

apuntes 2006-07

Bases de Datos 1

Eva Gómez Ballester

Patricio Martínez Barco

Paloma Moreda Pozo

Armando Suárez Cueto

Andrés Montoyo Guijarro

Estela Saquete Boro



**Dpto. de Lenguajes y
Sistemas Informáticos**

Escuela Politécnica Superior

Universidad de Alicante

<http://www.dlsi.ua.es/asignaturas/bd>

BIBLIOGRAFÍA BÁSICA

A continuación se presentan los libros de textos que se consideran manuales básicos para la asignatura. De cada uno de ellos, presentamos un breve resumen con la intención de facilitar al alumno una primera aproximación sobre la idoneidad de cada uno de ellos para cada una de las partes del temario de la asignatura.

CELMA03

Celma, M.; Casamayor, J.C.; Mota, L.
Bases de Datos Relacionales
Pearson-Prentice Hall, 2003.

DATE01

Date, C.J.
Introducción a los sistemas de bases de datos.
Addison-Wesley Publishing Company, Ed. 7, 2001.

ELMASRI02

Elmasri & Navathe
Fundamentos de Sistemas de Bases de Datos.
Addison-Wesley Publishing Company, Ed. 3, 2002

SILBERSCHATZ02

Silberschatz, S., Korth, H.
Fundamentos de Bases de Datos.
Mc Graw-Hill, Ed. 3, 2002

DRAE

Real Academia Española de la Lengua
Diccionario de la Lengua Española
Espasa, 2001

<http://buscon.rae.es/diccionario/drae.htm>

Comentarios a la bibliografía básica

CELMA03

Este libro está pensado para introducirse en la temática de las bases de datos relacionales mediante una presentación formal y rigurosa. En los capítulos 1 y 2 se introducen los conceptos de base de datos, sistemas de gestión de bases de datos y los principales modelos de datos. El capítulo 3 presenta los fundamentos del modelo relacional de datos desde la doble perspectiva algebraica y lógica, lo que permite introducir formalmente las estructuras de datos del modelo y sus operadores asociados mediante el Álgebra Relacional. Además proporciona la base formal lógica para introducir los lenguajes lógicos de interrogación, el Cálculo Relacional de Tuplas y el Cálculo Relacional de Dominios. En el capítulo 4 se introduce el lenguaje SQL, y en el capítulo 5 se profundiza en el concepto de sistema de gestión de bases de datos. El libro contiene numerosos ejemplos y ejercicios resueltos.

El libro ha sido realizado por un grupo de profesores de Bases de Datos de la Universidad Politécnica de Valencia y recoge su experiencia en la enseñanza de esta materia. Es por este motivo que se ajusta globalmente al enfoque docente propuesto en la asignatura, y tiene especial aplicación en las unidades 1, 2, 3, 4, y 7 del temario, aunque no incluye información sobre aspectos de normalización (unidad 5), ni sobre la organización física de una base de datos (unidad 6), y puesto que se basa en el modelo de datos relacional, no incluye información para el seguimiento de la unidad 8 del temario de la asignatura relativa al modelo entidad-relación.

DATE01

Este libro es la traducción en castellano de la séptima edición del texto *Date*, ed. 2000 que revisa actualiza y mejora ciertos aspectos contemplados en la versión previa *Date*, ed. 1993. El libro se organiza en seis partes principales. La parte I proporciona una amplia introducción a los conceptos básicos de las bases de datos, adaptándose al temario propuesto en la unidad 1 de la asignatura, e incluyendo información acerca de los sistemas de gestión de las bases de datos, adaptándose al propuesto en la unidad 7; la parte II aborda el modelo relacional de datos incluyendo información actualizada y revisada respecto a la 5ª edición, lo que lo hace más recomendable en este sentido para el alumno (unidades 3, 4 y 5 del temario); la parte III trata la teoría general del diseño de bases de datos incluyendo información acerca de la teoría de la normalización (unidad 5 del temario) y del modelo entidad-relación (unidad 8 del temario); la parte IV profundiza en los aspectos de recuperación y concurrencia y la V en otros aspectos entre los que se incluye el de la seguridad, lo que las hace recomendable a ambas para la extensión de los conocimientos que los alumnos han manejado en la unidad 7 del temario. Finalmente, la parte VI se dedica a la descripción del impacto de la tecnología orientada a objetos en los sistemas de bases de datos, lo que puede servir para iniciar al alumno hacia la enseñanza que se mostrará en la asignatura de Bases de Datos Avanzadas (optativa).

ELMASRI02

Tercera edición de otro de los libros clásicos en bases de datos. El texto contiene seis partes principales en las que abarca la gran parte de los aspectos necesarios para la enseñanza de las bases de datos. En concreto, la parte I se centra en los conceptos básicos de los sistemas de bases de datos, y la parte II en el modelo relacional de datos, siendo ambas especialmente recomendables para abarcar la enseñanza de la asignatura de *Bases de Datos I*. De este texto se destaca además el rigor y la extensión con los que trata cualquiera de los temas, y además las frecuentes referencias a ejemplos sobre sistemas de gestión de bases comerciales como Oracle y Microsoft Access. También se destaca la profundidad con la que se trata el modelo entidad-relación con un frecuente uso de ejemplos, así como la cobertura de otros aspectos relativos a tecnologías emergentes (novedad de esta tercera edición) sobre las bases de datos como los relativos a los almacenes de datos, tecnologías Web, multimedia y bases de datos distribuidas. Estos aspectos son brevemente introducidos durante la unidad 1 del temario y serán analizados con mayor profundidad en la asignatura Bases de Datos Avanzadas (optativa).

SILBERSCHATZ02

Este libro es una versión revisada, ampliada y corregida de un texto anterior que con el mismo título fue escrito por los mismos autores korth93. En esta revisión se ofrece un marco completo de los fundamentos y diseño de las

bases de datos, sus lenguajes de acceso y las diversas técnicas de implementación de bases de datos. Además incluye numerosos ejercicios después de cada tema que ayudan a la asimilación de los contenidos, así como frecuentes ejemplos para apoyar las diferentes explicaciones. Este libro se puede considerar como básico para el seguimiento de la asignatura ya que contiene un tratamiento elemental sobre todos y cada uno de los aspectos propios de las bases de datos, y además proporciona algunos aspectos más avanzados que pueden ser usados por el alumno como complemento a su enseñanza teórica o como introducción a otras asignaturas más avanzadas sobre las bases de datos.

Se destacan de este libro los capítulos 1 al 5 que contienen la introducción a las bases de datos, el modelo entidad-relación y posteriormente el desarrollo del modelo relacional encajando perfectamente en las unidades 1, 3, 4 y 8 del temario de la asignatura, aunque lo hace de una forma bastante elemental. Más interesantes resultan los capítulos 10 al 21 en los que se tratan los aspectos más avanzados de los sistemas de gestión de las bases de datos, desde el acceso físico a los datos pasando por aspectos como el procesamiento de las consultas, transacciones, concurrencia, seguridad, arquitectura, acceso cliente/servidor y bases de datos distribuidas, que permitirán al alumno profundizar en los temas mostrados en las unidades 6 y 7 del temario de la asignatura. Especialmente se destaca el capítulo 4 que realiza un estudio tanto teórico como práctico del lenguaje SQL lo que servirá al alumno para aclarar conceptos y problemas que se le presentan en las sesiones prácticas de la asignatura.

Sin embargo, el libro adolece de una falta de profundidad en el tratamiento del concepto de modelo de datos del que únicamente se presentan brevemente las diferencias entre los modelos de datos más tratados. Aún así puede ser considerado como una obra básica para la asignatura.

<i>I</i>	<i>introducción a las bases de datos</i>	<i>1</i>
I1.	introducción intuitiva	3
I2.	evolución de las técnicas de procesamiento electrónico de la información	8
<i>II</i>	<i>modelos de datos</i>	<i>15</i>
II1.	Sistemas de información	16
II2.	Conceptos y definiciones	19
II3.	Representación de un sistema de información	22
II4.	Cualidades de los modelos de datos	23
II5.	Clasificación de modelos de datos	23
<i>III</i>	<i>el modelo relacional</i>	<i>27</i>
III1.	introducción intuitiva	29
III2.	concepto de relación	30
III3.	representación de objetos	32
III4.	restricciones semánticas	43
III5.	operadores	55
III6.	otras características	56
III7.	conclusiones	58
<i>IV</i>	<i>álgebra relacional</i>	<i>61</i>
IV1.	Conceptos previos.	62
IV2.	definición informal de los operadores	65
IV3.	Resumen de los operadores del Álgebra Relacional	76
IV4.	ejemplos	77
IV5.	Referencia	83
<i>V</i>	<i>introducción al diseño de bases de datos relacionales</i>	<i>87</i>
V1.	Introducción	88
V2.	dependencia funcional	92
V3.	formas normales	94
V4.	forma normal de boyce-codd	97
V5.	Un ejemplo	98
<i>VI</i>	<i>la perspectiva lógica del modelo relacional</i>	<i>100</i>
VI1.	introducción	101
VI2.	cálculo de predicados de primer orden	101
VI3.	una base de datos relacional como una interpretación de un lenguaje de primer orden.	110
VI4.	fórmulas seguras	114
VI5.	cálculo relacional	118

<i>VII organización física de las bases de datos</i>	<i>125</i>
VII1.introducción	127
VII2.conceptos básicos	128
VII3.ficheros	131
VII4.implementación de bases de datos relacionales	140
 <i>VIII sistemas de gestión de bases de datos</i>	 <i>143</i>
VIII1. técnicas de base de datos	144
VIII2. arquitectura de un sistema de gestión de bases de datos	145
VIII3. el administrador de la bd	150
VIII4. componentes y funciones de un SGBD	151
VIII5. independencia, integridad y seguridad	153
VIII6. arquitectura cliente-servidor	156
 <i>IX introducción al modelo entidad-relación extendido</i>	 <i>159</i>
IX1. modelo entidad-relación	161
IX2. ejemplo 1	167
IX3. otros mecanismos de abstracción	171
IX4. ejemplo 2	172
IX5. deficiencias del modelo	174

I INTRODUCCIÓN A LAS BASES DE DATOS

Como primera introducción a la asignatura Bases de Datos I, se pretende efectuar un somero recorrido por los conceptos e ideas a tratar en ella, dando una visión superficial de las técnicas de bases de datos. Se expone, de una manera informal y no estructurada, el contexto y los contenidos de la asignatura.

I1. introducción intuitiva

La necesidad de manejar información

Pongamos como ejemplo un caso sencillo: queremos mantener de forma electrónica una lista con los discos que hemos comprado a lo largo de estos años. Tenemos un ordenador y un programa que nos permite almacenar la lista como se presenta a continuación.

autor	Título	format	año	tipo
COCTEAU TWINS	Victorialand	CS	86	Ambient
BJÖRK	Post	CD	95	Pop
BLACK CROWES	Amorica	CD	94	Rock
BLUE NILE,THE	High	CD	04	Pop
BOB MOULD		CD	96	Independientes
BLUR	Leisure	CD	90	Pop
BUD POWELL	Jazz Time	CD		Jazz
CANDY DULFER	Saxuality	CS	93	Fusión
CHURCH,THE	The Blurred Crusade	LP	82	Pop
COCTEAU TWINS	Blue Bell Knoll	CD	88	Ambient
CURVE	Pubic Fruit	CG		Independientes
COCTEAU TWINS	Milk And Kisses	CD	95	Ambient
CODE BLUE	Code Blue	LP	80	Pop
COP SHOT COP	Ask Questions Later	CD	93	Independientes
COMITE CISNE	Dulces Horas(Maxi)	LP	85	Pop
COMPLICES	La Danza De La Ciudad	CD	90	Pop
CONSTANCE DEMBY	Novus Magnificat	CD	86	Nuevas Músicas
CULT, THE	Sonic Temple	CD	89	Hard Rock
CURVE	Doppelgänger	CS		Nueva Psicodelia

La lista es muy sencilla, y está detallada por autor del volumen, título, año de publicación, formato en que lo tenemos disponible en nuestra discoteca (CD es disco compacto, CS es cassette, y LP es disco en vinilo), y una clasificación propia del estilo de música que contiene.

¿Para qué necesitamos almacenar los datos de esta manera? A lo largo del tiempo hemos ido adquiriendo más y más discos, y nos gusta intercambiar música con nuestros amigos (como se hacía antes, de forma inocente y legal, según lo que se entiende por legal hoy en día). Es más práctico dar una lista en papel, o enviarla por correo electrónico para que éste elija lo que más le guste, en vez de invitarle a casa y que él se lleve los discos viéndolos directamente en el estante; nuestro amigo también nos proporcionaría su propia lista para hacer nosotros lo propio.

