívoca y mínimamente cada tupla. Por la propia definición de relación, siempre hay, al menos, una clave candidata, ya que al ser una relación un conjunto, no existen dos tuplas repetidas, y por tanto, el conjunto de todos los atributos identificará unívocamente a las tuplas.

Una relación puede tener más de una clave candidata, entre las que distinguiremos:

- *Clave primaria*: aquella clave candidata que el usuario escogerá para identificar las tuplas de la relación.
- *Claves alternativas*: aquellas claves candidatas que no han sido escogidas como clave primaria.

Así, en toda relación tendremos una *dependencia de clave*, que restringe el conjunto de posibles tuplas de la relación a aquellas que cumplen la clave.

Dado que existen restricciones (intrarelacionales) que deben cumplir las tuplas de una determinada relación, será adecuado que éstas aparezcan en la definición de la parte invariante de la relación junto al formato de dicha relación. Así, si el conjunto de atributos K, con $K \subseteq X$, presentan una dependencia de clave, ésta se representará como $K \to X$. Esto es, si $\mu[K] = \nu[K] \Rightarrow \mu = \nu$, que si dos tuplas distintas μ y ν tienen los mismos valores en los atributos de K, entonces es que tienen iguales valores en todos los atributos de X.

Por lo tanto, podemos definir el *esquema de relación* como $R = (X, \Sigma_X)$: un nombre R, un formato X (conjunto de atributos), y un conjunto Σ_X de dependencias intrarelacionales.

Así, el esquema de relación $R = (X, \Sigma_X)$ puede ser utilizado como una descripción invariante en el tiempo del conjunto de todas las relaciones con el formato X que satisfacen Σ_X. Cada una de estas relaciones se llama una *instancia* de R.

Como ejemplo, podemos describir el conjunto de libros almacenados en la biblioteca como

$LIB = (\{sig,\ aut,\ tit,\ edi\}, \{sig \rightarrow \{sig,\ aut,\ tit,\ edi\}\})$

o de manera reducida

$LIB = (\{sig,\ aut,\ tit,\ edi\}, \{sig\})$

4.3 Bases de Datos Relacionales y Esquemas de Bases de Datos

Llegados a este punto, ya se puede intentar representar un diagrama ER en su correspondiente forma relacional. Tendremos que hacer la transformación en dos pasos, primero tratar los tipos de entidad y después los tipos de relación:

- Tipos de entidad: Por cada tipo de entidad introduciremos un esquema de relación con el mismo nombre, cuyo formato estará formado por el mismo conjunto de atributos que el tipo de entidad, y en el que se mantendrán las mismas dependencias de clave.

- Tipos de relación: Por cada tipo de relación introduciremos un esquema de relación con el mismo nombre, cuyo formato estará formado por los atributos de las claves primarias de los tipos de entidad que relaciona el tipo de relación renombrando los atributos repetidos, mas los atributos propios del tipo de relación. Para formar las dependencias de clave hemos de fijarnos en las cardinalidades (máximo) del tipo de relación (supongamos binaria) R entre los tipos de entidades E_1 y E_2:

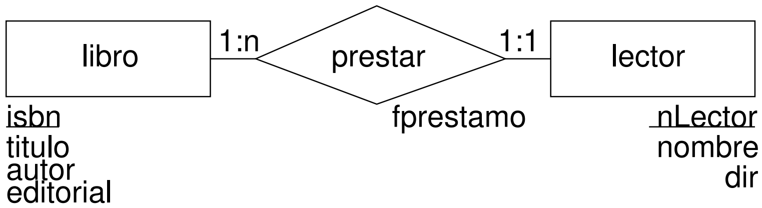
  - si R es $n : m$ entre E_1 y E_2 y E_i tiene como clave K_i, la clave se formará como $K_1 \cup K_2$.
  - si R es $1 : n$ entre E_1 y E_2, es suficiente con elegir K_2 para formar la clave.
  - si R es $1 : 1$ se puede elegir o K_1 o K_2.

  El ejemplo es para un tipo de relación binario, pero se puede extender fácilmente para cualquier grado mayor que 2.

Como resultado de la transformación de un diagrama ER en su forma correspondiente dentro del modelo relacional obtendremos un conjunto de esquemas de relación $\mathbf{R} = \{R_1, R_2, \cdots, R_k\}$. A este conjunto $\mathbf{R}$ de esquemas de relación se le puede llamar *formato de la base de datos*, como extensión del término formato de relación.

Las letras minúsculas serán utilizadas para representar objetos variables en el tiempo (tuplas, relaciones o bases de datos); mientras que las mayúsculas denotarán objetos invariantes en el tiempo (atributo, conjuntos de atributos, esquemas de relación, etc). También, los conjuntos de atributos de la forma $X = \{A, B, C, \cdots\}$ pueden escribirse como una concatenación de sus elementos (esto es, $X = ABCD \cdots$).

Un ejemplo de todo ello se puede observar en la figura 4.3, donde se muestra un diagrama ER y una instancia de la correspondiente base de datos según el modelo relacional.



LIB	*sibn*	*aut*	*tit*	*edi*
	1	Date	Intro DB I	AW
	2	Date	Intro DB II	AW
lib	3	de Miguel & Piattini	Concep y Diseño BD	RAMA
	4	Korth & Silberschatz	Fundament BD	MGH
	5	Ullman	Princ DB	CSP
	6	Vossen	Data Models, DB Lang	AW

LEC	*nle*	*nom*	*dir*
	500	Jose	VA
lec	550	Ana	LE
	600	Luis	PA

PRE	*isbn*	*nle*	*fpr*
	1	550	150496
pre	2	550	200496
	3	600	210496

Figura 4.3: Ejemplo de bases de datos relacional.

Dos aspectos que quedan pendientes de nuestra transformación del modelo de datos ER al Relacional son el tratamiento de la cardinalidad de los tipos de relación y la jerarquía derivada del uso de los tipo de relación IS-A.

Ambos tipos de restricción pueden ser expresadas en el modelo relacional utilizando las *dependencias interrelacionales*, que permiten establecer cuando una base de datos, esto es, un conjunto de relaciones, es 'razonable' o 'válida'.

Así, si $\mathbf{R}$ es un conjunto de esquemas de relación, y $\Sigma_{\mathbf{R}}$ un conjunto de dependencias interrelacionales, podemos definir un *esquema de base de datos* como $\mathbf{D} = \{\mathbf{R}, \Sigma_{\mathbf{R}}\}$.

En los capítulos siguientes vamos a tratar con esquemas de bases de datos en los que $\Sigma_{\mathbf{R}} = \emptyset$.

El caso concreto de los tipos de relación IS-A se puede ver con un poco mas de detalle.

Cuando nos encontramos con que hay que transformar al modelo Relacional un tipo de relación IS-A podemos hacerlo de dos maneras: (1) crear una relación por cada tipo de entidad, tanto sea un supertipo o un subtipo, de manera que los subtipos heredan los atributos que forman parte de la clave principal del supertipo; y (2) crear una sóla relación asociada al supertipo, en la que se añadirán los atributos que eran específicos de los subtipos.

Como criterio general para decidir cuál de estas dos formas de actuar es la más indicada se puede decir que si los subtipos aportan muchos atributos es mejor actuar del primer modo, ya que si creamos una sóla relación con la unión de todos los atributos, una tupla que esté asociada a uno de los subtipos tendrá valores nulos en todos los atributos que se refieran a los demás subtipos.

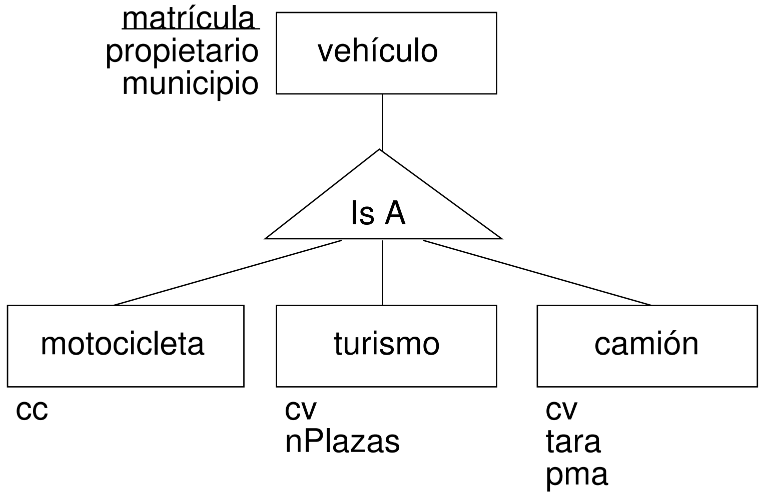
Todo esto se puede ver en la Figura 4.4. En el segundo caso, los atributos cc, cv, tur_npl, cam_tar y cam_pma son susceptibles de contener valores nulos.

4.4 El Modelo Relacional y la Arquitectura ANSI

El modelo relacional puede examinarse en el marco de la arquitectura ASNI de tres niveles.

Todos los objetos que hemos visto hasta el momento, esquemas de relación y restricciones, constituyen el esquema conceptual de la arquitectura ANSI. Las relaciones que podríamos llamar primarias, (cuyo esquema se define mediante la sentencia `CREATE TABLE` del SQL), se denominan *tablas base* o *reales*, ya que tienen una representación directa en el almacenamiento interno.

Existe otro tipo de tablas, denominadas *tablas virtuales* o *vistas*, que se definen sobre una o más tablas base. Las vistas son 'ventanas' sobre tablas reales, de las que sólo se almacena su definición y, por lo tanto, no tienen



1. veh({mat, pro, mun},{mat})
   mot({mat, cc},{mat})
   tur({mat, cv, npl},{mat})
   tur({mat, cv, tar, pma},{mat})
2. veh({mat, pro, mun, cc, cv, tur_npl, cam_tar, cam_pma},{mat})

Figura 4.4: Transformación de un tipo de relación IS-A.

representación directa en el almacenamiento; equivalen al esquema externo de la arquitectura ANSI.