Precisamente en este punto, cuando la cantidad de discos es grande, hacer dicha lista no es tan fácil. Podemos pensar que lo normal es comenzar a confeccionarla un día y anotar en ella las nuevas adquisiciones a medida que van llegando. Más tarde, si alguien nos la pide, podemos fotocopiarla y proporcionársela.

Sin embargo, es evidente que la lista no está ordenada bajo ningún criterio, salvo si nos hemos tomado la molestia de, cuando creamos la lista, anotar la información ordenada por autor, por ejemplo. No obstante, las nuevas entradas de la lista estarán desordenadas puesto que las anotamos al final de esa lista. Además, con la cantidad de discos que manejamos, es fácil que tengamos descripciones de discos repetidas, o mal catalogadas, o con el año equivocado; ¿qué hacemos?: ¿un borrón, escribir encima, escribirla a lápiz para poder borrar y rectificar?

Un día, un amigo nos pide una lista de los discos que tenemos, pero sabemos que lo que le gusta es el guitarreo y el ruido, lo que nosotros catalogamos

como *rock*, *duro*, o *independiente*. La única posibilidad es darle la lista y que él mismo se busque lo que le interesa.

Cansados de estas limitaciones decidimos utilizar el ordenador. Lo hacemos porque nos permite obtener listados ordenados por cualquier criterio, mantener la información actualizada, y corregir los errores fácilmente.

autor	titulo	formato	año	tipo
PRINCE	Batman	CD	89	Funk
PRINCE & THE NEW POWER GENERATION	Diamonds And Pearls	CD	91	Funk
PROPAGANDA	A Secret Wish	CD	85	Pop
PSYCHEDELIC FURS, THE	Mirror Moves	LP	84	Pop
QUINTO CONGRESO	Desaparecidos	LP	84	Pop
R.E.M.	Reckoning	LP	84	Rock
RADIOHEAD	The Bends	CD	95	Independientes
RAGE AGAINST THE MACHINE		CD	92	Duro
RAVEL	Bolero	CD	82	Clásica
RAY LYNCH	No Blue Thing	CD	89	Nuevas Músicas
RED HOT CHILI PEPPERS	Blood Sugar Sex Magik	CD	91	Rock
RED HOT CHILI PEPPERS	One Hot Minute	CD	95	Rock
REPLACEMENTS, THE	Please To Meet Me	CD	87	Rock
ROACH, BURMER, BRAHENY	Western Spaces	CD	89	Nuevas Músicas
ROBERT RICH/STEVE ROACH	Strata	CD	90	Electrónico
ROCKPILE	Seconds Of Pleasure	LP	80	Rock
ROGER DALTRY	Best Bits	LP	82	Pop
ROMANTICS, THE		LP	80	Pop
RONNIE WOOD	1 2 3 4	LP	81	Rock
ROUND MIDNIGHT (B.S.O.)		CD	86	Jazz
RY COODER	PARIS, TEXAS (B.S.O.)	CD	85	Folk, Texmex
SADE	Diamond Life	LP	84	Pop
SADE	Promise	CD	85	Pop
SARAH VAUGHAN	Jazz Time	CD		Jazz
SCREAMING BLUE MESSIAHS	Totally Religious	CS	89	Hard Rock
SCREAMING BLUE MESSIAHS, THE	Bikini Red	CS	87	Rock
SMASHING PUMPKINS	Siamese Dream	CD	93	Duro
SMASHING PUMPKINS	Mellon Collie And The Infinite Sadness	CD	95	Duro
SEAL		CD	91	Soul
SEAL		CD	94	Soul
SHAKESPEARE SISTERS	Hormonally Yours	CD	92	Pop
SILENCERS, THE	A Letter From St. Paul	LP	88	Pop

Figura 1.1. Ejemplo de bases de datos de discos

Además, esta información la podemos suministrar de cualquier forma: en papel mediante la salida por impresora, por correo electrónico, en un fichero de texto en un dispositivo de almacenamiento portátil o, en definitiva, en cualquier formato de intercambio. Podemos tener copias de seguridad por si se nos pierde la lista principal. Además, si queremos dar más datos descriptivos de nuestros discos, el ordenador nos da facilidades para hacerlo sin alterar la información anterior: sólo la definición de los listados se alterará para poder imprimir, a partir de entonces, los nuevos datos.

Herramientas para manejar la información

Ya hemos dicho que vamos a utilizar un ordenador y un programa para almacenar los datos y manejarlos. La primera opción en la que podemos pensar es un procesador de textos o una hoja de cálculo, donde la información es fácilmente accesible y modificable. Simplemente se trata de escribir la lista y guardarla en el disco duro. No obstante, el programa diseñado desde un principio para hacer lo que nosotros pretendemos es un programa de creación y manejo de bases de datos, un **sistema de gestión de bases de datos (SGBD)**. Una **base de datos (BD)**¹ es un conjunto de datos estructurados apropiadamente y relacionados entre sí (como, por ejemplo, nuestra lista de discos). Podemos tener tantas bases de datos almacenadas en nuestro disco duro como permita la capacidad del disco duro: la lista de discos, la agenda de teléfonos y direcciones de nuestros amigos, etc., son todas bases de datos diferentes; o podríamos tener relacionada los discos con la agenda de tal forma que sepamos en todo

¹ Para el profano en la materia es normal denominar al programa de gestión simplemente base de datos. Entiéndase que un sistema de gestión de bases de datos, el programa, puede manejar una o muchas bases de datos, uno o muchos conjuntos de información sobre un determinado tema.

momento a quien le prestamos los discos, con lo que todo sería una única base de datos.

El SGBD nos facilita un interfaz para introducir nuestra información desde teclado o cualquier otro periférico que lo permita, y procesar después esa información para obtener informes de cualquier tipo. Por ejemplo nos puede interesar tener un listado ordenado por autor y otro por tipo de música. Otro informe puede que sólo tenga la información del autor, título y año de publicación del disco.

La ventaja estriba en que la información sólo la hemos introducido una vez, y es el propio sistema de gestión de base de datos el que, según nuestras necesidades, se encarga de clasificar esa información cada vez que le pedimos un listado. Además, si nos hemos equivocado en el año de publicación de un disco, simplemente lo modificamos y en los siguientes listados ya saldrá corregido. Si quisiéramos borrar un disco, porque se nos haya perdido o roto, tampoco es un problema: simplemente, cuando el SGBD vaya a realizar un nuevo listado no se encontrará con ese disco entre los datps que maneja.

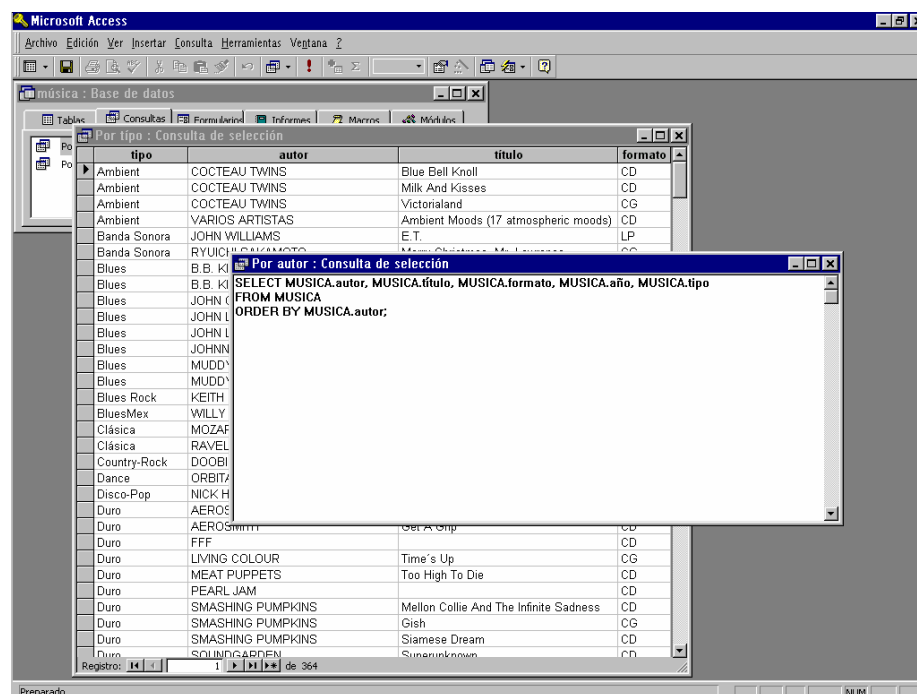


Figura 1.2. Ejemplo de consulta a la base de datos mediante una sentencia SQL.

Diseño del almacenamiento de la información

El fundamento de toda BD se encuentra en el análisis y el diseño. Al SGBD se le han de proporcionar dos cosas: los datos y la forma en que los vamos a almacenar. Es decir, un disco musical, para nosotros, es un objeto que tiene como características que lo diferencian de otro disco conceptos tales como la información del autor, el título, el año de publicación, el formato del disco y el tipo de música que contiene. Debemos, antes de nada, darle al SGBD estos conceptos con su correspondiente tipo de datos: si es un número, si es una cadena de caracteres, si es una fecha, etc. Una vez hecho esto, ya podemos introducir los datos de nuestros discos. De la misma forma, una vez que se han introducido los mismos, podemos realizar consultas sobre los datos almacenados basándonos en los objetos definidos.

I1.1. Sistema de Gestión de Base de Datos

Un SGBD es un programa de ordenador que facilita una serie de herramientas para manejar bases de datos y obtener resultados (información) de ellas. Además de almacenar la información, se le pueden hacer preguntas sobre esos datos, obtener listados impresos, generar pequeños programas de mantenimiento de la BD, o ser utilizado como servidor de datos para programas más complejos realizados en cualquier lenguaje de programación. Además, ofrece otras herramientas más propias de la gestión de BD como sistemas de permisos para autorización de accesos, volcados de seguridad, transferencia de ficheros, recuperación de información dañada, indización, etc.

En general, un SGBD es un software de BD que

- centraliza² los datos en un único “lugar” lógico al que acceden todos los usuarios y aplicaciones.
- es utilizable por múltiples usuarios y aplicaciones concurrentemente.
- ofrece visiones parciales del conjunto total de información, según las necesidades de un usuario en particular.
- posee herramientas para asegurar:

la **independencia** de datos: a varios niveles, permitiendo la modificación de las definiciones de datos sin afectar a las aplicaciones o esquemas que no utilizan esos datos.

la **integridad** de los datos: que los datos sean correctos en todo momento, de acuerdo con las especificaciones o reglas impuestas al sistema

la **seguridad** de los datos: que sólo las personas autorizadas puedan acceder a determinados datos y que sólo puedan efectuar las operaciones para las que han sido autorizados.

Hay muchos tipos de SGBD, pero la mayor parte de los utilizados comercialmente en la actualidad son **relacionales**, es decir, se basan en una cierta teoría o forma de representar los datos para implementar sus herramientas e interfaces, en este caso el **modelo relacional**. Entendemos por *representación de los datos* como la forma en que se presentan al usuario y que permiten ciertas operaciones para poder manejarlos.

De hecho, en estos SGBD, la información se presenta en forma de **tablas** (“relación” es el término formal), con columnas para las características de los objetos o conceptos que pretende representar la tabla, y filas para cada caso concreto o instancia de objeto. Existe un lenguaje considerado como estándar para manejar esas tablas, el *SQL*, que permite crear y modificar tablas, y consultarlas, introducir nuevos datos, modificar los ya almacenados, o borrarlos.

Al decir que un SGBD es relacional, estamos hablando de que, como mínimo, sigue todas las reglas y conceptos propuestos por el modelo relacional. El modelo relacional se basa en la teoría de conjuntos y es, por tanto, un modelo con un fundamento matemático. Este modelo maneja una estructura de datos, la **relación** (concepto matemático que se representa “físicamente” como una *tabla*), y unos operadores definidos sobre ella.

² Nos referimos a las bases de datos centralizadas. En otras asignaturas se profundiza en el concepto de bases de datos distribuidas.

Estos operadores se agrupan en lenguajes de especificación y manipulación, que nos proporcionan una sintaxis para poder dar ordenes al SGBD. A nivel teórico se proponen tres lenguajes equivalentes en cuanto a las cosas que pueden hacer: el **Álgebra Relacional**, el **Cálculo Relacional orientado a Tuplas** y el **Cálculo Relacional orientado a Dominios**. El álgebra está basado en el álgebra de conjuntos, mientras que los dos cálculos se basan en el cálculo de predicados de primer orden.

El SQL, que podemos decir que es el lenguaje “real” utilizado, tiene su origen formal en los anteriores. Además, aporta más operadores, necesarios para manejar eficazmente una BD relacional.

Como procedimiento de comprobación de la calidad de nuestros esquemas de base de datos relacionales veremos la teoría de la **normalización**. Ésta nos dice cuando una tabla es correcta: no contiene redundancia innecesaria, teniendo en cuenta las dependencias que existen entre las columnas de la tabla, y proporciona los métodos para corregir las deficiencias detectadas

No debemos olvidar que para almacenar datos en una BD primero debemos diseñarla, decir cuáles y cómo son las tablas (si hablamos exclusivamente de los sistemas relacionales). Para tal diseño se suelen utilizar otros modelos de datos más intuitivos y expresivos. Uno muy cercano al modelo relacional, el **Entidad-Relación Extendido**³ (EER) se utiliza para definir la BD sobre papel. Una vez que ya sabemos cómo representar la información nos queda, simplemente, crear las tablas e introducir la información.

Hemos hablado de dos de los muchos modelos de datos que existen. Brevemente, un **modelo de datos** es un conjunto de reglas y conceptos que sirve para describir un conjunto de información. Estos modelos de datos pueden ser de tipo gráfico (p.ej. el Entidad-Relación) o no, e incluir más o menos formas de representar hechos y objetos de la vida real. Al modelar un conjunto de datos, un sistema de información, lo que estamos haciendo es desechar la información no útil quedándonos únicamente con la que nos interesa, y representando lo más fielmente posible las interrelaciones entre los datos de ese sistema.

Pero, ¿qué había antes de las BD y los SGBD, y por qué nacen éstos? Los comienzos gestión administrativa ayudada por ordenador fueron fruto del desarrollo de los lenguajes de programación normales de aquella época, como COBOL, FORTRAN, BASIC, etc., y la información se manejaba mediante **ficheros convencionales de registros**. Los avances en hardware y software generaron un incremento exponencial del volumen de dato manejado, y el rendimiento de estos programas bajó al tiempo que los costes de mantenimiento crecieron de forma alarmante. Los datos se duplicaban innecesariamente, no había suficientes controles de seguridad frente a accesos no autorizados a la información, ni controles de calidad de los datos. Alterar las estructuras de datos, los ficheros, llevaba aparejado una ingente cantidad de trabajo para modificar todos los programas afectados. Como solución a estos y otros problemas se desarrollaron las **técnicas de Bases de Datos**.

³ También conocido como Entidad-Interrelación, por diferenciar el concepto *relación* en este modelo y en el relacional: en el primero es una dependencia o asociación entre objetos, mientras que en el segundo es la estructura de datos que almacena los datos.

I2. evolución de las técnicas de procesamiento electrónico de la información

I2.1. sistemas de información mecanizados tradicionales.

Veamos ahora, de una forma muy sucinta, cuales fueron los inicios de los sistemas de información mecanizados. Éstos han ido evolucionando hasta el presente al ritmo de las innovaciones tecnológicas tanto en hardware como en software. El hecho de disponer de una determinada tecnología siempre conlleva ciertas ventajas sobre los sistemas anteriores, y una serie de limitaciones impuestas por las posibilidades de la técnica de ese momento. Todo se traduce en una carrera en la que se solucionan problemas y carencias para mejorar la calidad, prestaciones, flexibilidad y seguridad de nuestro sistema de información, a la vez que la mayor exigencia y las nuevas necesidades de los usuarios plantea nuevos problemas no previstos o no abordables en un momento dado.

Los **sistemas de información basados en archivo convencional** se apoyan en las distintas organizaciones de fichero: secuenciales, directos (direccionamiento directo, calculado), indexados, invertidos... Estas organizaciones llevan aparejados unos métodos de acceso a los registros particulares: el acceso secuencial recorre todos los registros hasta encontrar el buscado; el indexado puede acceder en un solo paso al registro si estamos buscando por un campo clave. Para el manejo de estos ficheros los sistemas operativos llevan integradas rutinas que facilitan las operaciones básicas: inserción, borrado, modificación y recuperación.

Para entender mejor el origen y necesidad de las bases de datos es interesante analizar las características del sistema tradicional. La característica básica de estos sistemas es que los ficheros se diseñaban para un programa concreto. Esto los hace muy eficientes en principio, pero los problemas aparecen cuando hay que ampliar o modificar el sistema inicial. Puesto que la definición de los datos está dentro de cada programa de aplicación, cualquier alteración de la estructura de los ficheros que manejan nos obliga a recompilar todos los programas que utilicen esos ficheros, o bien construir nuevos programas que utilicen nuevos ficheros con información replicada o calculada en base a los antiguos.

La solución más rápida y fácil suele ser construir nuevos programas y ficheros con información redundante, más si se piensa en sistemas grandes donde cada departamento representa un conjunto de usuarios con una visión parcial de la Organización (la que es necesaria para su propio cometido), y por lo tanto, con un conocimiento parcial del sistema global.

Por ejemplo, en una Universidad, la sección de Personal, la secretaría del Centro y el Departamento tienen una visión distinta de los datos almacenados sobre un empleado docente, algunos comunes a todos (nombre, dirección, categoría, ...), pero otros únicamente útiles para una de ellos. La información sobre cuenta bancaria, estado civil o número de hijos es necesaria para Personal, pero no las asignaturas impartidas por el profesor o su horario. Esta distinta perspectiva de la organización es la que conduce en muchos casos a desarrollar aplicaciones separadas con ficheros propios.

En definitiva, todos ellos manejan información que pertenece a la organización, pero el desarrollo de los tratamientos de esos datos se realiza independientemente de los otros usuarios, de tal forma que cada aplicación es un objeto autónomo.

Puestas así las cosas, es fácil que nos encontremos, en un sistema de información mecanizado basado en archivo convencional, con los siguientes problemas:

- Redundancia de datos.
- Dependencia de los programas respecto de los datos.
- Insuficientes medidas de seguridad en tres aspectos:
 - control de accesos simultáneos
 - recuperación de ficheros
 - control de autorizaciones

Pasamos ahora a describir cada uno de estos puntos.

12.2. deficiencias de los sistemas basados en archivo convencional

Redundancia de datos.

El desarrollo de las aplicaciones no termina nunca. Las necesidades de la organización son cambiantes y evolucionan con el tiempo. Esto quiere decir que siempre se están creando nuevas aplicaciones y modificando las existentes. En un sistema de ficheros tradicional, cada programa lleva su propia definición de datos y maneja sus propios ficheros. Además, suelen ser varios los programadores que las realizan, bien en el mismo período de tiempo, o porque se van sustituyendo unos a otros.

El resultado fue, habitualmente, que muchos ficheros utilizados por diversos programas almacenaban la misma información. Y no solo eso, sino que la mayoría de las veces no recibían el mismo nombre ni coincidían los tipos de datos. Por ejemplo, un campo *ciudad* (cadena de 20 caracteres de longitud) en un fichero, se llamaba *localidad* en otro y podía tener una longitud mayor que la primera.

Evidentemente, es la falta de control sobre los datos que generaba la empresa lo que llevaba a estas situaciones. Una persona o equipo que se dedicara a supervisar todas las aplicaciones podría intentar mejorar este problema. En realidad, estos sistemas no son los adecuados para la tarea por lo costoso que resultaría tal control (y así aparecerán las técnicas bases de datos).

Aunque cada aplicación gestiona información propia, siempre hay datos comunes a varias aplicaciones. Al estar estos datos almacenados en ficheros independientes se produce redundancia dentro del sistema de información, lo que genera situaciones indeseables:

- *inconsistencia*: al tener almacenada la misma información en varios sitios, es difícil mantenerlos en el mismo estado de actualización (que en todo lugar tenga el mismo valor), pudiendo producir información incorrecta.
- *laboriosos programas de actualización*: no es lo mismo modificar el valor de un dato una sola vez que tantas veces como se halle duplicado.

- *mayor ocupación de memoria.*

Dependencia de los programas respecto de los datos.

En los sistemas clásicos la descripción de los ficheros usados por un programa, con información sobre formato de los registros, organización y modo de acceso, localización del fichero, etc., forma parte del código del programa.

Esto significa que cualquier cambio realizado en alguno de estos tres aspectos obliga a reescribir y recompilar todos los programas que utilicen el fichero modificado. Piénsese, por ejemplo, si se cambiara la organización de un fichero de secuencial a indexado, o que se añadiera un campo a un registro para una aplicación nueva, hecho éste que, en teoría, no tendría que afectar a las antiguas.

Podemos decir que los programas son completamente dependientes de los datos, lo que provoca:

- poca flexibilidad del sistema de información frente a futuras variaciones en los requerimientos de información.
- alto coste de mantenimiento del software.

Insuficientes medidas de seguridad:

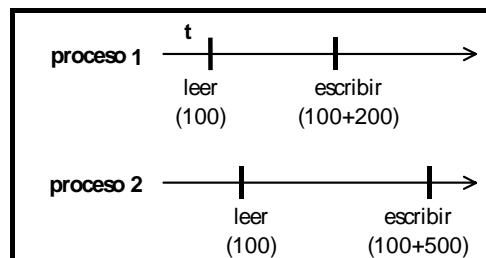
- control de accesos simultáneos

El acceso simultáneo de dos o más programas a unos mismos datos puede conducir a errores en la información almacenada.

Supongamos dos procesos que deben acceder al mismo dato, que en ese instante vale 100, y lo hacen concurrentemente, de tal forma que el primero suma al valor leído 200 y el segundo 500, por lo que finalmente deberíamos obtener un valor de 800 y almacenarlo.

Supongamos que el primer proceso llega antes que el segundo. Las respectivas transacciones comprenden una operación de lectura del dato almacenado y la posterior escritura del dato incrementado (la transacción está formada por dos operaciones atómicas).

Cuando el primero ha terminado de leer (y obtiene el valor 100) y antes de actualizar el dato (sumándole 100), el segundo proceso también efectúa una operación de lectura recuperando el mismo valor. Debido a la secuencia de operaciones en el tiempo, la actualización del proceso 1 se pierde puesto que, inmediatamente después, el proceso 2 modifica el mismo dato pero con una suma errónea. Es como si el proceso 1 nunca se hubiera ejecutado.



- recuperación de ficheros

En el caso de procesos de actualización incompletos o erróneos hace falta devolver los ficheros a un estado anterior correcto a partir del cual se puedan repetir, ahora correctamente, los procesos de actualización

rechazados. Tradicionalmente se recurre a copias de seguridad de los ficheros afectados.

- Control de autorizaciones

No todos los usuarios deben poder acceder a los mismos datos, por motivos de privacidad de la información, ni pueden acceder de la misma forma, por permisos a la hora de realizar recuperaciones, actualizaciones, etc. En los sistemas clásicos, al tener aplicaciones independientes, el volumen de información y el número de usuarios de cada una era reducido, pudiendo aplicarse estas medidas de seguridad a nivel humano.

A medida que fueron creciendo los sistemas se vio la necesidad de que el software dispusiese de mecanismos de seguridad adecuados a estos niveles.

En resumen, las características de los sistemas basados en archivo convencional adolecen de los siguientes problemas al incrementarse las exigencias y el volumen de datos:

- *Pobre control de los datos*: los datos pueden estar replicados innecesariamente, llamarse de distinta forma y tener distintas definiciones en diferentes ficheros.
- *Capacidades de manipulación de los datos inadecuadas*: las organizaciones de ficheros no son adecuadas para cierto tipo de operaciones que impliquen acceder a los datos para obtener información elaborada (o simplemente, en el mejor de los casos, que el criterio de búsqueda no está indexado).
- *Excesivo esfuerzo de programación*: en entornos de este tipo, la programación de nuevas aplicaciones obligaba a construir de nuevo las definiciones de fichero y rutinas de acceso en la mayoría de los casos.

Podemos decir que esta situación es la que “obliga” a replantear la forma de gestionar grandes volúmenes de datos, buscando principalmente la independencia de las aplicaciones respecto de la representación física de los datos almacenados. Nacen entonces las **técnicas de bases de datos**, que se abordan en el siguiente tema.