Por lo que respecta al esquema interno, el modelo relacional no especifica absolutamente nada puesto que se trata de un modelo lógico. Cada producto ofrecerá los elementos internos, como índices, agrupamientos, tablas hash, particiones, etc, que el administrador utilizará para mejorar el rendimiento del sistema. En la Figura 4.5 se da una visión global del modelo relacional dentro del marco definido por la arquitectura ANSI, mostrando además ciertas sentencias del lenguaje SQL que ayudan a definir los elementos correspondientes en los tres niveles. Las sentencias que atañen al nivel interno variarán de un producto a otro y no están estandarizadas.

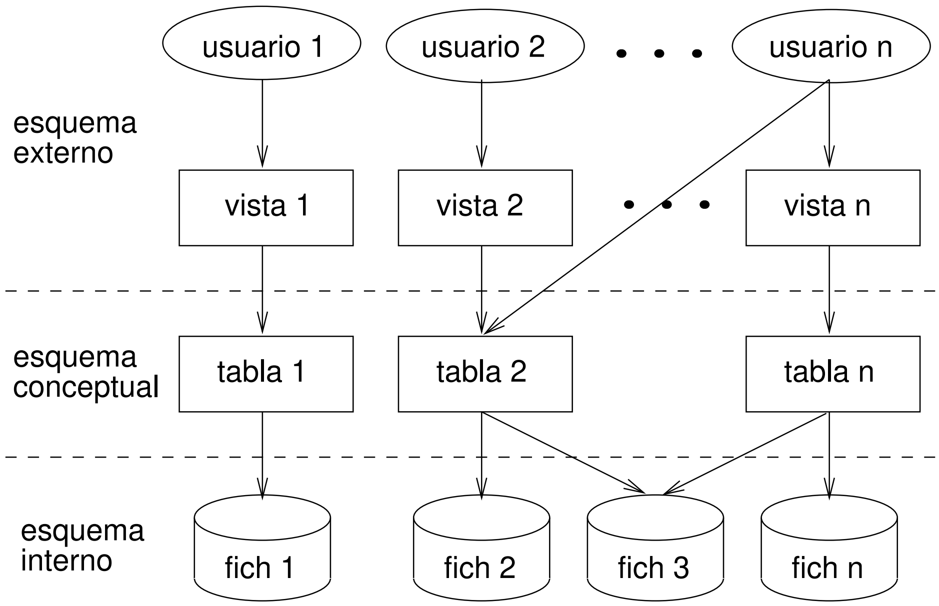


Figura 4.5: Arquitectura a tres niveles en un SGBD relacional.

Capítulo 5

Manipulación de Datos en el Modelo Relacional

En este capítulo vamos a familiarizarnos con dos lenguajes de consulta de las bases de datos relacionales: el álgebra relacional y el SQL.

5.1 Algebra Relacional

A continuación se presentan las operaciones sobre relaciones; que dan significado al lenguaje de manipulación llamado álgebra relacional.

Para comenzar, vamos a introducir las operaciones unarias, que pueden ser utilizadas para acortar las tablas en horizontal ('proyección') o en vertical ('selección').

Para ello vamos a definir el esquema de relación $R = (X, .)$ y la relación $r \in Rel(X)$, además de $Y \subseteq X$.

Proyección, π $\quad \pi_Y(r) = \{\mu[Y] \mid \mu \in r\}$

La proyección de r sobre Y, $\pi_Y(r)$, produce una relación r' con el formato Y a partir de la relación r. Es evidente que han de eliminarse las tuplas repetidas que se hayan generado al restringir el conjunto de atributos a considerar a solamente los incluidos en Y.

Selección, σ $\quad$ Sea $A \in X$, $a \in dom(A)$ y $\Theta \in \{<, \leq, >, \geq, =, \neq\}$. Entonces $\sigma_{A\Theta a}(r) = \{\mu \in r \mid \mu[A]\Theta a\}$.

La selección de r con respecto a $A\Theta a$, produce una relación r' a patir de r con el mismo formato X, pero en la que sólo permanecen las tuplas que cumplen la condición impuesta. Esta condición puede tomar formas más elaboradas incluyendo los operadores de relación $\wedge$,$\vee$ y $\neg$.

Como ejemplo, podemos cosiderar la siguiente relación $r \in Rel(ABC)$:

r:

A	B	C
1	2	1
1	1	1
2	2	2

$\pi_{AC}(r)$:

A	C
1	1
2	2

$\sigma_{A=1}(r)$:

A	B	C
1	2	1
1	1	1

$\sigma_{C>1}(r)$:

A	B	C
2	2	2

$\sigma_{B=2\wedge(A=1\vee C=2)}(r)$:

A	B	C
1	2	1
2	2	2

Para simplificar el trabajo con estas operaciones se pueden aplicar varias leyes:

Sea $r \in Rel(X)$

- $Z \subseteq Y \subseteq X \Rightarrow \pi_Z(\pi_Y(r)) = \pi_Z(r)$
- $Z, Y \subseteq X \Rightarrow \pi_Z(\pi_Y(r)) = \pi_{Z\cap Y}(r)$
- $\sigma_{C_1}(\sigma_{C_2}(r)) = \sigma_{C_2}(\sigma_{C_1}(r))$
- $\sigma_{C_1}(\sigma_{C_2}(r)) = \sigma_{C_1\wedge C_2}(r)$
- $A \in Y \subseteq X \Rightarrow \pi_Y(\sigma_{A\Theta a}(r)) = \sigma_{A\Theta a}(\pi_Y(r))$

Ahora estamos en disposicion de introducir algunos operadores binarios bien conocidos: *intersección* ($\cap$), *unión* ($\cup$) y *direrencia* ($-$):

Sean $r, s \in Rel(X)$:

- $r \cap s = \{\mu \in Tup(X) \,|\, \mu \in r \wedge \mu \in s\}$
- $r \cup s = \{\mu \in Tup(X) \,|\, \mu \in r \vee \mu \in s\}$
- $r - s = \{\mu \in r \,|\, \mu \notin s\}$

Es importante destacar que en ambas relaciones los operandos lo son sobre el mismo formato.

El siguiente operador destaca por su importancia en el trabajo con bases de datos relacionales. Es el primero que se define para más de dos operandos. Aunque el caso de dos operandos se pueda derivar fácilmente de ésta.

Sean $X_1, \cdots, X_n$ conjuntos de atributos, y sea $r_i \in Rel(X_i)$ con $1 \leq i \leq n$. Se define el *producto natural* de $r_1, \cdots, r_n$ como

$$\bowtie_{i=1}^{n} r_i = \{\mu \in Tup(\cup_{i=1}^{n} X_i) \mid (\forall i, 1 \leq i \leq n)\, \mu[X_i] \in r_i\}$$

Para el caso de $n = 2$, tendremos

$$r_1 \bowtie r_2 = \{\mu \in Tup(X_1 \cup X_2) \mid \mu[X_1] \in r_1 \wedge \mu[X_2] \in r_2\}$$

El producto natural calcula una nueva relación a partir de dos o más, en la que los atributos son la unión de los atributos de todos sus operandos, y una tupla del resultado será obtenida de aquellas tuplas de los operandos que tengan valores iguales en los atributos comunes.

Como ejemplo se pueden considerar las siguientes relaciones:

r_1 :

A	B	C
1	1	1
1	2	2
2	0	2

r_2 :

A	D
1	1
0	1
2	0

r_3 :

C	D	E
1	1	0
0	1	1
2	1	0
2	2	1

entre las que se cumple lo siguiente:

$r_1 \bowtie r_2$:

A	B	C	D
1	1	1	1
1	2	2	1
2	0	2	0

$(r_1 \bowtie r_2) \bowtie r_3$:

A	B	C	D	E
1	1	1	1	0
1	2	2	1	0

Sobre el producto natural se pueden aplicar algunas reglas sencillas:
Sea $r_i \in Rel(X_i)$ para $1 \leq i \leq 3$:

- $r_i \bowtie r_i = r_i$
- $r_1 \bowtie r_2 = r_2 \bowtie r_1$
- $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$
- $X_1 \cap X_2 = \emptyset \Rightarrow r_1 \bowtie r_2 = r_1 \times r_2$ [1]
- $X_1 = X_2 \Rightarrow r_1 \bowtie r_2 = r_1 \cap r_2$

de las que se sigue que el producto natural es conmutativo y asociativo; que 'degenera' en el producto cartesiano o en la intersección si el conjunto de atributos de los operandos es disjunto o idéntico, respectivamente.

En este punto se podría seguir definiendo otras operaciones que completarán el conjunto de operadores del álgebra relacional, pero con los operadores definidos tenemos suficiente para expresar consultas en el álgebra relacional.

Como ejemplo, consideremos la base de datos de la biblioteca presentada en la Figura 4.3, sobre la que se plantean varias consultas en el álgebra relacional:

[1] $\times$ representa el producto cartesiano.

- Mostrar todos los libros escritos por 'Date'.

$E_1 = \sigma_{aut=\text{'}Date\text{'}}(lib)$

sig	*aut*	*tit*	*edi*
1	Date	Intro DB I	AW
2	Date	Intro DB II	AW

- Mostrar todos las editoriales de las cuales tenemos algún libro.

$E_2 = \pi_{edi}(lib)$

edi
AW
RAMA
MGH
CSP

- Mostrar las fechas de préstamo de los libros que tienen signaturas o 1 o 2.

$E_3 = \pi_{fpr}(\sigma_{sig=1 \vee sig=2}(pre)$

fpr
150496
200496

- Mostrar la dirección de cada lector que tiene un libro en préstamo.

$E_4 = \pi_{dir}(lec \bowtie pre)$

dir
VA
LE
PA

- Mostrar el título del libro y el nombre del lector de cada prestamo.