	1ª generación (Desde mediados de los 40 a mediados de los 50)	2ª generación (Desde mediados de los 50 a mediados de los 60)	3ª generación (Desde mediados de los 60 a mediados de los 70)	4ª generación (Desde mediados de los 70 a mediados de los 80)	5ª generación (Desde mediados de los 80 a mediados de los 90)
Modelos de datos			<ul style="list-style-type: none"> Modelo jerárquico Modelo red 	<ul style="list-style-type: none"> Modelo relacional 	<ul style="list-style-type: none"> Modelos semánticos M. Orientados a Objetos ...
Dispositivos de almacenamiento	<ul style="list-style-type: none"> programas + datos tarjetas perforadas Cintas magnéticas (1945) 	<ul style="list-style-type: none"> Discos magnéticos 	<ul style="list-style-type: none"> Tambores SGI Discos 	<ul style="list-style-type: none"> 	<ul style="list-style-type: none">
Productos			<ul style="list-style-type: none"> IDS de General Electric (1965) BOMP, DBOMP, CFMS de IBM TOTAL de Cincon (1971) IMAGEN de HP ADABAS de Software AG SYSTEM 2000 de MRI SGBD IMS/1 de IMB (1969) (Modelo jerárquico) Sistemas de red CODASYL (1969-71) IDS/2 de Honeywell DMS-1100 de Univac IDMS de BF Goodrich DBMS de Digital 	<ul style="list-style-type: none"> INGRES de la U. Berkeley (1973-77) System R de IBM (1974-78) INGRES de RTI (1980) SQL/DS de IBM (1981) ORACLE de RSI (1981) DB2 de IBM (1983) RDB de Digital (1983) 	<ul style="list-style-type: none"> ORION de MCC OpenOODB de TI IRIS de HP Gemstone de ServioLogic ONTOS de Ontologic O2 de O2 Tech. ObjectStone de Object Design CORAL de U. Wisconsin LDL de MCC
Acceso de datos	<ul style="list-style-type: none"> Ficheros secuenciales 	<ul style="list-style-type: none"> Ficheros de a. directo Ficheros indexados Ficheros hash 	<ul style="list-style-type: none"> Ficheros integrados Ficheros invertidos Ficheros secuencial-indexado 		
Avances destacados de la generación	<ul style="list-style-type: none"> Gestión de datos apoyado en aplicaciones 	<ul style="list-style-type: none"> Integración de información Independencia de datos SGBD prrelacionales 	<ul style="list-style-type: none"> Sistemas de gestión de bases de datos relacionales 	<ul style="list-style-type: none"> Sistemas de gestión de bases de datos postrelacionales 	

Visión diacrónica de la evolución de la tecnología de las bases de datos

CONCLUSIONES

Toda organización, sea pequeña o grande, tiene unas necesidades de información, bien en la forma tradicional de datos administrativos, bien en sistemas avanzados de tratamiento de información de todo tipo. De todos los datos que entran y salen de esa organización, en el formato que sea, unos son importantes y otros no tanto.

El objetivo de un analista es identificar la información importante y estructurarla de forma que sea útil para todos los miembros de la organización. Ese sistema de información puede ser mecanizado mediante herramientas informáticas y servir así a la productividad de la entidad.

En un principio, los sistemas de información a mecanizar eran sencillos y reflejaban más o menos exactamente el flujo administrativo de papel del exterior hacia la empresa, dentro de la misma empresa, y de la empresa hacia el exterior nuevamente. Para ello se utilizaban los lenguajes de programación disponibles, más o menos adecuados para la tarea, que manejaban ficheros organizados según lo permitía la tecnología del momento.

Pero pronto nuevas necesidades y expectativas hicieron que el mantenimiento y creación de aplicaciones informáticas, junto con el incremento masivo de la cantidad de datos a almacenar y tratar, se convirtiera en un cuello de botella debido a problemas de redundancia (e inconsistencia) de datos, deficientes medidas de seguridad, baja calidad de la información almacenada, y pérdidas de información por diversas causas. La tecnología del momento no era adecuada para sistemas de información en constante evolución y con unos requerimientos de rendimiento y fiabilidad cada vez más exigentes.

La aparición de las técnicas de bases de datos vino a solucionar gran parte de estos problemas.

BIBLIOGRAFÍA

[CELMA97]

[DATE2000]

[KORT87]

Prieto, A.; Lloris, A.; Torres, J. C.;
Introducción a la Informática
McGraw-Hill, 2ª edición.

II MODELOS DE DATOS

Antes de entrar a la descripción del modelo relacional, objetivo central de la asignatura, debemos definir lo que es un modelo de dato en general.

La calidad del análisis y diseño de un sistema de información que pretendemos mecanizar dependerá de los modelos de datos que utilicemos para cada una de las fases de desarrollo. Además, disponer de herramientas software basadas en modelos de datos adecuados a nuestra tarea nos hará más fácil y rentable el diseño y el mantenimiento.

Podemos decir que, en líneas generales, el diseño de un sistema de información, en lo que atañe a las bases de datos, tiene tres fases:

- **Diseño conceptual:** en la que se formalizan las estructuras que se observan en el mundo real produciendo lo que se denomina *Esquema Conceptual*.
- **Diseño Lógico:** en la que se estructura el conjunto de información de la fase anterior teniendo en cuenta el SGBD que se vaya a utilizar. En esta fase obtendremos el *Esquema Lógico*.
- **Diseño Físico:** en la que se estructuran los datos en términos de almacenamiento en los dispositivos del ordenador. Es lo que se conoce como *Esquema Interno*.

Para definir esas representaciones, es decir, los distintos esquemas, utilizaremos uno o varios **modelo de datos**, un formalismo o un lenguaje que nos permite representar una realidad con una mayor o menor riqueza de detalle.

Podemos considerar que los Modelos de Datos se utilizan, entre otras aplicaciones, para:

- Como herramienta de especificación, para definir tipos de datos y la organización de los datos de una BD específica.
- Como soporte para el desarrollo de una metodología de diseño de BD.
- Como formalismo para el desarrollo de familias de lenguaje de muy alto nivel, para la resolución de requerimientos y manipulación de datos.
- Como modelo soporte de la arquitectura de los SGBD.
- Como vehículo para investigar el comportamiento de diversas alternativas en la organización de los datos.

En una primera aproximación, un **modelo de datos** es un conjunto de conceptos y unas reglas de composición de esos conceptos que, combinados de alguna forma, son capaces de representar un sistema de información, tanto en su parte estática como dinámica.

Pero, hablando de bases de datos, ¿qué es antes: el modelo de datos o el SGBD? En realidad, el concepto de modelo de datos es totalmente independiente de las técnicas de BD. Es una herramienta que permite describir un determinado concepto o sistema. Los SGBD se apoyan en un determinado modelo de datos para poder estructurar la información a almacenar y gestionarla rentablemente. Esto implica unas ciertas reglas que el diseñador de la BD debe seguir para “informar” al SGBD de cómo se van a introducir los datos y que límites, restricciones o evolución pueden o deben tener esos datos.

No olvidemos, pues, que el modelo de datos es una “forma de hablar”, un lenguaje de representación que podemos utilizar independientemente de que se vaya a utilizar un ordenador o no, y que nos sirve para definir las propiedades esenciales de un sistema de información y, al mismo tiempo, poder comunicarnos con otras personas para explicarles esas propiedades.

El siguiente punto pretende ser el marco de partida para introducir los modelos de datos, herramientas que se utilizan para describir, mejor o peor, una realidad concreta. El concepto a desarrollar es el Sistema de información, la realidad que queremos modelar y convertir (seguramente) en un sistema de información mecanizado por ordenador.

II1. Sistemas de información

Uno de los pilares de cualquier organización es la información que necesita para su funcionamiento. Asimismo, el tratamiento de dicha información es una de sus ocupaciones básicas. Podemos decir que la Organización dispone en un primer momento de una cantidad de datos desordenados e inconexos sobre el entorno en el que se desarrolla y la actividad o actividades a las que se dedica. Es primordial, pues, ordenar e interrelacionar esos datos y obtener conclusiones de ellos, es decir, obtener información elaborada que permita tomar decisiones o conocer el estado de la Organización.

El tratamiento de la información (sea manual o electrónico) tiene como objetivo proporcionar esa información elaborada, de tal forma que sea correcta, se obtenga en el momento y lugar adecuado, para la persona autorizada y con el mínimo coste.

Si un **sistema** es un conjunto de “cosas” que, ordenadamente relacionadas entre sí, contribuyen a cumplir unos determinados objetivos, un **sistema de información** es un conjunto de elementos (en este caso datos), ordenadamente relacionados entre sí siguiendo unas ciertas reglas, que aporta al sistema objeto (la organización a la que sirve y que le marca las directrices de funcionamiento) la información necesaria para el cumplimiento de sus fines, para lo cual tendrá que *recoger*, *procesar* y *almacenar* los datos, facilitando la *recuperación*, *elaboración* y *presentación* de los mismos.

Dicho de otra forma, un sistema (y un sistema de información en particular) es un conjunto de componentes conectados de forma organizada. Dependiendo de cómo estén conectados el sistema se comporta o contribuye a sus fines de una manera concreta; si cambiamos esas interrelaciones, su comportamiento cambiará, al igual que si eliminamos o agregamos componentes.

Esos componentes del sistema de información, datos en nuestro caso, se representarán mediante papeles que se archivan o cualquier otro tipo de

almacén de información (¿una base de datos?). Así, si deseamos controlar el inventario de un almacén, el sistema físico que queremos representar es el local donde se almacenan y se retiran ciertas mercancías.

En realidad, el encargado podría pasarse todos los días por el lugar y comprobar si se ve el suelo del almacén, y decidir entonces que es necesario reponer existencias. Otra forma de organizar el almacén es construir un sistema de información en el que las entradas y salidas no sean bienes materiales sino notas de pedido, albaranes, etc., es decir, papeles que van de un archivador a otro. Sería la representación mediante la circulación de información del sistema físico *almacén*. En la Figura 2.1 podemos ver este ejemplo del almacén visto desde el punto de vista físico y desde el punto de vista del sistema de información.

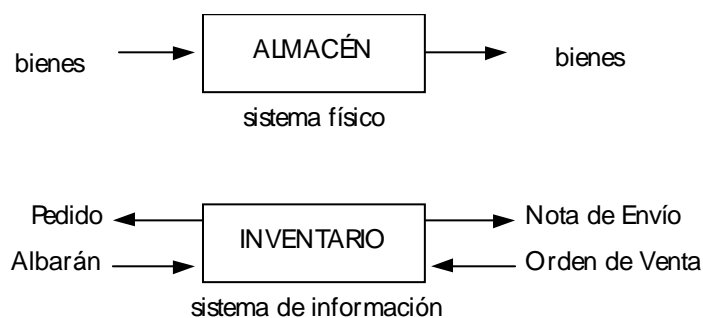


Figura 2.1. Diferencias entre un sistema físico y un sistema de información.

Una decisión a posteriori sería la de mecanizar este sistema de información o no. Es perfectamente posible que se dé el caso en el que la incorporación de un ordenador no reporte ningún beneficio en cuanto a eficiencia y rendimiento, e incluso que fuera perjudicial para una buena gestión.

Un *sistema de información mecanizado* (SIM) será aquel soportado por un ordenador. Si nos circunscribimos a este supuesto, podemos particularizar los componentes básicos de un sistema de información:

- **contenido:** los datos y su descripción.
- **equipo físico:** el ordenador soporte de la información.
- **equipo lógico:** sistema de gestión de bases de datos, sistema de comunicaciones, etc.
- **administrador:** la persona o equipo de personas responsables de asegurar la calidad y disponibilidad de los datos.
- **usuarios.**

Resumiendo, los SIM están compuestos de máquinas, programas que se ejecutan en esas máquinas, un responsable o responsables de su buen funcionamiento experto en temas de análisis, diseño e implementación de sistemas informáticos y, por supuesto, los usuarios finales que son los que indican cuales son las necesidades de la organización.

Los sistemas de información soportados por un ordenador han sufrido una rápida evolución. Desde el punto de vista de su funcionalidad podemos hacer una somera distinción de estos:

- SIM de procesos de transacción.
- SIM de ayuda a la toma de decisiones.

Los primeros tienen como objetivo el tratamiento de los datos necesarios para llevar a cabo las tareas rutinarias de la organización: nóminas, contabilidad, etc.; eran *sistemas de información para la gestión*.

Pero el desarrollo técnico en el campo de los ordenadores ha permitido exigir cada vez más prestaciones a los sistemas de información, que pasan a proporcionar una información más elaborada de ayuda a la toma de decisiones: son los *sistemas de información para la ayuda a la toma de decisiones*. Pensemos en una sistema de información de diseño asistido por ordenador (CAD); la máquina será la encargada de simular el comportamiento de un determinado objeto (el diseño de un avión, por ejemplo) en distintas condiciones ambientales, información ésta que nos dará una base para decidir si es factible su fabricación o si, por el contrario, no debemos seguir adelante con el proyecto. Los almacenes de datos (más conocidos como *Data Warehouses* y las aplicaciones OLAP constituyen hoy en día el núcleo de estos sistemas de apoyo a la toma de decisiones). Estos sistemas son la parte central de la asignatura optativa *Bases de Datos Multidimensionales*.

II1.1. Desarrollo de un sistema de información mecanizado.

Para construir un sistema de información se parte de una definición de los posibles usuarios, sus necesidades y las fuentes de información, pasando a dar respuesta a un conjunto de temas entre los que destacan los relativos a la organización de los datos, que determinan en gran medida el rendimiento, flexibilidad y fiabilidad de todo el sistema. Así, cualquier sistema de información pretende, por medio de una abstracción del mundo real, representar con un conjunto de datos estructurado toda la información necesaria para el cumplimiento de los objetivos de una organización, para lo que se deben seguir distintas fases apoyadas en métodos y reglas.

En primer lugar, habrá que aislar la parcela del mundo real objeto de estudio, identificando objetos con sus propiedades, las relaciones entre ellos, así como su semántica asociada.

A continuación y por medio de un proceso de abstracción, intentaremos obtener una imagen (representación) de esta parcela del mundo real. Es lo que se llama *modelado*, proponer un esquema bajo un determinado sistema de representación que simule el comportamiento de la parte de la realidad en estudio. Obtendremos así las *estructuras* que alberguen nuestros datos y los *procesos* (operaciones) que las han de afectar para obtener la información elaborada final.

Por último, habrá que organizar el conjunto de información definido en la etapa anterior para almacenarla en un soporte informático, lo que exige el conocimiento de técnicas cada vez más sofisticadas entre las que se encuentran las técnicas de bases de datos.

En definitiva, siguiendo el ciclo de vida clásico del software podemos diferenciar tres fases a la hora de desarrollar un sistema de información, en nuestro caso mecanizado, a saber:

- *Análisis*: investigación y modelado
- *Diseño*: lógico y físico
- *Implementación*: programas, carga de datos, pruebas.

En la fase de análisis se realizan labores de recogida de requerimientos, y se obtiene un modelo no influido por un sistema mecanizado concreto (modelo conceptual), que representa, lo más fielmente posible, el conjunto de datos en estudio y las interrelaciones que hubiere entre ellos, así como los procesos que los afectan. En definitiva vamos a describir, mediante un formalismo adecuado, nuestro sistema físico.

En la etapa de diseño trasladamos las ideas obtenidas en la fase anterior a un modelo comprensible por el ordenador, el diseño lógico y el físico, que representan sucesivos acercamientos al nivel más bajo de detalles de almacenamiento.

Con la implementación introducimos en la máquina el resultado del diseño y los datos necesarios para la inmediata explotación de los mismos, tras las correspondientes pruebas de fiabilidad y rendimiento. En la Figura 2.2. podemos ver un resumen del proceso de diseño de una base de datos desde el punto de vista del ciclo de vida de evolución del software.

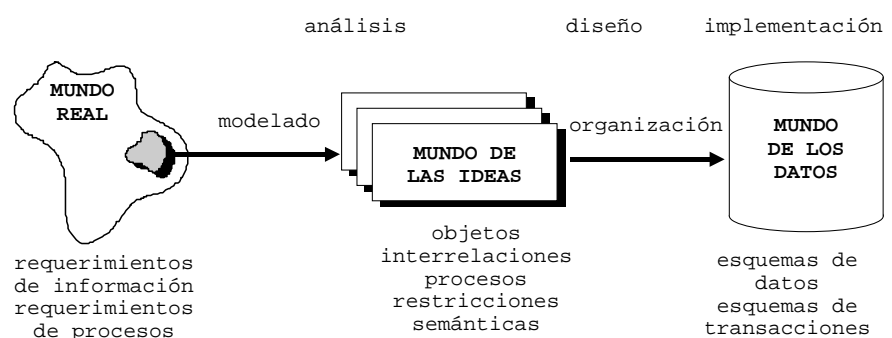


Figura 2.2. Diseño de una base de datos desde el punto de vista del ciclo de vida de evolución del software

II2. Conceptos y definiciones

Propiedades de un sistema de información

En un sistema de información encontraremos propiedades estáticas, dinámicas y restricciones de integridad.

- Las propiedades estáticas se refieren a los datos en si, qué información es la que se maneja en el sistema, qué valores puede contener el sistema, y cómo dependen unos de otros. Son propiedades que no varían en el tiempo.
- Las *dinámicas* hacen referencia a la evolución de esos datos en el tiempo, a las operaciones que se les aplican, más concretamente transacciones (secuencias indivisibles de operaciones simples).
- Las *restricciones de integridad* son reglas utilizadas para definir las propiedades estáticas y dinámicas. Dicho de otro modo, es el mecanismo por el que establecemos cuales son los valores válidos de los objetos de nuestro sistema en cada instante.

La mayoría de ellas se podrán expresar adecuadamente con los conceptos que maneja el modelo de datos. Al estructurar la información de una cierta manera y al especificar las transacciones ejecutables estamos estableciendo ya unas reglas que cumplirá el sistema. No obstante, hay propiedades imposibles de expresar con los mecanismos anteriores que necesitan una definición adicional mediante un formalismo de especificación de aserciones. De una forma hasta cierto punto coloquial, habitualmente se habla de restricciones de integridad refiriéndose únicamente a las segundas, ya que las primeras no necesitan de esa especificación aparte.

Resumiendo, en el proceso de análisis y diseño de un sistema de información obtendremos finalmente un conjunto estructurado de datos, un conjunto de operaciones a realizar sobre ellos, y una serie de afirmaciones adicionales sobre las combinaciones de valores que son posibles dentro de nuestro sistema. Para reflejar todo ello necesitamos un formalismo, un “lenguaje” que nos permita estructurar esa información y definir esas operaciones para satisfacer las demandas de los usuarios finales. Este formalismo es el *modelo de datos*.

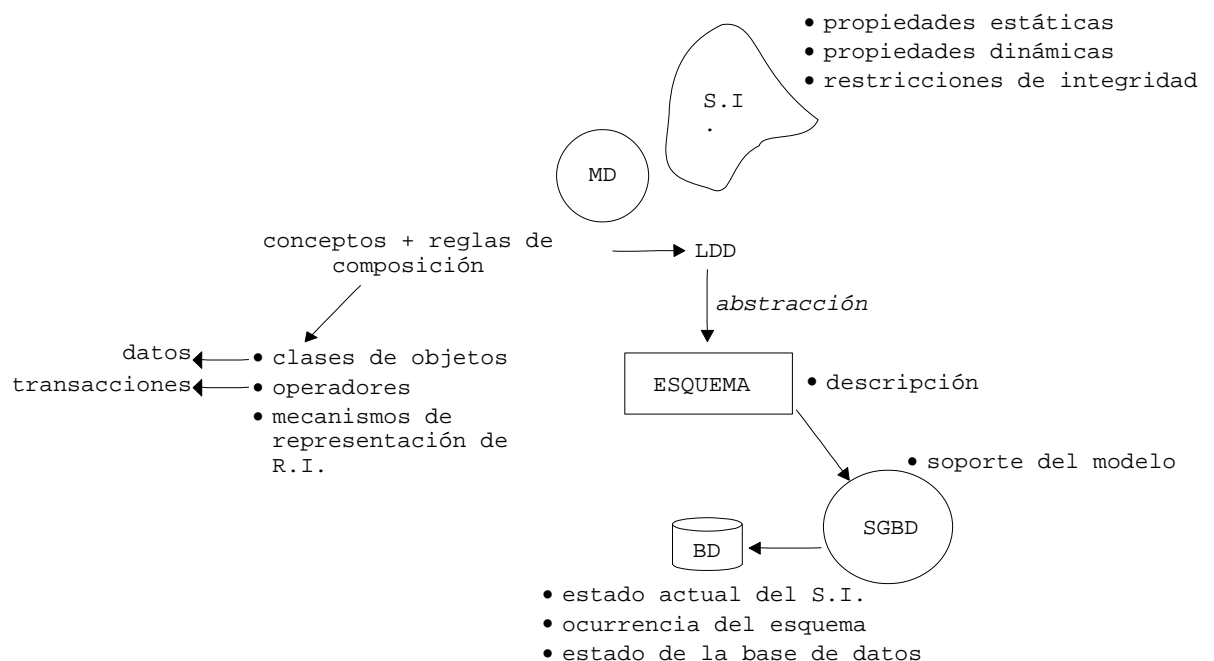


Figura 2.3. Ejemplo del proceso de diseño de una base de datos

Modelo de datos

Definimos un **modelo de datos** como la herramienta intelectual que nos permite estructurar los datos de forma que se capte la semántica de los mismos. Éste nos ofrecerá un conjunto de *conceptos* y *reglas* que nos permitirán representar, con mayor o menor fidelidad, un conjunto de datos interrelacionados y operaciones sobre los mismos, a los que afectan unas restricciones que han de cumplir en todo momento. Cuanta más información sobre los datos podamos representar dentro del mismo, *más expresivo* será el modelo de datos (y por la tanto más deseable).

Esquema

La aplicación de un cierto modelo de datos a una realidad nos dará como resultado un **esquema**, una representación de esa realidad que podrá ser de tipo gráfico o lingüístico y que describirá un conjunto de objetos e interrelaciones entre ellos, un conjunto de operaciones (combinaciones de

insertar, borrar, modificar y consultar) a aplicar sobre los primeros, y todas las restricciones semánticas que afecten al sistema.

Lenguajes de definición y manipulación de datos

Para aplicar un modelo de datos deberíamos disponer de un *lenguaje de definición de datos* (LDD) y de un *lenguaje de manipulación de datos* (LMD). Con el LDD describiríamos las estructuras en las que se almacenarían los datos, y con el LMD las transacciones a efectuar para obtener información elaborada a partir de esos datos. Podemos decir, pues, que un modelo de datos se compone de dos partes estrechamente relacionadas: las **estructuras de datos**⁴, que representan las propiedades estáticas del modelo, y la **especificación de transacciones**, que hace lo propio con las dinámicas.

No todos los modelos tienen en cuenta los dos tipos de propiedades, centrándose bien en uno o en otro. De hecho, los más implantados de forma comercial se basan casi exclusivamente en la parte estática, seguramente por ser la menos costosa de solucionar.

¿Quién es el encargado de proporcionar los lenguajes antes mencionados y de gestionar los datos estructurados en función de un determinado modelo de datos? El *sistema de gestión de bases de datos* es la herramienta software que soporta a un modelo de datos o, dicho de otro modo, todo SGBD debe tener un modelo de datos subyacente que permita describir los datos de una forma concreta.

Disponemos, pues, de un conjunto de reglas, aplicables mediante el LDD, para la generación de esquemas. El **esquema de una base de datos**, resultado de la utilización del LDD, es la definición de las estructuras de acuerdo a cada modelo. Se usa para representar las propiedades estáticas según el modelo que se utilice. En algunos modelos se distinguen dos subconjuntos de este conjunto de reglas: uno para la definición de estructuras en sí, y otro para la definición de restricciones de integridad estáticas.

Los LMD pueden ser **navegacionales** o **de especificación**. Los primeros se basan en la utilización interna de punteros, por lo que los operadores seleccionan un único objeto por la posición lógica que ocupa, es decir, se recupera la información partiendo del valor actual del puntero (del objeto al que está apuntando). En otras palabras, debemos recorrer la estructura desde el lugar en el que nos encontremos hasta llegar a la información que precisamos.

Los LMD de especificación, sin embargo, recuperan los datos en función de qué es lo que estamos buscando, indicando explícitamente qué datos precisamos y qué condiciones han de cumplir, pero dejando al SGBD la tarea de cómo obtiene esa información.

Sistema de Gestión de Bases de Datos

Un *Sistema de Gestión de Bases de Datos* (SGBD) es el software que implementa las herramientas asociadas a un modelo de datos.