$E_5 = \pi_{tit,nom}(lib \bowtie lec \bowtie pre)$

tit	*nom*
Intro DB I	Jose
Intro DB II	Jose
Concep y Diseño BD	Luis

Un aspecto importante a destacar llegados a este punto es que las selecciones y las proyecciones reducen generalmente los resultados intermedios (con respecto al número de filas y columnas), y que no se cumple en el producto. De este modo, a menudo es posible acortar el tiempo de evaluación o de respuesta. Esto se puede ilustrar gráficamente mediante un árbol sintáctico. Consideremos las dos consultas equivalentes siguientes que pretenden obtener la signatura y el título de todos los libros prestados a Laura:

$E_1 = \pi_{sig,tit}(\sigma_{nom='Laura'}(lib \bowtie lec \bowtie pre))$

$E_2 = \pi_{sig,tit}(\pi_{sig,tit}(lib) \bowtie (pre \bowtie \sigma_{nom='Laura'}(lec)))$

Sus correspondientes árboles sintacticos nos pueden ilustrar el que es mejor acercar las proyecciones y las selecciones hacia las hojas del mismo, para reducir el conjunto de tuplas y el número de columnas de las relaciones resultado intermedio, antes de aplicar la operación producto natural:

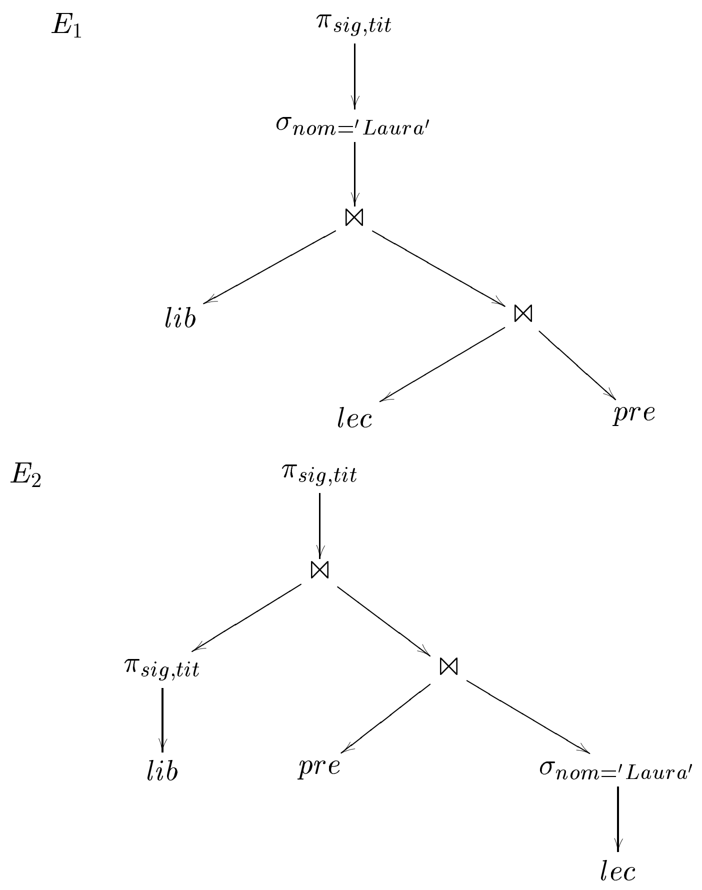


Figura 5.1: Arboles sintacticos.

5.2 El Lenguaje SQL

Cuando a comienzos de los años 70 las universidades y los centro de investigación empiezan a preocuparse por el desarrollo de prototipos que respondan a las propuestas de Codd relativas a un nuevo modelo de datos, surge la necesidad de diseñar lenguajes que soporten el modelo relacional.

Los lenguajes predicativos y algebráicos, tal como habían sido definidos a nivel teórico adolecían de dificultades derivadas de su forma abstracta, distinta de la de los lenguajes tradicionales y demasiado matemática.

El SQL surge originariamente con el nombre de SEQUEL (*Structured English QUEry Languaje*) implementado sobre un prototipo de IBM durante los años 1974-75. Posteriormente se introduce en el sistema R, y tras una serie de modificaciones pasa a denominarse SEQUEL II, para convertirse má tarde, por cuestiones legales, en el SQL (*Structured Query Languaje*). EN 1979 surge el primer SGBDR comercial basado en SQL (Oracle), posteriormente van apareciendo otros productos basados en este lenguaje como son el

SQL/DS, DB2, SYSBASE, INFORMIX, RDB, etc. E incluso otros productos que no poseían SQL como lenguaje nativo, como INGRES y ADABAS empiezan a ofrecer interfaces SQL. Por lo que este lenguaje se confierte en un estándar *de facto*, aunque con múltiples variantes, según los distintos fabricantes.

En 1982 el comité de bases de datos X3H2 de ANSI, presenta un lenguaje relacional estándar basado principalmente en el SQL propio de los productos IBM; en 1986 este organismo aprueba el lenguaje como norma pasando a denominarse SQL/ANSI, que también es aprobado al año siguiente como norma ISO.

Este estándar ha recibido numerosas críticas, ya que resulta ser una intersección de las implementaciones existentes, concebido primordialmente para proteger los intereses de los fabricantes.

En 1989 se publica una nueva versión del estándar conocida como *Addendum*.

EN 1992 se aprobó una nueva norma, el SQL2, en el que se incrementa substancialmente la capacidad semántica del esquema relacional, se añaden nuevos operadores, se mejora el tratamiento de errores y se incluyen normas para el SQL embebido.

En la actualidad está en desarrollo una nueva versión, SQL3.

No vamos a hacer una descripción exhaustiva del lenguaje SQL, sino a presentar algunos ejemplos que nos permitan apreciar la estructura y sintaxis del lenguaje, ya que el SQL se describe en toda su extensión en casi cualquier libro de bases de datos relacionales. Además la versión de SQL que se va a mostrar puede que no coincida con la versión SQL utilizada por los SGBD comerciales, aunque la diferencia entre ambas versiones no será demasiado grande.

5.2.1 Definición de Datos

En esta sección se tratan los aspectos relacionados con la declaración de los esquemas de relación; esto incluye los atributos y sus tipos de datos, el uso de valores nulos, claves primarias, indices, derechos de acceso, etc. Cada declaración se almacena en el diccionario de datos del sistema.

Creación de Tablas

Un esquema de relación se va a representar por una tabla, que tendrá un nombre, una serie de atributos y un tipo de dato para cada uno de ellos. Opcionalmente tendrá una clave primaria. Todo esto se puede realizar con el comando `CREATE TABLE`:

```
CREATE TABLE nombreTabla
(nombreColumna1 tipoColumna1,
```

nombreColumna2 tipoColumna2, ... , PRIMARY KEY (columnasClave))

Tipos de Datos

Aunque existen más, estos son los principales:

- INTeger: enteros (-2.147.483.647, 2.147.483.647).
- SMALLINT: enteros pequeños (-32767,32767).
- FLOAT(n): números en punto flotante con n digitos para el exponente (5.4E-79,7.2E+75).
- CHARacter(n): cadenas de caracteres de longitud fija ($n < 254$).
- VARCHARacter(n): cadenas de longitud variable ($n \leq 32767$).
- DATE: fechas.
- TIME: tiempo.

Modificaciones de Esquemas

También se pueden modificar las tablas definidas con anterioridad utilizando los siguientes comandos:

`ALTER TABLE` nombreTabla
`ADD` nombreColumna tipoColumna

`ALTER TABLE` nombreTabla
`ADD PRIMARY KEY` (listaColumnas)

`ALTER TABLE` nombreTabla
`DROP` nombreColumna

`DROP TABLE` nombreTabla

Actualización de Tablas

Después de crear las tablas se puede insertar, modificar o eliminar información en las mismas.

`INSERT INTO` nombreTabla
[(listaColumnas)]
{`VALUES` (datos) | comandoSelección }

`UPDATE` nombreTabla
`SET` nombreColumna = expresión

`DELETE FROM` nombreTabla
`WHERE` condición

5.2.2 El Comando SELECT

Las consultas en SQL son posibles gracias al comando `SELECT`, que tiene una estructura muy sencilla:

`SELECT` atributos	('salida')
`FROM` relacion(es)	('entrada')
`WHERE` condición	

En términos del álgebra relacional, el comando `SELECT` es equivalente a la asociación de una selección , una proyección y un producto cartesiano. Esta asociación se hace de la siguiente manera. Los atributos de la proyección se especifican en la cláusula `SELECT`. La cláusula `FROM` especifica las relaciones de las que se han de tomar esos atributos. Y finalmente, la cláusula `WHERE` contiene una condición de selección (opcional).

La sintaxis del `SELECT` es tan flexible como compleja (ver diagramas de flujo), y está explicada en los textos base del curso.

Se puede establecer una correspondencia entre diferentes casos del `SELECT` y el significado de los mismos:

Sintaxis	Semántica
`SELECT` Y `FROM` R	$\pi_Y^+(R)$
`SELECT DISTINCT` Y `FROM` R	$\pi_Y(R)$
`SELECT` $*$ `FROM` R `WHERE` C	$\sigma_C(R)$
`SELECT` Y `FROM` R `WHERE` C	$\pi_Y^+(\sigma_C(R))$
`SELECT` $*$ `FROM` R,S	$R \times S$
`SELECT` Y `FROM` R,S `WHERE` C `AND` D	$\pi_Y^+(\sigma_{C\wedge D}(R \times S))$

En el caso de que distintos esquemas de relación compartan el nombre de uno de sus atributos, al referirse a dichos atributos se tendrá que utilizar como prefijo el nombre del esquema del que provienen ('$R.A$'). Esto es de vital importancia para poder obtener el producto natural:

Si R tiene como atributos $A_1, \cdots, A_n, B_1, \cdots, B_m$ y S tiene como atributos $B_1, \cdots, B_m, C_1, \cdots, C_l$, la expresión `SELECT` correspondiente a la operación $R \bowtie S$ es la siguiente:

`SELECT` $A_1, \cdots A_n, R.B_1, \cdots, R.B_m, C_1, \cdots, C_l$
`FROM` R, S `WHERE` $R.B_1 = S.B_1$ `AND` $\cdots$ `AND` $R.B_m = S.B_m$

Podemos ver todo esto un poco mejor con un ejemplo de una base de datos de animales transgénicos (aquellos que han sido inyectados con material genetico):

animales	*numAni*	*sexo*	*color*	*fechNac*
	SMALLINT	CHAR	CHAR(7)	DATE
	NOT NULL		NOT NULL	NOT NULL
	12	m	marrón	1986-02-15
	15	f	negro	1986-02-22
	18	f	rojo	1986-10-10
	22	?	gris	1987-03-10
	23	f	gris	1987-03-10
	27	m	negro	1987-04-01
	35	m	marrón	1987-08-22
	43	m	marrón	1987-11-30
	52	?	naranja	1988-01-28
	88	?	azul	1988-04-20
	92	f	marrón	1988-05-22

jaulas	*numAni*	*numHabit*	*numCaja*
	SMALLINT	CHAR(4)	CHAR(3)
	NOT NULL	NOT NULL	NOT NULL
	12	1140	120
	15	1140	125
	18	1140	120
	22	1142	150
	23	1142	155
	27	2210	212
	35	2210	212
	43	2315	240
	52	2315	240
	88	2315	255
	92	2315	240

transgenicos	*numAni*	*fechInyec*	*tiempInyec*	*peso*
	SMALLINT	DATE	TIME	DEC(5,2)
	NOT NULL	NOT NULL	NOT NULL	
	18	1986-04-04	15.30.09	50.25
	22	1986-10-02	14.22.31	53.36
	35	1987-01-05	20.14.22	48.00
	88	1987-10-02	08.48.00	60.05
	92	1987-12-12	10.12.30	72.18

1. Mostrar, para cada animal transgénico, el número de animal y su peso en miligramos.

```
SELECT numAni, 'peso en miligramos =',peso * 1000 FROM
trangenicos
```

$numAni$	
18	peso en miligramos = 50250
22	peso en miligramos = 53360
35	peso en miligramos = 48000
88	peso en miligramos = 60050
92	peso en miligramos = 72180

2. Mostrar el número de caja y el número de animal para los ratones de la habitación 2315.

```
SELECT numAni, numCaja FROM jaulas WHERE numHabit =
'2315'
```

$numAni$	$numCaja$
43	240
52	240
88	255
92	240

3. Mostrar el número de animal para los ratones de la habitación número 2315, caja 240.

```
SELECT numAni FROM jaulas WHERE numHabit='2315' AND
numCaja='240'
```

$numAni$
43
52
92

4. Mostrar el número de animal para todos los animales en las habitaciones 1140 y 2315.

```
SELECT numAni FROM jaulas WHERE numHabit='1140' OR
numHabit='2315'
```

y también

```
SELECT numAni FROM jaulas WHERE numHabit IN ('1140'
, '2315')
```

$numAni$
12
15
18
43
52
88
92

5. Mostrar el número de animal de los que pesen más de 51 y menos de 80 gramos.

```
SELECT numAni FROM transgenicos WHERE peso > 51 AND
peso < 80
```

y también

```
SELECT numAni FROM transgenicos WHERE peso BETWEEN
51.01 AND 79.99
```

numAni
18
22
88
92

6. Mostrar los números de animal de los ratones cuyo color comience con una 'm'.

```
SELECT numAni FROM animales WHERE color LIKE 'n%'
```

numAni
15
27
52

7. Mostrar los números de animal de los ratones que viven en la segunda planta.

```
SELECT numAni FROM jaulas WHERE numHabit LIKE '2___'
```

numAni
27
35
43
52
88
92

8. Mostrar los números de animal y tiempo de inyección de todos los transgénicos que pesen menos de 60 gramos, presentándolos en orden decreciente de tiempo de inyección.

```
SELECT numAni, tiempInyec
FROM transgenicos
WHERE peso < '60' ORDER BY tiempInyec DESC
```

numAni	*tiempInyec*
35	20.14.22
18	15.30.09
22	14.22.31

9. Mostrar el número de todas las habitaciones que tiene mas de dos ratones.

```
SELECT DISTINCT numHabit FROM jaulas
GROUP BY numHabit HAVING COUNT(*)>2
```

jaulas	*numAni*	*numHabit*	*numCaja*
	12	1140	120
	15	1140	125
	18	1140	120
	22	1142	150
	23	1142	155
	27	2210	212
	35	2210	212
	43	2315	240
	52	2315	240
	88	2315	255
	92	2315	240

numHabit
1140
2315

10. Mostrar el número de animal para cada ratón en una habitación que aloje a más de dos ratones, en orden descendente de número de caja.

```
SELECT numAni FROM jaulas
GROUP BY numHabit HAVING COUNT(*) > 2
ORDER BY numCaja DESC
```

numAni
15
12
18
88
43
52
92

11. Mostrar el número de animal de todos los ratones nacidos en 1987.

```
SELECT numAni FROM animales
WHERE fechNac BETWEEN '1987-01-01' AND '1987-12-31'
```

y también de manera más simple

```
SELECT numAni FROM animales
WHERE YEAR(fechNac) = 1987
```

12. Mostrar el número de animal de todos los transgénicos que fueron inyectados por la tarde o por la noche.

```
SELECT numAni FROM transgenicos
WHERE HOUR(tiempInyec) >=12 AND HOUR(tiempInyec) <=24
```

13. Mostrar cuántos ratones transgénicos hay.

```
SELECT COUNT(*) FROM transgenicos
```

14. Mostrar el número de cajas actualmente en uso.

```
SELECT COUNT(DISTINCT numCaja) FROM transgenicos
```

15. Mostrar el número de animales en cada habitación

```
SELECT DISTINCT numHabit, COUNT(numAni)
FROM jaulas
GROUP BY numHabit
```

Jaulas es dividida en grupos de igual número de habitación, y dentro de cada grupo, se han contado el número de elementos.

numHabit	
1140	3
1142	2
2210	2
2315	4

16. Mostrar el número de cada ratón transgénico cuyo peso esté por debajo de la media.

```
SELECT numAni FROM transgenicos
WHERE peso < (SELECT AVG(peso) FROM transgenicos)
```

17. Mostrar todos los transgénicos con su número de animal, sexo, color y peso.

```
SELECT transgenicos.numAni, sexo, color, peso
FROM transgenicos, animales
WHERE transgenicos.numAni = animales.numAni
```

Se ha efectuado un producto natural entre las relaciones transgenicos y animales.

18. Mostrar el número de habitación y el número de caja donde se encuentre un ratón marrón.

```
SELECT numHabit, numCaja
FROM animales, jaulas
WHERE jaulas.numAni = animales.numAni AND color='marron'
```

El mismo resultado se puede obtener mediante una subconsulta.

```
SELECT numHabit, numCaja
FROM jaulas
WHERE numAni IN
(SELECT numAni FROM animales
WHERE color = 'marron)
```

La subconsulta se evalúa primero, y su resultado se utiliza para evaluar la cláusula `WHERE` de la otra consulta.

19. Mostrar el número de habitación y de caja de aquellas que contengan ratones transgénicos marrones.

```
SELECT numHabit, numCaja
FROM animales, jaulas, transgenicos
WHERE animales.numAni = jaulas.numAni
AND jaulas.numAni = transgenicos.numAni
AND color = 'marron'
```

También se puede obtener el mismo resultado con dos niveles de subconsultas en lugar de realizar el producto natural de las tres tablas.

```
SELECT numHabit, numCaja
FROM jaulas
WHERE numAni IN (SELECT numAni
FROM animales
WHERE color = 'marron'
AND numAni IN
(SELECT numAni FROM transgenicos))
```

20. Mostrar el peso de los ratones marrones que viven en la habitación 2315.

```
SELECT peso FROM jaulas, transgenicos, animales
WHERE jaulas.numAni = transgenicos.numAni
AND transgenicos.numAni = animales.numAni
AND color = 'marron' AND numHabi= '2315'
```

Otra vez más, existe otra consulta alternativa basada en subconsultas.

```
SELECT peso FROM transgenicos
WHERE numAni IN
(SELECT numAni FROM jaulas
```

```
WHERE numHabi= '2315' AND numAni IN
(SELECT numAni FROM animales
WHERE color = 'marron'))
```

Las subconsultas pueden ser anidadas en otro orden, pero mientras el resultado sigue siendo el mismo, los tiempos de ejecución pueden ser diferentes, dependiendo del tamaño de los resultados intermedios o de la disponibilidad de índices.

21. Mostrar el número de animal y la fecha de nacimiento de todos los ratones transgénicos, en orden descendente de fechas de nacimiento.

```
SELECT animales.numAni, fechNac
FROM animales, transgenicos
WHERE animales.numAni = transgenicos.numAni
ORDER BY fechNac DESC
```

22. Mostrar todos los transgénicos para los que la diferencia entre sus fechas de inyección y de nacimiento sea más de 200 días.

```
SELECT transgenicos.numAni
FROM animales, transgenicos
WHERE DAY(fechNac - fechInyec) > 200
AND transgenicos.numAni = animales.numAni
```

23. Mostrar todas las parejas de animales que viven en la misma caja.

```
SELECT uno.numAni, dos.numAni
FROM jaulas uno, jaulas dos
WHERE uno.numCaja = dos.numCaja
AND uno.numAni < dos.numAni
```

numAni	*numAni*
12	18
27	35
43	52
52	92
43	92

24. Mostrar el peso total de todos los transgénicos marrones.

```
SELECT SUM(peso) FORM transgenicos WHERE numAni IN
(SELECT numAni FROM animales WHERE color = 'marron')
```

25. Mostrar el número de animal de los ratones nacidos o inyectados en 1987.

```
SELECT numAni FROM animales WHERE YEAR(fechNac) = 1987
UNION
SELECT numAni FROM transgenicos WHERE YEAR(fechInyec)
= 1987
```

La consulta `UNION` permite combinar los resultados de varias consultas. Se evalua primero las consultas individuales y luego se calcula el resultado global, cuyos nombres de columnas se toman de la primera consulta, y en cuyo contenido no se encontrarán tuplas repetidas. Existen algunas consideraciones acerca de la consulta `UNION`:

- la claúsula `ORDER BY` debe aparecer después de la última consulta, ya que se aplicará al resultado global.
- los dominios correspondientes a los ítems de las cláusulas `SELECT` de todas las consultas deben ser idénticos.
- si el resultado ha de aparecer en algún orden, la lista de atributos correspondiente al `ORDER BY` debe especificar el número de orden, no su nombre (ej. `ORDER BY 1,3`).
- no se puede utilizar `UNION` ni en una subconsulta ni en la definición de una vista.

5.2.3 Vistas

Una vista se diferencia de una tabla base en que es una tabla *virtual* que se deriva de una o más tablas base. Esto permite que sea posible hacer algunos datos visibles en el nivel externo según algún criterio (ej. todos los ratones marrones), o mantenerlos ocultos. Cuando se crea una vista lo único que se almacena es su definición, ya que los datos que la componen son tomados de las tablas base, y sobre ellos se aplican las operaciones o consultas que se refieran a la vista. Una vez que las operaciones sobre la vista se han realizado, el contenido de la vista se pierde, quedando sólo su definición.