Estados de la BD

Diremos que para un esquema de BD existen muchas **ocurrencias de base de datos**, cada una de las cuales tiene las mismas estructuras y obedece a las mismas restricciones de integridad. La estructura de la base de datos es siempre la misma o sufre alteraciones cada bastante tiempo pero las operaciones de manipulación de datos son constantes y, por tanto, los datos

⁴ En un sentido amplio; piénsese que en el análisis y diseño orientados a objeto no se habla de estructuras sino de objetos.

contenidos en la base de datos cambian continuamente. Tradicionalmente, los términos **ocurrencia del esquema** y **base de datos** se suelen utilizar indistintamente.

Un **estado de la base de datos** está definido en un instante de tiempo por la descripción de la misma y la información almacenada; generalmente se entiende que el resultado de una transacción o una simple operación atómica (insertar, borrar, ...) generan un nuevo estado de la BD (se ha modificado, de alguna forma, la BD con respecto al estado inmediatamente anterior).

Evidentemente, las operaciones que provocan la transición de un estado de BD a otro nunca varían la estructura de la BD (el esquema).

II3. Representación de un sistema de información

Resumiendo de forma intuitiva todo lo expuesto en las secciones anteriores, todo modelo de datos utiliza los siguientes mecanismos de abstracción para la construcción de las clases de objetos apropiadas:

- *Clasificación*: definir un concepto como una clase de objetos
- *Agregación*: definir una nueva clase de objetos a partir de otras clases
- *Generalización*: definir subtipos de una clase

Igualmente, para expresar las restricciones aplicables a las clases de objetos en particular, y al sistema en general, sean de tipo estático o dinámico se utilizan las:

- *restricciones de dominio*: definen el conjunto de valores (escalares o complejos) agrupados en una clase de objetos.
- *restricciones de identificación*: no hay dos objetos dentro de una clase de objetos iguales.
- *restricciones de correspondencia entre clases*: restricciones de cardinalidad, de dependencia de identificador, de existencia y de cobertura de las generalizaciones.

Características dinámicas del sistema de información: restricciones dinámicas.

La parte dinámica de un sistema hace referencia a la variación del contenido del sistema de información (o incluso su evolución como esquema) a través del tiempo. Visto de una forma simplista, su comportamiento frente a inserciones, borrados, modificaciones y consultas de los datos en él almacenados.

Tal y como estudiaremos en el tema 3 dedicado al modelo relacional, éste es en concreto, un modelo que como todos los denominados clásicos y algunos de los semánticos, únicamente es capaz de representar la parte estática de un sistema de información. Posteriores extensiones o nuevos modelos de datos más potentes han incorporado herramientas para representar el

comportamiento en el tiempo de los datos, esto es, las *restricciones dinámicas* del sistema.

Es evidente que siendo un tema importante este último, tradicionalmente se ha dejado de lado por su dificultad de definición e implementación, y no va a ser tratado en más profundidad en esta asignatura.

II4. Cualidades de los modelos de datos

A la hora de evaluar un modelo de datos debemos fijarnos en los siguientes puntos:

- **Expresividad:** cuantos más mecanismos o conceptos de representación tenga un modelo mayor será la cantidad de propiedades del sistema de información que pueda captar, y menor el uso de aserciones en forma de restricciones de integridad que no se pueden reflejar directamente sobre el esquema.
- **Simplicidad:** también es deseable que el modelo sea simple para que los esquemas sean fáciles de entender por terceras personas. Debe llegarse, pues, a un equilibrio entre la potencia del modelo mencionada en el punto anterior y esa simplicidad deseable.
- **Minimalidad:** cada concepto tiene un significado distinto de los demás conceptos utilizados en el modelo de datos; no se puede expresar un concepto en función de otros.
- **Formalidad:** todos los conceptos del modelo tienen una interpretación única, precisa y bien definida. Puesto que el esquema pretende ser una especificación formal del sistema de información a representar, esta cualidad permitiría el tratamiento matemático de sus conceptos.

Si el modelo utiliza un lenguaje de definición gráfico, también tendremos en cuenta:

- **Compleción gráfica:** un modelo es gráficamente completo si todos sus conceptos poseen representación gráfica.
- **Facilidad de lectura:** que los símbolos gráficos sean fácilmente distinguibles unos de otros.

II5. Clasificación de modelos de datos

A medida que han ido evolucionando el software y el hardware, las posibilidades y las demandas de los usuarios han ido creciendo; paralelamente, los modelos de datos fueron enriqueciéndose y salvando carencias de sus predecesores. Cronológicamente, podemos clasificarlos de la siguiente forma:

- **Primitivos:** basados en sistemas de ficheros convencionales
- **Clásicos**

- Jerárquico (el más conocido, IMS: *IBM*)
- Red (CODASYL)
- Relacional (desarrollado por *E. Codd*)
- **Semánticos⁵**
 - EER (Entidad-Relación Extendido: *Chen*)
 - RM/T (Relational Model/Tasmania: *Codd*)
 - Semántico General
 - Orientado a Objetos
 - Modelo Funcional

Se dice que tanto primitivos como clásicos están basados en registros, mientras que los semánticos se apoyan en la filosofía Orientada a Objetos.

Los **modelos de datos primitivos** se usaron durante la década de los 70, cuando aun no se utilizaban las técnicas de bases de datos. Los objetos se representaban como registros organizados en ficheros, y las relaciones mediante referencias explícitas a otros registros en algún campo del mismo. Los lenguajes de manipulación dependen por entero de la organización física de los datos, y las operaciones básicas son la lectura y la escritura.

Para garantizar, o al menos mejorar, la independencia de los aplicaciones frente a los datos aparecen los primeros SGBD, basados en lo que ahora llamamos **modelos de datos clásicos**. Los primeros en aparecer fueron el jerárquico y el red de CODASYL, cuyos nombres muestran cual es la estructura de datos subyacente en los modelos. Los objetos siguen siendo representados por registros pero las relaciones entre objetos se expresan, con ciertas limitaciones implícitas del modelo, mediante la estructura en que se basan. Los lenguajes de manipulación de datos son navegacionales.

Sin embargo, y dentro todavía de esta generación, la aparición del Modelo Relacional provocó la representación de los objetos como relaciones (tablas en su denominación informal), cuyas tuplas (filas en su denominación informal) identifican a ocurrencias del objeto patrón, y la vuelta a las referencias explícitas (unos atributos que relacionan un objeto con un segundo por comparación de valores iguales en uno y otro) para expresar las interrelaciones. La estructura de datos más simple y la aparición de lenguajes de especificación totalmente declarativos ha hecho de este modelo el más ampliamente utilizado en los SGBD comerciales actuales.

Uno de los grandes problemas que plantean los SGBD comerciales actuales, en general, es que no soportan modelos con la suficiente expresividad como para dejar libre al diseñador de sistemas de información de molestas tareas de administración de datos a bajo nivel. Es práctica habitual utilizar en la confección del EC modelos de datos con un fuerte potencial semántico para traducirlo después a modelos que si tienen un software comercial disponible, pero muy probablemente más pobres semánticamente.

Por eso, la aparición de los **modelos de datos semánticos** está justificada por la pretensión de aumentar la capacidad expresiva de los modelos clásicos incorporando conceptos y mecanismos de abstracción no contemplados en los anteriores. Se utilizan preferentemente para la confección del Esquema Conceptual, y como modelo subyacente de algunos SGBD aun en experimentación. Son los más potentes pero no tienen un reflejo comercial en algún producto de amplia difusión; la representación del EC ha de traducirse a

⁵ El modelo EER será estudiado en profundidad en la asignatura Bases de Datos II, mientras que el resto de los modelos semánticos son objeto de estudio en la asignatura Bases de Datos Avanzadas.

un Esquema Lógico, en general a un modelo clásico, para poder ser explotado eficientemente en algún SGBD.

Así y todo, parece que el EER está teniendo alguna penetración sobre todo en herramientas de análisis y diseño de Sistemas de Información, en forma de módulo de ciertas herramientas CASE que se puede traducir de forma automática al Modelo Relacional.

Otros modelos de datos, que denominaremos **de propósito particular**, se desarrollaron sobre aplicaciones concretas: cartografía, CAD/CAM, hipertexto, etc.

BIBLIOGRAFÍA

[DATE01]. Capítulos 1 y 2.

[ELMASRI02]. Capítulo 1.

[SILBERSCHATZ02]. Capítulo 1 y apéndices A (Modelo de Red) y B (Modelo Jerárquico).

[MOTA02]. Capítulo 2.

III EL MODELO RELACIONAL

La teoría del Modelo Relacional se desarrolló hacia el 1970 de la mano de E. Codd, que propuso también tres lenguajes de definición y manipulación de datos basados en el Álgebra de conjuntos y el Cálculo de Predicados de Primer Orden. Desde entonces, el Modelo Relacional se ha impuesto claramente sobre sus inmediatos predecesores, el Jerárquico y el Red, por su sencillez y por la aparición de un lenguaje de especificación, el SQL (*Standard Query Language*), de fuerte aceptación como lenguaje de explotación.

Veremos a continuación las estructuras que definen el modelo, los operadores asociados y los mecanismos para representar restricciones de integridad.

III1. introducción intuitiva

Se puede decir que empezamos a hablar de tecnologías de bases de datos propiamente dichas con la aparición del modelo relacional. De hecho, el modelo jerárquico y el modelo en red, los otros clasificados dentro del grupo de los modelos clásicos, no eran considerados como tales sino que la aparición del relacional forzó la formalización de esos dos sistemas de gestión de ficheros.

Sin duda es el modelo de datos de más éxito entre los SGBD comerciales. Su difusión se basa en la sencillez en la representación de los datos y en la aparición de un lenguaje de manipulación de datos que es considerado como estándar y cada vez más utilizado.

Los modelos clásicos, el relacional entre ellos, se considera que están orientados a registro (mientras que los semánticos se dice que están orientados al objeto). No obstante, de cara al usuario, el modelo relacional no presenta la información en registros sino como **tablas**. La tabla presenta la información referente a un concepto en forma de *filas*, y las *columnas* representan una cierta característica o propiedad del concepto; los nombres descriptivos de dichas propiedades están en la cabecera de la tabla.

tabla ALUMNO

exp	dni	nombre	titulación	curso	grupo
1	21	PEPE	ITIG	2	A
3	52	LUISA	ITIS	2	C
2	23	ANA	ITIG	2	A

No existe otra forma de representar la información: la única estructura que ofrece el modelo para captar cualquier realidad es la *tabla*. La información siempre se almacena en forma de valores explícitos, es decir, no existen punteros o mecanismos similares que relacionen la información sino que valores iguales en dos columnas de diferentes tablas indican una posible relación.

Además, los lenguajes de manipulación de datos no son navegacionales, no implican procesos de recuperación secuencial de registros, sino que, simplemente, el usuario le dice al sistema que condiciones ha de cumplir la información resultante. Ese resultado también se presenta en forma de tabla, o lo que es lo mismo, el lenguaje es cerrado ya que operandos y resultado son del mismo tipo *conjunto*.

El estándar actual, el SQL, sufre pocas variaciones de una marca a otra de SGBD y ha contribuido también a la alta aceptación del modelo.

III2. concepto de relación

La teoría del Modelo Relacional se basa en el concepto matemático de **relación**, formalismo en el que se apoya la tabla⁶. Definiremos los conceptos necesarios hasta llegar al de Relación Matemática: dominio y producto cartesiano. La relación nos permitirá representar objetos y restricciones semánticas, y los operadores utilizarán relaciones como operandos y darán como resultado nuevas relaciones.

dominio

Un dominio es un conjunto de valores escalares, en el sentido de que son las unidades semánticas de datos más pequeñas, son valores simples que no tienen una estructura interna.

En realidad, hablar de dominios es hablar de *tipos de datos* sobre los que toman valores los objetos del sistema de información, y es un concepto clave a la hora de poder establecer criterios de ordenación o, simplemente, comparar dos objetos.