Los comandos que permiten crear y destruir vistas son los siguientes:

`CREATE VIEW` nombreVista [(listaColumnas)]
`AS` sentenciaSelección

`DROP VIEW` nombreVista

Por ejemplo, se puede crear una vista con todos los animales femeninos:

```
CREATE VIEW femeninos AS
SELECT * FROM animales WHERE sexo = 'f'
```

Hay que tener en cuenta que no se puede utilzar `UNION` ni `ORDER BY` en la sentencia `SELECT` que define una vista.

Las vistas también pueden sufrir operaciones de actualización. Un comando `INSERT`, `DELETE` o `UPDATE` aplicado a una vista se transforma en el correspondiente comando sobre la relación base correspondiente. El resultado de una actualización será eventualmente visible en la vista (dependiendo de la definición de la vista); y siempre visible en la relación base correspondiente, y eventualmente sobre alguna otra vista definida sobre esa misma tabla base. Cuando se ejecuta un comando `INSERT` sobre una vista, los atributos de la relación base que faltan en la vista serán rellenados con valores nulos. Además se puede decir que:

- es posible insertar una tupla en una vista que viole la definición de la misma, ya que es ejecutada en la relación base, aunque su resultado no será visible desde la vista.

- las vistas pueden contener elementos duplicados que provengan de tuplas distintas de la relación base. En este caso hay que tener cuidado con las operaciones `DELETE` y `UPDATE`, ya que pueden tener efectos colaterales no deseables.

Capítulo 6

Dependencias Funcionales

Vamos a estudiar en detalle un tipo especial de restricción intrarelacional, la dependencia funcional. Veremos como estas restricciones generalizan el concepto de dependencias de clave, y que existen conexiones con los métodos y paradigmas utilizados en la lógica.

6.1 Definición de Dependencia Funcional: Implicación

En un capítulo anterior se presentaron las dependencias de clave; una manera diferente de interpretrar una depedencia de la forma $X \to Y$ es decir que, en una relación, dados unos valores para X determinan *funcionalmente* los valores para Y. Esto es, se puede decir que r satisface $X \to Y$, o que $X \to Y$ es válida en r, si cualesquiera dos tuplas de la relación r que sean iguales para X, también lo son para Y (para X,$Y \in V$, $r \in Rel(V)$). Por ello, una dependencia de clave es un caso especial de dependencia funcional. Por lo que se extiende la notación a las DFs.

Como un primer ejemplo podemos considerar $V = ABCDE$, y $r \in Rel(V)$:

	A	B	C	D	E
	1	1	1	1	1
r	1	0	1	1	1
	2	2	0	0	1
	2	3	2	0	1

Es fácil ver que r satisface la DF $AB \to C$, además de las DFs $A \to D$ y $D \to E$, por lo que tendremos el siguiente conjunto de DFs:

$F = \{AB \to C, A \to D, D \to E\}$

Podemos introducir la siguiente notación relativa a las DFs. Una DF se puede identificar con un nombre, $f : AB \to C$, o simplemente f. La parte izquierda (parte derecha) de una DF f se puede denotar como I_f (D_f), y el conjunto $I_f \cup D_f$ de todos los atributos que aparecen en f se puede denotar como $atr(f)$. Por ejemplo, en $f : AB \rightarrow C$, $I_f = AB$, $D_f = C$, $atr(f) = ABC$.

Tal y como se dijo en capítulos anteriores, las DFs pueden ser consideradas como unas *restricciones semánticas* para una relación. Ya que sólo pueden pertenecer a la relación aquellas tuplas que las cumplan.

Con respecto a las operaciones de actualización, las DFs representan una restricciones de integridad o de consistencia. Y se supone que son comprobadas por el DBMS en tiempo de ejecución, por lo que será deseable disponer de algoritmos eficientes que permitan efectuar esas comprobaciones. Teniendo esto en cuenta, podemos hacer las siguientes definiciones:

Sean F y G dos conjuntos de DFs sobre un conjunto de atributos V, esto es, $(\forall f \in F)\ atr(f) \subseteq V$ y $(\forall g \in G)\ atr(f) \subseteq V$.

- F y G se denominan *equivalentes*, de manera abreviada $F \approx G$, si el conjunto de relaciones que satisfacen F es igual al conjunto de relaciones que satisface G.

- F se denomina *redundante* si existe un conjunto $F' \subset F$ de DF's tal que $F' \approx F$.

Bajo la interpretación de que un conjunto de DFs representa una especificación semántica para un formato de relación dado, las siguientes dos nociones tienen un significado natural: dos conjuntos F y G serán llamados equivalentes si son identicos en sus especificaciones semánticas; y la redundancia significa que dado un conjunto de DFs F tiene más restricciones de las necesarias, algunas pueden ser omitidas sin que varie nada. Estas dos nociones se pueden reducir a otra más básica.

Sea F un conjunto de DFs sobre el conjunto de atributos V, y sea f una DF con $atr(f) \subseteq V$. Se puede decir que F *implica* f, $F \models f$, si toda relación sobre V que satisface F también satisface f.

Con esta noción, se puede concluir fácilmente lo siguiente.

Sean F y G conjuntos de DFs sobre V.

- $F \approx G \Leftrightarrow (\forall f \in F)G \models f \wedge (\forall g \in G)F \models g$

- F es redundante $\Leftrightarrow (\exists f \in F)\ /\ F - \{f\} \models f$

Para ilustrar lo anteriormente dicho, condiderese el conjunto de atributos $V = ABC$, y los dos conjuntos de DFs $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C, AB \rightarrow C\}$ y $G = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$.

Sea $r \in Rel(ABC)$ siguiente relación: r

A	B	C
1	1	1
2	0	1
0	0	1

Se puede comprobar fácilmente que r satisface tanto F como G, ya que $F \approx G$, y F es redundante.

6.2 Derivación: el Problema de la Pertenencia

En el último ejemplo, vimos como $G \models A \to C$, pero esto resultó posible gracias a que el número de DFs de G era pequeño. Pero esto no es normalmente así, ya que podemos tener muchas DFs que observar. Por ello tendremos que establecer algún mecanismo automático que nos permita determinar si una determinada DF es implicada por un conjunto de DFs dado.

Para llegar a ello tendremos que hacer la siguiente definición:

Sea F un conjunto de DFs sobre el conjunto de atributos V. Definimos la *clausura* de F, F^+ como el conjunto de todas posibles DF implicadas por F.

$F^+ := \{f /\, atr(f) \subseteq \cup_{g \in F} \wedge F \models f\}$

Resulta obvio el hecho de que $F \subseteq F^+$ se cumpla para todo conjunto F de DFs.

Con esta noción, la comprobación de implicación es lo mismo que determinar cuándo una DF dada f es miembro de la clausura de un conjunto F de DFs. Esta cuestión es llamada el *problema de la pertenencia*.

Los razonamientos que nos ayudaron a resolver el último ejemplo se han generalizado hasta llegar a la forma de reglas. Estas reglas nos permiten determinar los miembros de una clausura de un modo sencillo; como en casos anteriores, supongamos que todos los conjuntos de atributos utilizados son subconjuntos del conjunto V:

(A1) $Y \subseteq X \Rightarrow X \to Y$ (trivial)

(A2) $X \to Y \wedge Y \to Z \Rightarrow X \to Z$ (transitividad)

(A3) $X \to Y \wedge Z \subseteq W \Rightarrow XW \to YZ$ (aumento)

Si se aplican estas reglas al problema de calcular la clausura, se puede obtener lo siguiente:

(1) $Y \subseteq X \Rightarrow X \to Y \in F^+$ (incluso si $F = \emptyset$

(2) $\{X \to Y, Y \to Z\} \subseteq F \Rightarrow X \to Z \in F^+$

(3) $X \to Y \in F \wedge Z \subseteq W \Rightarrow XW \to YZ \in F^+$

Así, ahora podemos formalizar la obtención de nuevas DFs a partir de unas dadas.

Sea F un conjunto de DFs sobre V, y sea $X \to Y$ una DF con $X, Y \subseteq V$. $X \to Y$ se dice que es *derivable* de F, $F \vdash f$, si existe una secuencia $f_1, \ldots, f_n$ de DFs con las siguientes propiedades:

- f_n es de la forma $X \to Y$.

- $\forall i, 1 \leq i \leq n$, o $f_i \in F$ o f_i se puede generar aplicando las reglas (A1), (A2) y (A3) al conjunto $\{f_1, \ldots, f_{i-1}\}$.

De aquí se puede seguir inmediatamente que toda DF f derivable de algún conjunto F mediante la utilización de las tres reglas anteriores, también es implicada por F. Por lo tanto, el sistema de reglas formado por (A1), (A2) y (A3) se dice que es *sólido* (correcto), ya que cualquier derivación basada en él, produce DFs validas.

El siguiente teorema muestra que este sistema de reglas también es *completo*, en el sentido de que cualquier DF implicada por algún conjunto de DFs F, también es derivable desde F utilizando estas reglas.

Sea F un conjunto de DFs y f una DF. Entonces
$F \models f$ si y sólo si $F \vdash f$

Las reglas de derivación (A1), (A2) y (A3) son conocidas como *axiomas de Armstrong*, y hemos visto que son sólidas y completas, esto es, que las nociones de implicación y derivación coinciden para las DFs.

Los siguiente que vamos a mostrar es una manera normalizada de derivación. Para ello vamos a presentar unas reglas de derivación adicionales:

(A4) $X \rightarrow Y \wedge X \rightarrow Z \Rightarrow X \rightarrow YZ$ (adición)

(A5) $X \rightarrow Y \wedge Z \subseteq Y \Rightarrow X \rightarrow Z$ (proyección)

(A6) $X \rightarrow X$ (reflexividad)

(A7) $X \rightarrow YZ \wedge Z \rightarrow AW \Rightarrow X \rightarrow YZA$ (acumulación)

El sistema de reglas $\{A5, A6, A7\}$ es sólido y completo, y se conoce como *reglas RAP* (acronimo de Reflexividad, Acumulación y Proyección). Este sistema se utiliza para llevar a cabo derivaciones de una manera concreta, llamadas *derivaciones RAP*:

Una DF $X \rightarrow Y$ es un elemento de F^+ si y sólo si existe una secuencia de derivaciones $f_1, f_2, \ldots, f_n$ que cumple lo siguiente:

- f_1 es de la forma $X \rightarrow X$,
- F_n es de la forma $X \rightarrow Y$,
- $(\forall i, 2 \leq i \leq n-1) f_i \in F \vee f_i$ es derivable utilizando (A7).

Así, en una derivación RAP se utiliza primero la regla (A6), la regla (A5) se usa en último lugar, y entre ellas se aplica la regla (A7).

Considérese el siguiente ejemplo. Sea $F = \{AB \rightarrow E, BE \rightarrow I, E \rightarrow G, GI \rightarrow H\}$. Utilizaremos la derivación RAP para resolver el problema de pertenencia que se plantea con ¿$AB \rightarrow GH \in F^+$?

f_1 :	$AB \to AB$	(A6)	R
f_2 :	$AB \to E$	$\in F$	
f_3 :	$AB \to ABE$	(A7)	A
f_4 :	$BE \to I$	$\in F$	
f_5 :	$AB \to ABEI$	(A7)	A
f_6 :	$E \to G$	$\in F$	
f_7 :	$AB \to ABEIG$	(A7)	A
f_8 :	$GI \to H$	$\in F$	
f_9 :	$AB \to ABEIGH$	(A7)	A
f_10 :	$AB \to GH$	(A5)	P

Como es posible derivar $AB \to GH$ de F, se puede concluir que $AB \to GH \in F^+$.

De la misma manera que definimos el cierre de un conjunto de DFs F, podremos definir el cierre de un conjunto de atributos X:

Sea X un conjunto de atributos con $X \subseteq V$, y sea F un conjunto de DFs sobre V. Definiremos la clausura de X, X_F^+ como

$X_F^+ := \{A \in V / X \to A \in F^+\}$

El siguiente teorema nos muestra como se puede basar el cálculo del cierre de un conjunto de DFs F en el cálculo del cierre de un conjunto de atributos X.

Sea F un conjunto de DFs sobre V, y sean X,$Y \subseteq V$, entonces:

$X \to Y \in F^+ \Leftrightarrow Y \subseteq X^+$

Se puede elaborar un teorema que englobe las afirmaciones que se han ido haciendo sobre el tema de la pertenencia:

Sea V un conjunto de atributos, F un conjunto de DFs sobre V, y $f : X \to Y$ una DF con $f \in F$. Entonces las siguientes afirmaciones son equivalentes:

- $F \models f$ (implicación)
- $f \in F^+$ (pertenencia)
- $F \vdash f$ (derivación AA)
- $Y \subseteq X_F^+$ (cierre respecto a F)

Este cálculo del cierre se puede realizar mediante el siguiente algoritmo:

Algoritmo: CLAUSURA
Entrada: F conjunto de DFs, X conjunto de atributos
Salida: X^+

{

$Y := X$
mientras $(\exists S \to T \in F)\ S \subseteq Y \wedge T \not\subseteq Y$
hacer $Y := YT$

}

Resulta evidente, pues, que ahora se puede extender el algoritmo CLAUSURA para conseguir un test de pertenencia, añadiendo la comprobación de que la parte derecha de la DF sea un subconjunto del resultado final.

Algoritmo: PERTENENCIA
Entrada: F conjunto de DFs, $X \rightarrow Z$ DF
Salida: Si si $X \rightarrow Z \in F^+$; No en otro caso

{
pertenencia:= No
Y:= CLAUSURA(F,X)
si $Z \subseteq Y$ **entonces** pertenencia := Si
}

Como una primera aplicación del algoritmo PERTENENCIA, podemos resolver el problema de comprobar la equivalencia:

Algoritmo: EQUIVALENCIA
Entrada: F y G conjuntos de DFs sobre V
Salida: Si si $F \approx G$; No en otro caso

{
equivalencia:= No
si $(\forall X \rightarrow Y \in F)$ PERTENENCIA(G, $X \rightarrow Y$)
y $(\forall Z \rightarrow W \in G)$ PERTENENCIA(F, $Z \rightarrow W$)
entonces equivalencia := Si
}

Otra aplicación del problema de la pertenencia es el cálculo de una clave para un esquema de relación dado. El algoritmo que lo realiza es el siguiente:

Algoritmo: CLAVE
Entrada: $V = \{A_1, \ldots, A_m\}$ conjunto de atributos y F conjuntos de DFs sobre V
Salida: una clave $K \subseteq V$

{
K:= V
para i := 1 **hasta** m **hacer**
    si PERTENENCIA(F, $K - \{A_i\} \rightarrow V$)

entonces $K := K - \{A_i\}$

}

El algoritmo CLAVE elimina sucesivamente atributos desde la inicialización $K = V$ y comprueba si el conjunto resultante sigue implicando funcionalmente al conjunto entero V de atributos. El resultado del algoritmo es un conjunto K con $K \to V \in F^+$ y $K' \to V \notin F^+$ para todo $K' \subset K$.

Por ejemplo, sea $V = ABCDE$, y $F = \{AB \to C, A \to D, D \to E\}$. En este caso $(BCDE)^+ = BCDE$ por lo que A no puede ser eliminada de V. Además, $(ACDE)^+ = ACDE$, $(ABDE)^+ = (ABE)^+ = (AB)^+ = ABCDE$, por lo que el resultado es $K = AB$.

El algoritmo CLAVE calcula una clave arbitraria, dependiendo del orden de entrada de la secuencia de atributos. Puede suceder que existan varias claves para un esquema de relación dado, y es desable poder determinar cuál de ellas tiene la menor cardinalidad.

Por ejemplo, con $F = \{AB \to E, BE \to I, E \to G, GI \to HB\}$ si se aplica el algoritmo CLAVE con la secuencia $ABEIGH$ la clave encontrada será AIG; sin embargo para la entrada $AEIGHB$ la salida será AB.

Desgraciadamente existen estudios que indican que el problema de encontrar la clave con cardinalidad mínina no es solucionable de una manera eficiente.

6.3 Bases de Dependencia

Vamos a reconsiderar el problema de la redundancia de un conjunto de DFs. Buscaremos un conjunto de DFs libre de redundancia que nos aporte la misma información que el conjunto original. Para ello introducimos la siguiente notación:

Sean F y G dos conjuntos de DFs tales que $F \approx G$. Entonces F se dice que es un recubrimiento de G, y viceversa.

Por lo tanto, un conjunto dado F será redundante si existe un subconjunto $F' \subset F$ que recubra F. Con respecto a cuestiones relacionadas con el diseño de base de datos, el interés del recubrimiento no sólo está en la eliminación de la redundancia, sino también en la ordenación del conjunto de DFs, como se puede ver en la siguiente definición:

Sea F un conjunto de DFs, $f : X \to Y \in F$:

(1) f se llama *i-minimal* si no existe $X' \subset X$ tal que $F - \{f\} \cup \{X' \to Y\} \approx F$.

(2) f se llama *d-minimal* si $|Y| = 1$.

(3) F se llama *minimal* se todo elemento de F es *i- y d-minimal.*

(4) Un recubrimiento G de F se llama *base* (de dependencia) de F si G es no redundante y minimal.

Como ejemplo, se puede considerar $F = \{A \to BC, B \to C, AB \to D\}$. Entonces $F' = \{A \to B, A \to C, B \to C, AB \to D\}$ es un recubrimiento d-minimal de F, $F'' = \{A \to B, A \to C, B \to C, A \to D\}$ es un recubrimiento minimal de F, y $G = \{A \to B, B \to C, A \to D\}$ es una base de F.

Se puede hacer notar que el témino 'base' puede ser entendido como una analogía de las bases del espacio vectorial del álgebra lineal. Mientras que cada elemento del espacio vectorial puede ser representado como una combinación de los elementos de la base, en el caso de las DFs cada miembro del cierre de un conjunto F puede ser derivada a partir de los elementos de una base de F.

Claramente, la noción de base va más allá del concepto de eliminación de la redundancia. Intuitivamente, la derivación de una base significa, por ejemplo, con respecto a las operaciones de actualización que se tengan que comprobar menos DFs; y al mismo tiempo, la comprobación de cada una de las DFs puede ser más eficiente.

Ahora el problema se plantea a la hora de conseguir un algoritmo para calcular una base dado un conjunto de DFs.

Algoritmo: BASE
Entrada: $F = \{f_1, \ldots, f_n\}$ sobre V
Salida: una base G de F

{
/* hacer F d-minimal */
$G := \emptyset$
para i := 1 **hasta** n **hacer**
si $f_i : X \to Y$ **y** $Y = A_1 \ldots A_m$
entonces $G := G \cup \{X \to A_1, \ldots, X \to A_m\}$
/* con lo que $G = \{f_1, \ldots, f_k\}$ */

/* hacer G i-minimal */
para i := 1 **hasta** k **hacer**
si $f_i : X \to A$ **y** $X = B_1 \ldots B_s$
entonces para j :=1 **hasta** s **hacer**
si PERTENENCIA(G, $X - B_j \to A$)
entonces $X := X - \{B_j\}$

/* eliminar DFs redundantes */
para i :=1 **hasta** h **hacer**
si PERTENENCIA($G - \{f_i\}$, $f_i : X \to Y$)

entonces $G := G - \{f_i\}$

}

Como ejemplo se puede considerar $F = \{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, ACD \rightarrow B, D \rightarrow EG, BE \rightarrow C, CG \rightarrow BD, CE \rightarrow AG\}$. La partición de las partes derechas produce el siguiente conjunto de DFs: $G = \{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, ACD \rightarrow B, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow B, CG \rightarrow D, CE \rightarrow A, CE \rightarrow G\}$.

La reducción de las partes izquierdas se puede aplicar a $ACD \rightarrow B$, que se sustituye por $CD \rightarrow B$. Finalmente se estudia la redundancia, obteniendo la siguiente base: $G = \{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow B, CE \rightarrow G\}$.

Al igual que pasaba con el cálculo de una clave, con el algoritmo propuesto es fácil encontrar una base arbitraria para un conjunto de DFs, pero sin embargo, no se asegura que sea la base con el menor número de DFs posible.