Por otro lado, es evidente que si definimos un dominio y una cierta clase de objetos sobre él, estamos describiendo directamente algunas de las restricciones semánticas del sistema de información, en concreto las que limitan los valores que puede tomar un objeto.

producto cartesiano

Dada una colección de conjuntos D_1, D_2, \dots, D_n , no necesariamente disjuntos, el *producto cartesiano* de los n conjuntos $D_1 \times D_2 \times \dots \times D_n$ es el conjunto de todas las posibles n -tuplas $\langle d_1, d_2, \dots, d_n \rangle$ tales que la componente i -ésima de la misma pertenece al i -ésimo conjunto.

$$D_1 \times D_2 \times \dots \times D_n = \{ \langle d_1, d_2, \dots, d_n \rangle / d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n \}$$

relación matemática

R es una *Relación Matemática* definida sobre los dominios D_1, D_2, \dots, D_n si es un subconjunto del producto cartesiano $D_1 \times D_2 \times \dots \times D_n$.

$$R \subseteq D_1 \times D_2 \times \dots \times D_n$$

⁶ Desde un punto de vista formal, el termino correcto sería relación, sin embargo se utilizan indistintamente los términos tabla y relación.

grado de una relación matemática

A los conjuntos D_i se les denomina **dominios**. El número de dominios sobre el que está definida una Relación Matemática recibe el nombre de *grado* de la relación.

$$\text{Grado}(R) = n.$$

cardinalidad de una relación matemática

Asimismo, se denomina *cardinalidad* de la relación al número de n-tuplas que contiene R . Resulta evidente que mientras el grado es constante, la cardinalidad de una relación varía en el tiempo (a medida que se incluyen o eliminan tuplas de la relación).

De la definición de Relación Matemática podemos deducir las siguientes conclusiones:

1. En una Relación Matemática, por ser un conjunto, no hay tuplas duplicadas. Es evidente que el producto cartesiano de los dominios no produce la misma tupla dos veces, y la relación es un subconjunto de ese producto cartesiano.
2. Las tuplas, por la misma razón, no están ordenadas entre sí.
3. La tupla es un conjunto ordenado de valores (lista de valores) de forma que el i -ésimo valor siempre pertenece al i -ésimo dominio.

Supongamos una relación que agrupe a los ALUMNOS DE INFORMÁTICA, de los que se conoce información tal como el *número de expediente*, *nombre*, *titulación*, *curso* y *grupo*, que toman valores, respectivamente, en los siguientes dominios:

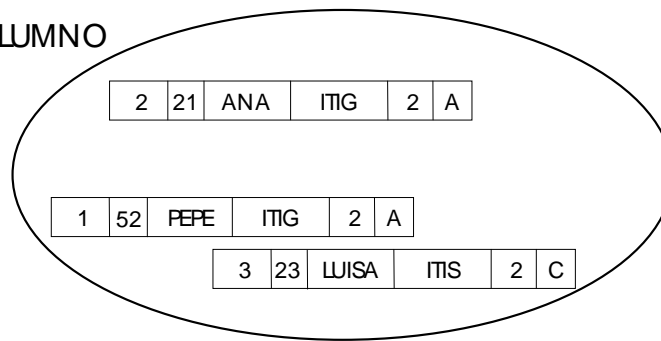
número de expediente	∈	$domExp =$	$\{ 1.. 3000 \}$
dni	∈	$domDni =$	$\{ c / c \text{ es cadena}(8) \}$
nombre	∈	$domNom =$	$\{ c / c \text{ es una cadena}(30) \}$
titulación	∈	$domTit =$	$\{ 'II', 'ITIG', 'ITIS' \}$
curso	∈	$domCur =$	$\{ 1, 2, 3, 4, 5 \}$
grupo	∈	$domGrp =$	$\{ 'A' .. 'Z' \}$

La relación matemática ALUMNO sería un conjunto de tuplas cuya primera componente pertenecería al primer dominio, $domExp$, la segunda al segundo dominio, etc.:

$$ALUMNO \subseteq domExp \times domDni \times domNom \times domTit \times domCur \times domGrp$$

El grado de la relación es 6, y un posible subconjunto de tuplas del producto cartesiano antes definido podría ser el mostrado a continuación, de cardinalidad 3.

ALUMNO



III3. representación de objetos

Una base de datos representa objetos y conceptos de la vida real. Hay procesos previos a la introducción de datos en el ordenador que implican la identificación de tales objetos y su diseño para aplicarlos a un sistema informático concreto.

Podemos decir que, tras una fase de investigación en la que se detectan las necesidades de información de una determinada organización o sistema, la información se estructura en clases de objetos que pretenden representar esa realidad particular.

En este punto veremos en primer lugar, de forma general, cuáles son los mecanismos a través de los cuales se llega a esa estructuración concreta de los datos. Posteriormente, ya dentro del modelo relacional, se detallarán las herramientas y conceptos del mismo que ayudan a esa representación.

Es importante destacar que los conceptos que se desarrollan en este punto no son exclusivos del modelo relacional sino que, por necesidad de justificación del porqué se definen las tablas como se definen, se exponen ahora.

Como recordaremos en el tema de modelos de datos, estos conceptos son generales y aplicables a cualquier modelo de datos, sean cuales sean sus herramientas de representación.

III3.1.mecanismos de abstracción

La realidad, el sistema de información a representar, tiene demasiados detalles como para poder tenerlos todos en cuenta. De hecho, muchos de ellos son irrelevantes para nuestro objetivo final, que es administrar un conjunto de datos que refleje la problemática de nuestro sistema de datos. Se hace necesario obtener una representación que únicamente abarque los detalles importantes para nuestros objetivos.

Los esquemas de base de datos hacen precisamente eso, resumir las características más importantes de nuestro sistema de información y

hacerlas comprensibles a nuestro SGBD. Esos esquemas se apoyan en un determinado *modelo de datos*.

Un **modelo de datos** es un conjunto de conceptos y unas reglas de composición de esos conceptos que, combinados de alguna forma, son capaces de representar un sistema de información, tanto en su parte estática como dinámica.

Dicho de otra manera, un lenguaje de descripción de datos, apoyado en un modelo de datos, nos permite definir las estructuras de datos que van a almacenar nuestra información, las operaciones que se realizan sobre esas estructuras y la definición de las restricciones de integridad que determinan cuales son los valores válidos en todo momento.

Pensemos, por ejemplo, en un modelo de datos cuyas herramientas de representación sean registros, campos de registro y algunos tipos de datos, y que queremos representar el hecho de la matriculación de los alumnos en ciertas asignaturas.

MODELO DE DATOS	⇒	ESQUEMA
Conceptos: registros campos tipo entero tipo carácter operador Insertar() operador Borrar() operador Consultar() Reglas: Los registros se componen de campos. Los campos son de un determinado tipo. Los operadores se aplican a registros y devuelven registros.	abstracción	REG ALUMNO (nombre: char(30) dni:char(9) dir:char(30)) REG ASIGNATURA (cod:char(5) nombre:char(30) créditosT:entero créditosP:entero) REG ASISTE (asig:char(5) alum:char(9)) asiste.Insertar(r) = si (alumno.Consultar(x) y asignatura.Consultar(y))

OCURRENCIA DEL ESQUEMA

dni	nombre	dir
21	Pepe	c/Toro, 10
22	Juan	c/Moisés, 30 3 D

cod	nombre	créditosT	créditosP
BD1	Bases de Datos 1	6	3
BD2	Bases de Datos 2	3	3

asig	alum
BD1	21
BD2	21
BD1	22

Así, la primera tarea a realizar es seleccionar el conjunto de características representativas del sistema y excluir lo irrelevante. En este proceso de **abstracción** identificaremos los objetos importantes, las relaciones que se dan entre ellos, las operaciones a realizar, y los

límites de comportamiento que afectan a todos, tanto datos como operaciones (restricciones semánticas o de integridad).

Al modelar una determinada situación estamos aplicando los siguientes mecanismos de abstracción:

- Clasificación: definir un concepto como una clase de objetos
- Agregación: definir una nueva clase de objetos a partir de otras clases
- Generalización: definir subtipos de una clase

clasificación.

Entendemos por clasificación como la definición de un concepto como una *clase de objetos*.

Si observamos los objetos *enero, febrero, marzo, ..., y diciembre*, podemos establecer una clase de objetos que represente al concepto MES, que será, por decirlo de otra manera, un conjunto de objetos (los meses en sí) agrupados bajo un mismo epígrafe.

Los nombres de persona se pueden clasificar como una clase de objetos que los defina:

NOMBRE = { Alfredo, Antonio, Antonia, Armando, ... }

o, de forma más general:

NOMBRE = { s / s es una cadena de caracteres de longitud variable
}

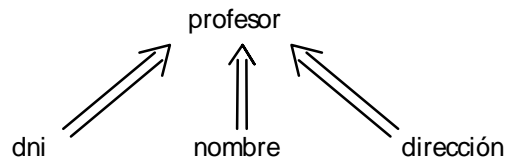
También, para el D.N.I.:

DNI = { s / s es una cadena de 8 dígitos }

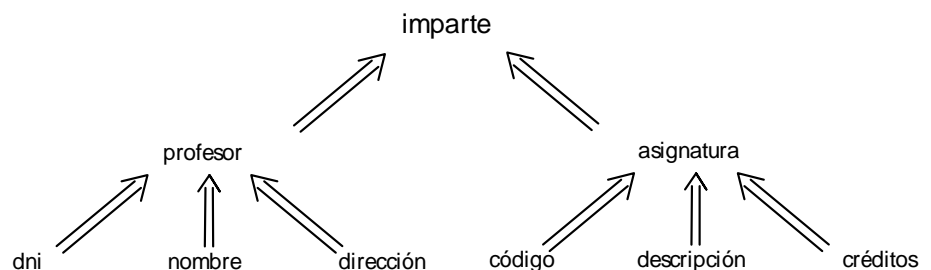
Resumiendo, la clasificación tipifica en una clase de objetos un conjunto de objetos reales o, desde otro punto de vista, dichos objetos son ***miembros de*** una clase de objetos

agregación

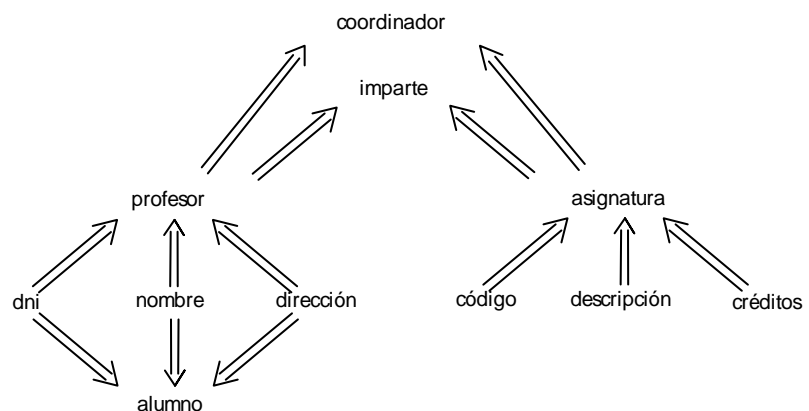
La definición de una clase de objetos a partir de otras ya definidas se conoce como **agregación**. Por ejemplo, la clase PROFESOR se define a partir de la agregación de los objetos *dni, nombre, y dirección*.



Definimos la clase de objetos como una agregación de sus partes componentes, (*dni es parte de profesor*). Un ejemplo un poco más complejo de agregación sería el siguiente, que define una clase IMPARTE, en función de las clases profesor y asignatura.



Por otro lado, una misma clase de objetos puede participar en más de una agregación.



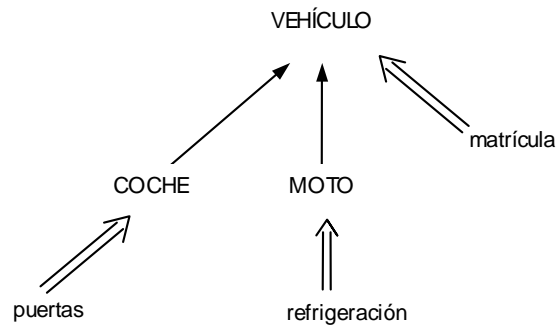
generalización

Este mecanismo nos sirve para definir subtipos de una clase de objetos. Pensemos en COCHES y MOTOS: todos ellos son VEHÍCULOS, pero aunque la agregación de matrícula es común a todos ellos, sólo podemos hablar de número de puertas en los coches, mientras que el tipo de refrigeración del motor es una propiedad relevante únicamente en las motocicletas.

Un objeto generalizado es un objeto definido a partir de dos o más objetos especializados con propiedades comunes. Las propiedades del objeto generalizado serán las comunes a todos los especializados,

mientras que estos últimos, además, aportan las suyas que no se pueden aplicar al resto de objetos dentro de la generalización.

Decimos que un objeto especializado **es_un** objeto generalizado (un coche es *un* vehículo).



III3.2.restricciones semánticas

Las **restricciones de integridad** hacen referencia a todas aquellas reglas o normas que delimitan los valores que pueden tomar las ocurrencias del esquema, y que modifican, en un sentido amplio, el *significado* de los conceptos en él recogidos. Estas restricciones pueden reflejarse simplemente mediante la utilización de los mecanismos de abstracción antes mencionados, o como añadidos no soportados por tales representaciones (*los nacidos en Alicante tienen un descuento del 10% en su declaración de renta*). Afectan tanto a las propiedades estáticas del sistema de información como a las dinámicas (las operaciones).