Capítulo 7

Recuperación, Integridad y Seguridad de Datos

7.1 Recuperación de los Datos

7.1.1 Transacciones

Un objetivo del SGBD es preservar la consistencia de las bases de datos con las que trabaja. Sin embargo, puede que no sea posible garantizar un estado consistente de la base de datos después de cada instrucción individual del programa.

Por ejemplo, en una aplicación de un banco, las restricciones de integridad pueden requerir que la suma de las cantidades depositadas en todas las cuentas sea igual a una constante p. Consideremos ahora la transferencia de dinero desde una cuenta hacia otra. Esto se realizará en dos pasos, el primero de los cuales sustrae cierta cantidad de dinero de una cuenta, mientras que el segundo sumará la misma cantidad de dinero a la otra. Incluso si el estado de la base de datos fuera consistente antes de iniciar la ejecución de la transferencia, después del primer paso se perdería esa consistencia, ya que la suma de todas las cuentas será menor que esa constante p. Sin embargo, después del segundo paso, se habrá restablecido la consistencia.

Una actualización de la base de datos ha llevado a ejecutar dos operaciones sobre la base de datos. Esto, y la idea de que la ejecución de un programa pueda llevar temporalmente a la base de datos a un estado inconsistente, partiendo de un estado consistente, y que esas inconsistencias puedan ser toleradas mientras el programa esté en ejecución, se plasma en la noción de transacción. Así, una transacción no es necesariamente sólo una operación sobre la base de datos, sino que más bien estará compuesta por una secuencia de varias operaciones, que transformarán un estado consistente de la base de datos en otro estado consistente, sin tener que preservar la consistencia en todos los pasos intermedios. Esta propiedad de las transacciones se denomina *atomicidad.*

Está claro que no debemos permitir que suceda que una de las actualizaciones del ejemplo anterior se ejecute y la otra no, ya que dejariamos a la base de datos en un estado inconsistente. Por ejemplo, el sistema se puede caer entre las dos actualizaciones, o se puede dar un desbordamiento aritmético. Si el sistema soporta el procesamiento de transacciones no debemos temer por la integridad de la base de datos, ya que garantiza que si sucede algún fallo durante la ejecución de una transacción que implique actualizaciones sobre la base de datos, entonces todas aquellas actualizaciones serán desechas. Así, o una transacción se ejecuta en su totalidad, o es totalmente cancelada (como si no se hubiera ejecutado nunca). De este modo, una secuencia de operaciones que fundamentalmente no es atómica, puede hacerse que lo parezca desde un punto exterior de vista.

El componente de sistema que proveen de esta atomicidad se conoce como *gestor de transacciones*, y la clave de su modo de trabajar está en las operaciones COMMIT y ROLLBACK:

- La operación COMMIT señala la finalización con éxito de la transacción: comunica al gestor de transacciones que se ha completado con éxito una unidad lógica de trabajo, que la base de datos está otra vez en un estado consistente, y que todas las modificaciones realizadas por esa unidad de trabajo pueden ser ahora hechas permanentes (*commit*= cometer).

- La operación ROLLBACK, por contra, señala un fin de transacción insatisfactorio: le comunica al gestor de transacciones que algo ha ido mal, que la base de datos puede estar en un estado inconsistente, y que todas las actualizaciones realizadas por esa unidad lógica de trabajo deben ser deshechas. También sirve para arrepentirse de una actualización indeseable que fue llevada a cabo con éxito.

Llegados a este punto, vamos a ver cómo es posible deshacer una actualización. La solución está en que el sistema mantiene un diario (*log*) con todos los detalles de las modificaciones, en particular, los valores del objeto antes y después de la modificación. Asi, si llegara a ser necesario deshacer alguna modificación particular, el sistema puede utilizar la correspondiente entrada en el diario para restaurar el valor anterior del objeto modificado.

Ejecutar un COMMIT o un ROLLBACK establece lo que se llama un punto de sincronización. Un punto de sincronización representa el límite entre dos transacciones consecutivas; un punto en el que la base de datos está (o deberia estar) en un estado consistente.

Podemos ver que las transacciones no sólo son la unidad de trabajo, sino también la unidad de recuperación. Si una transacción exitosa ejecuta su COMMIT, entonces el sistema debe garantizar que sus actualizaciones permanecen en la base de datos, incluso si el sistema se cae en el siguiente instante de tiempo. Es bastante probable, por ejemplo, que el sistema pueda caer después de que el COMMIT se invoque, pero antes de que las actualizaciones sean escritas físicamente en la base de datos. Incluso si sucede esto, el procedimiento de rearranque del sistema podrá hacer firmes aquellas actualizaciones en la base de datos; se pueden descubrir los valores escritos examinando las entradas relevantes en el diario.

Para ello se siguen estas dos reglas:

- *Regla del Commit*: Para asegurar que cualquier transacción 'cometida' supere una caida inmediatamente posterior (esto es, que pueda se rehecha si fuera necesario), las modificaciones producidas por la transacción deben ser escritas en la memoria secundaria (log) *antes* de que esa transacción ejecute su COMMIT, escribiendo el OK en el fichero log.

- *Protocolo de Escritura adelantada del diario*: Para garantizar que una transacción activa pueda ser deshecha después de una caida, la imagen pre-transacción del objeto debe ser escrita en el diario *antes* de que la imagen post-transacción del objeto sea escrita en la base de datos. En el caso de que esto último tenga lugar, la transacción se habrá 'cometido'. Así, el procedimiento de rearranque puede recuperar el estado anterior a las transacciones que finalizaron exitósamente pero cuyas modificaciones no se grabaron físicamente antes de la caida, por lo que se dice que las transacciones son una unidad de recuperación.

7.1.2 Recuperación de Datos

Los sistemas debes estar preparados para recuperar, no sólo los fallos locales, tales como un desbordamiento dentro de una transacción, sino también los fallos globales, tales como un fallo de tensión en la CPU. Un fallo local, por definición, afecta sólo a la transacción en la que ocurre, y ya hemos tratado el modo de gestionarlos. Un fallo global, por contra, afecta a varias, puede que a todas, de las transacciones en ejecución en el momento del fallo, y por lo tanto tiene implicaciones a nivel de sistema. Vamos a considerar aspectos relacionados con la recuperació de fallos globales.

Estos fallos se pueden clasificar en dos categorias:

- *fallos del sistema* (ej. falta de tensión) que afectarán a todas las transacciones en progreso pero no que no dañará la base de datos. Un fallo del sistema también se conoce por *soft crash*.

- *fallos de almacenamiento* (ej. aterrizaje de las cabezas del disco) que causan daños a la base de datos, o a alguna porción de la misma, y que afectan a aquellas transacciones que actualmente están utilizando esa porción. Un fallo en el almacenamiento también se conoce por *hard crash*.

Fallos del Sistema

El principal aspecto que hay que tener en cuenta es que los contenidos de la memoria principal se pierden. No se puede conocer el estado de ninguna transacción en ejecución en el instante del fallo. Así, algunas transacciones deberán ser deshechas porque fueron interrumpidas. Pero otras deberán ser rehechas, ya que pudieron acabar satisfactoriamente antes del fallo pero no pudieron transferir sus actualizaciones a la base de datos física.

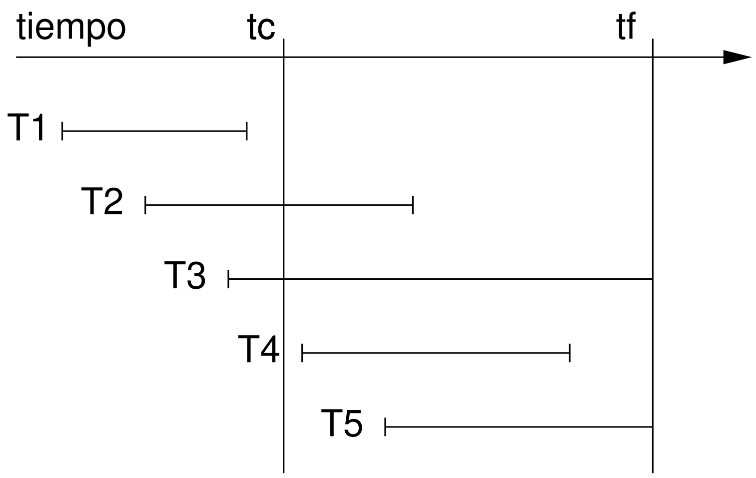


Figura 7.1: Cinco categorias de transacciones.

La cuestíón es: ¿Cómo conoce el sistema al rearrancar cuales son las transacciones que ha de deshacer (*undo*), y cuales rehacer (*redo*)? La respuesta está en que el sistema establece puntos de control (*checkpoints*) a intervalos. En cada punto de control el sistema (a) escribe físicamente todas las modificaciones pendientes producidas por las transacciones en la base de datos física, y (b) escribe un registro especial de control en el diario. Estos registros de control permiten obtener una lista de todas las transacciones que estaban en ejecución en el momento en que se tomó el punto de control. Para ver como se utiliza esta información ver la Figura 7.1:

- Ocurre un fallo del sistema en el instante T_f.
- El punto de control más reciente antes del fallo fue tomado en t_c.
- Las transacciones del tipo T_1 fueron completadas antes de t_c.
- Las transacciones del tipo T_2 comenzaron antes de t_c y finalizaron entre t_c y t_f.

- Las transacciones del tipo T_3 comenzaron antres de t_c pero no habían acabado en t_f.
- Las transacciones del tipo T_4 comenzaron despué de t_c y acabaron antes de t_f.
- Las transacciones del tipo T_5 también comenzaron después de t_c pero no habían acabado en t_f.

Ahora se puede ver que cuando el sistema rearranque, las transacciones del los tipos T_3 y T_5 deben ser deshechas, y las transacciones de los tipos T_2 y T_4 deben ser rehechas. Hay que darse cuenta, sin embargo, que las transacciones del tipo T_1 no deben ser tenidas en cuenta al rearrancar, ya que sus actualizaciones fueron físicamente escritas en la base de datos en el instante t_c como parte del proceso de control.

Fallos del Almacenamiento

Un fallo de almacenamiento lleva asociado que una parte de la base de datos sea destruida. La recuperación de un fallo de estas características implica la restauración de la base de datos desde una copia de seguridad y utilizar luego el diario para rehacer todas las actualizaciones realizadas después de que se hiciera la copia de seguridad. No será necesaria deshacer las modificaciones en ejecución ya que el fallo se ha encargado de deshacerlas (destruirlas).

Sobre las copias de seguridad (*backup copy*) hay que decir que existiran dos procedimientos asociados: *copy* o *dump*, que realiza la copia de seguridad de la base de datos en un dispositivo de almacenamiento masivo; y *restore*, que es el encargado de rehacer la base de datos después de un fallo de almacenamiento. Las copias de seguridad pueden ser totales, o *incrementales*, donde sólo se copian aquellas modificaciones realizadas desde la última copia de seguridad.

7.2 Seguridad e Integridad de los Datos

Los términos *seguridad* e *integridad* suelen aparecer juntos muy frecuentemente en contextos de bases de datos, aunque estos dos conceptos son bastante distintos. *Seguridad* se refiere a la protección de los datos contra la revelación, alteración o destrucción de los datos; *integridad* se refiere a la precisión o valided de los datos. En otras palabras:

- La seguridad tiene que ver con que los usuarios hagan sólo las cosas que tienen permitidas hacer.
- La integridad se encarga de comprobar que las cosas que intentan hacer los usuarios son correctas.

Existen algunas similitudes, sin embargo: en ambos casos el sistema debe cuidad de que ciertas restricciones no sean violadas; en ambos casos esas restricciones deben ser especificadas (normalmente por el ABD) en algún lenguaje adecuado, y deben ser almacenadas en el catálogo o diccionario de la BD; y en ambos casos el SGBD debe monitorizar las acciones de los ususarios para asegurarse de que las restricciones son observadas de hecho.

7.2.1 Seguridad: Consideraciones Generales

Existen numerosos aspectos relacionados con el problema de la seguridad, entre ellos podemos destacar los siguientes:

- Aspectos legales, sociales y éticos (ej, ¿tiene derecho legalmente una persona a solicitar información, digamos, sobre los datos médicos de un paciente?)
- Controles físicos (ej, ¿la sala del computador o de terminales está cerrada, o en otro caso suficientemente vigilada?
- Cuestiones de política de acceso (ej, ¿cómo decide la empresa que posee el sistema quién debe acceder a qué?).
- Problemas de operación (ej, si se utilizan *passwords*, ¿cómo se han de mantener secretos?, ¿cada cuánto tiempo se han de cambiar?)
- Controles del hardware (ej. ¿puede la CPU proveer de aspectos relacionados con la seguridad tales como protección del almacenamiento de las claves, o modo de operación privilegiado?)
- Seguridad del Sistema Operativo (ej. ¿borra el SO lo ficheros cuando ya no se necesitan?).
- Problemas específicos de la BD (ej. ¿tiene la BD el concepto de pertenencia de los datos?).

Por razones obvias, limitaremos nuestas consideraciones en éste último punto.

La unidad de datos para los propósitos de la seguridad varía desde la BD entera hasta el valor que ocupa una determinada posición en una columna de una tabla. Un usuario dado tendrá diferentes derechos sobre diferentes objetos (ej. puede estar autorizado a realizar SELECT sobre una tabla, y SELECT y UPDATE sobre otra, y así siguiendo). Y también, por supuesto, diferentes usuarios pueden tener diferentes derechos sobre el mismo objeto.

En el caso del SQL, nos encontramos con dos mecanismos más o menos independientes relacionados con la seguridad: (1) el mecanismo de vistas puede ser utilizado para ocultar datos sensibles a accesos un autorizados, y (2) el subsistema de autorización, el cual permite a los usuarios a los usuarios mantener derechos específicos sobre los objetos, y conceder/revocar derechos a otros usuarios.

Por supuesto, todas estas decisiones como qué derechos se han de conceder a qué usuarios son decisiones políticas, no técnicas. Por lo que quedan fuera de la jurisdición del SGBD; todo lo que el SGBD puede hacer es asegurar su cumplimiento una vez que sean formuladas. Para que el SGBD pueda ejecutar esta función correctamente:

- El resultado de esas decisiones deben ponerse en conocimiento del sistema (esto es hecho en SQL mediante las sentencias GRANT y REVOKE), y deben ser recordadas por el sistema (salvandolas en el catálogo, en forma de restricciones de seguridad).

- Debe haber alguna forma de comprobar si una solicitud de acceso cumple las restricciones de seguridad aplicables (solicitud de acceso = operación + objeto + usuario).

- El sistema debe conocer el origen de la petición, el usuario. Por esta razón, cuando el usuario entra en el sistema, tiene que introducir tambíen su password.

Seguridad en SQL Como todo sistema que incluye la posibilidad de definir vistas sobre una o más tablas base, el SQL puede utilizar la definición de vistas como un primer sistema de seguridad. Sim embargo, esto no nos permite especificar las operaciones que los usuarios autorizados pueden realizar sobre estos objetos, Esta función se realiza mediante las sentencias GRANT (*conceder*) y REVOKE (*revocar*).

Antes de que un usuario pueda ejecutar operación alguna, debe tener la autorización para ello, Por ejemplo, incluso para ejecutar una sentencia tan simple como SELECT * FROM S; , el usuario debe tener derecho de selección sobre la tabla S.

Si el ABD concede el derecho de crear algún objeto, una vista o una tabla, al usuario U, y supongamos que U crea ese objeto, el usuario U tendrá automáticamente todos los derechos sobre ese objeto, incluyendo el derecho de otorgar derechos a otros usuarios. Cuando nos referimos a todos los derechos, estamos hablando de todos los derechos que tengan sentido, ya que si un usuario U tiene derecho de SELECT sobre una tabla T, y crea una vista V sobre T, entones U no recibirá el derecho de UPDATE sobre V.

Los derechos que se pueden aplicar sobre tablas y vistas son:

- SELECT

- UPDATE (puede especificares la/s columna/s)

- DELETE

- INSERT

Y hay dos derechos más que se aplican sólo a tablas:

- ALTER (derecho a ejecutar ALTER TABLE)
- INDEX (derecho a ejecutar CREATE INDEX)

Aquí tenemos varios ejemplos:

- GRANT SELECT ON TABLE S TO PEPE;
- GRANT SELECT, UPDATE (ciudad, provincia) ON TABLE S TO LUIS;
- GRANT ALL ON TABLE S,T,U TO ANA, RAUL;
- GRANT SELECT ON TABLE S TO PUBLIC;
- REVOKE SELECT ON TABLE S TO PEPE;
- REVOKE ALL ON TABLE S,T,U TO ANA, RAUL;

Un último aspecto que merece la pena mencionar, es la opción GRANT. Si el usuario U1 quiere conceder a otro usuario el derecho de conceder/revocar derechos sobre el objeto A, se puede hacer con la opción GRANT:

- *U1*: GRANT SELECT ON TABLE S TO U2 WITH GRANT OPTION;
- *U2*: GRANT SELECT ON TABLE S TO U3 WITH GRANT OPTION;
- *U3*: GRANT SELECT ON TABLE S TO U4 WITH GRANT OPTION;
- *U1*: REVOKE SELECT ON TABLE S TO U2;

Entonces, la operación de revoque producirá una reacción en cascada en la que se revocarán los derechos que otorgó U2 a U3, y U3 a U4. No podemos pensar que con GRANT y REVOKE está solucionado el tema de la seguridad. Hay que tener en cuenta otros aspectos:

- No hay que suponer que el sistema es perfectamente seguro.
- Los datos deben estar encriptados,
- Es mejor hacer públicos los mecanismos de seguridad que suponen que no se conocen.

7.2.2 Integridad: Consideraciones Generales

Como ya explicamos antes, la integridad se refiere a la precisión o corrección de los datos en la base de de datos. Una restricción de integridad puede ser considerada como una condición que deben sarisfacer todos los estados correctos de la base de datos. Como ejemplo, en una tabla con los datos de personas se puede establecer la siguiente restricción: `FORALL persona (persona.fechTrab >= persona.fechNac + 18)`.

Si un usuario intenta ejecutar una operación que viole alguna restricción, el sistema debe rechazar la operación o ejecutar alguna acción compensadora en alguna otra parte de la base de datos que asegure que el resultado final sea un estado correcto. Por lo tanto el lenguaje que se utilice para especificar las restricciones de seguridad debe incluir, no sólo la capacidad de especificar condiciones arbitrarias, sino también la posibilidad de especificar acciones compensadoras donde sean apropiadas. Una posible sentencia de este lenguaje sería:

```
CREATE INTEGRITY RULE R1
ON INSERT persona.fechTrab,
UPDATE persona.fechTrab :
CHECK FORALL persona (persona.fechTrab >= persona.fechNac
+ 18)
ELSE REJECT;
```

Los sistemas actuales tienen poco desarrollado el control de integridad y somenten a los programadores a una gran carga ya que se ha estimado que la especificación de las restricciones de integridad de una base de datos típica, puede representar el 90 definición de la base de datos. Por lo que el área de trabajo relacionada con la integridad de las bases de datos es un importante campo de investigación y desarrollo.

Bibliografía

[1] Carlo Batini, Stafano Ceri, and Shamkant B. Navathe. *Diseño Conceptual de Bases de Datos. Un enfoque de entidades–interrelaciones*. Addison-Wesley/Diaz de Santos, 1994.

[2] C. J. Date. *An Introduction to Database Systems*, volume 1 of *The Systems Programming Series*. Addison Wesley, Reading, MA, fifth edition, 1990.

[3] Adoración de Miguel and Mario Pittini. *Concepción y Diseño de Bases de Datos. Del modelo E/R al Modelo Relacional*. Paradigma. RA-MA, Madrid, España., 1993.

[4] Ramez Elmasri and Shamkant B. Navathe. *Sistemas de Bases de Datos. Conceptos Fundamentales*. Addison-Wesley, 2 edition, 1997.

[5] Henry R. Korth and Abraham Silberschatz. *Fundamentos de Bases de Datos*. McGraw-Hill, Madrid, second edition, 1993.

[6] Jeffrey D. Ullman. *Principles of Database an Knowledgebase Systems*, volume 1 Classical Database Systems. Computer Science Press, Rockville, MD, 1988.

[7] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice Hall, 1997.

[8] Gottfried Vossen. *Data Models, Database Languages and Database Management Systems*. International Computer Science Series. Adison Wesley Publishing Company, Workingham, U.K., 1991.