Veremos qué tipo de restricciones semánticas se aplican a cada uno de los conceptos que aparecen en un modelo de datos:

- *restricciones de dominio*
- *restricciones de identificación*
- *restricciones de correspondencia entre clases*

restricciones de dominio.

Al definir una clase como una asociación de objetos estamos determinando cuales son los valores válidos para la instanciación de esa clase, es decir, estamos estableciendo, por definición, el dominio sobre el que tomará valores.

restricciones de identificación.

No existen duplicados entre los posibles valores de una instanciación de la clase de objetos (aún dos hermanos clónicos, siendo exactamente iguales, ocupan espacios diferentes y son, a efectos de identificación, dos personas diferentes).

restricciones de correspondencia entre clases.

Tanto la agregación como la generalización establecen correspondencias entre las clases de objetos involucradas.

En la agregación IMPARTE, un profesor puede impartir varias asignaturas y una asignatura ser impartida por varios profesores, y podemos obligar a que todo profesor imparta al menos una asignatura. Estamos estableciendo restricciones de cardinalidad.

Entendemos como cardinalidad mínima $CardMin(o, a)$ como el número de correspondencias mínimo que se establece entre el objeto o y la agregación a . De igual forma, la cardinalidad máxima, $CardMax(o, a)$, es el máximo de correspondencias entre la clase de objetos o y la agregación a .

Por abreviar, utilizaremos la notación $Card(o, a) = (m, M)$ donde m es la cardinalidad mínima y M la máxima.

Siguiendo el mismo ejemplo las cardinalidades serían:

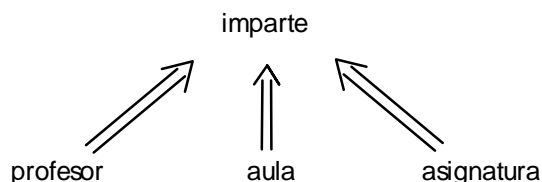
$$\begin{aligned} Card(\text{profesor}, \text{imparte}) &= (1, n) \\ Card(\text{asignatura}, \text{imparte}) &= (0, n) \end{aligned}$$

Aunque los valores típicos para la mínima son 0 y 1 y para la máxima 1 y n (n indicaría que no existe límite) se admiten otros valores si así lo requiere el sistema de información a representar. Serían del tipo que un profesor ha de impartir al menos 3 asignaturas y/o que el máximo de asignaturas que puede impartir son 5.

En las cardinalidades indicadas arriba estamos representando la obligatoriedad para todo profesor de impartir al menos una asignatura, mientras que una asignatura pudiera no tener profesor que la impartiera. Si nos fijamos en los límites máximos, ninguno de los objetos de la agregación tiene restricción alguna.

En esta agregación, la definición de la clase de objetos agregada se apoya en otras dos clases ya definidas. Sería el caso de una agregación binaria; en general, hablamos de *agregación n-aria* cuando el número de clases participantes es n .

Sea la agregación siguiente, que representa que la correspondencia entre *profesor*, *asignaturas* que imparte y *aula* donde las imparte, así como las restricciones de cardinalidad que se especifican para tal agregación.



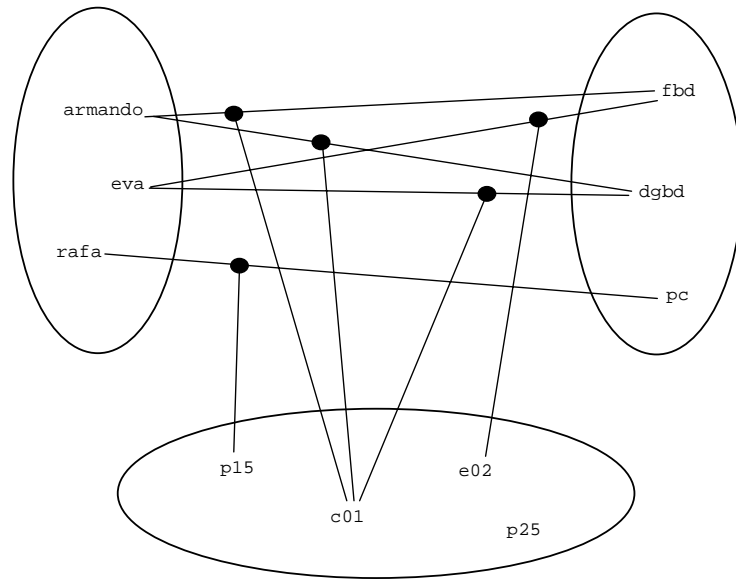
$$Card(\text{profesor}, \text{imparte}) = (1, n)$$

$$Card(\text{asignatura}, \text{imparte}) = (0, n)$$

$$Card(\text{aula}, \text{imparte}) = (0, n)$$

En este caso, la semántica asociada a la agregación indica que un profesor imparte una asignatura en un aula determinada, con la única

restricción de que todo profesor ha de estar asignado a un aula y una asignatura al menos.



Las restricciones de integridad que se refieren a las correspondencias entre clases que se producen por la generalización se conocen como **propiedades de cobertura** de la generalización.

Si dividimos a las personas entre varones y hembras todas ellas serán o varón o hembra pero no las dos cosas a la vez.

Por contra, de todas las asignaturas obligatorias de Informática, unas son compartidas por dos o más titulaciones (ITIG, ITIS, o II), y toda asignatura obligatoria pertenece a alguna titulación.

Si generalizamos a los pintores y escultores como ARTISTA (un pintor o un escultor *es un* artista), no todos los artistas son de las dos artes mencionadas, pero un pintor puede ser al mismo tiempo escultor.

Resumiendo, la generalización puede ser **total** o **parcial**, y **disjunta** o **solapada**:

total: si toda ocurrencia del objeto generalizado se corresponde con una de al menos uno de los objetos especializados.

parcial: si alguna ocurrencia del objeto generalizado no tiene su correspondiente ocurrencia en uno de los objetos especializados.

disjunta: si una ocurrencia del objeto generalizado se corresponde, como máximo, con una y sólo una ocurrencia de uno de los objetos especializados.

solapada: si una ocurrencia del objeto generalizado se corresponde con ocurrencias de objetos especializados distintos.

Las generalizaciones mencionadas tienen las siguientes propiedades de cobertura:

- PERSONA: total y disjunta.
- ASIGNATURA OBLIGATORIA DE INFORMÁTICA: total y solapada.
- ARTISTA: parcial y solapada.

III3.3.adaptación del concepto de relación matemática al modelo relacional.

La relación es la estructura básica que proporciona el modelo relacional para representar una determinada realidad. Es el medio por el que podemos hacer referencia en una BD relacional a personas, cosas o interrelaciones entre esos objetos o individuos.

No obstante, necesitamos adaptar el concepto matemático antes expuesto para la “tarea” que va a realizar como estructura del modelo relacional. Veamos cuáles son esas adaptaciones y como representar con él los objetos y relaciones entre objetos de nuestro sistema de información.

En primer lugar, recalcar el hecho de que la única referencia a una componente de una tupla, en una relación matemática, es su posición relativa dentro de ella. En otras palabras, la única información sobre la “forma” que tiene la relación son los dominios de los que se extraen los valores y el orden que ocupan en el producto cartesiano.

En el contexto del Modelo Relacional, pretendemos evitar tratar así las tuplas de la relación, como listas de valores donde cada componente se referencia por la posición que ocupa. Modificaremos, pues, la definición que se ha dado de Relación Matemática.

Una relación sobre los dominios D_1, D_2, \dots, D_n (no necesariamente disjuntos) consta de una intensión y una extensión.

- **Intensión:** un conjunto de nombres de atributos distintos A_1, A_2, \dots, A_n , cada uno de ellos asociado a su dominio correspondiente. También recibe el nombre de esquema o cabecera de la relación. Consta del nombre de la relación y de los nombres de los atributos y los dominios asociados, en forma de pares $\langle \text{nombreDeAtributo} : \text{Dominio} \rangle$. Por ejemplo:

ALUMNO = { $\langle \text{exp} : \text{domExp} \rangle, \langle \text{nombre} : \text{domNom} \rangle, \langle \text{titulación} : \text{domTit} \rangle, \langle \text{curso} : \text{domCur} \rangle, \langle \text{grupo} : \text{domGrp} \rangle$ }⁷

- **Extensión:** un conjunto de n-tuplas donde cada tupla es un conjunto de pares $(\text{nombreDeAtributo}_i : \text{Valor})$, siendo n el grado de la relación, uno por cada atributo del conjunto anterior, donde Valor siempre es un valor del dominio asociado a nombreDeAtributo. Se le conoce también como cuerpo de la relación.

$$T = \{ \begin{aligned} &\langle \text{exp}: 3 \rangle, \langle \text{nombre}: \text{LUISA} \rangle, \langle \text{titulación}: \text{ITIS} \rangle, \langle \text{curso}: 2 \rangle, \\ &\quad \langle \text{grupo}: \text{C} \rangle \} \\ &\{ \langle \text{nombre}: \text{PEPE} \rangle, \langle \text{grupo}: \text{A} \rangle, \langle \text{titulación}: \text{ITIG} \rangle, \langle \text{exp}: 1 \rangle, \\ &\quad \langle \text{curso}: 2 \rangle \} \\ &\{ \langle \text{exp}: 2 \rangle, \langle \text{titulación}: \text{ITIG} \rangle, \langle \text{curso}: 2 \rangle, \langle \text{grupo}: \text{A} \rangle, \langle \text{nombre}: \text{ANA} \rangle \} \end{aligned}$$

⁷ Aunque ésta es la notación formal, para abreviar, en adelante utilizaremos la forma ALUMNO(exp: domExp, nombre : domNom, titulación : domTit, curso : domCur, grupo : domGrp)

}
(T es la extensión de ALUMNO)

En la definición del punto anterior, la relación matemática era simplemente un conjunto de tuplas, donde cada valor se movía en el dominio que le correspondía por la posición que ocupaba en la tupla. Ahora una relación matemática consta de dos conjuntos, el de nombres de atributo y el de tuplas de valores.

Fijémonos que ahora no existe un orden entre los componentes de una tupla, puesto que siempre sabemos qué dominio, por ser el asociado al nombre de atributo, es el conjunto al que pertenece ese valor.

Disponemos, pues, de dos formas de describir una Relación Matemática (según la nueva definición del concepto): por intensión o por extensión.

- - por intensión: $R \{ (A_1 : D_1), (A_2 : D_2), \dots, (A_n : D_n) \}$.
- - por extensión: $\{ \{ (A_1 : v_{i1}), (A_2 : v_{i2}), \dots, (A_n : v_{in}) \} \mid i=1, 2, \dots, m \}$.
- ($\text{Grado}(R) = n$; $\text{Cardinalidad}(R) = m$)

Así definida la relación, los componentes de una tupla se referencian por el nombre del atributo, no por su posición relativa dentro de ella; no existe, pues, un orden dentro de cada tupla para sus componentes.

III3.4.percepción de la relación: la tabla

Una vez familiarizados con el concepto matemático necesitamos trasladar el concepto a un medio “físico” con el que podamos trabajar, ya sea en papel o en el ordenador, debemos representar la relación de alguna manera. La representación de la idea abstracta de relación sobre un SGBD relacional es la *tabla*.

La tabla se asemeja a una matriz donde la fila representa una tupla de la relación matemática, y la columna los valores de las componente de cada tupla asociados al dominio correspondiente.

exp	nombre	titulación	curso	grupo
1	PEPE	ITIG	2	A
3	LUISA	ITIS	2	C
2	ANA	ITIG	2	A

Decimos que la tabla es la representación física de la relación matemática. Sin embargo, una tabla sugiere unas características que no tiene la relación:

- En una tabla vemos que existe un orden entre sus filas. Por tanto, si podemos diferenciar cada tupla por el orden que ocupa dentro de la tabla, es perfectamente posible tener dos tuplas con los mismos valores para todos sus atributos.

- También sugiere un orden entre las columnas: podemos referirnos a una componente de una fila por su nombre o por la posición dentro de ella.

No obstante, son detalles de representación que no influyen a la hora de su tratamiento: nosotros entenderemos una tabla como una Relación Matemática, con todas las características de la definición antes expuesta.

Así pues, al trasladar el concepto de Relación Matemática al Modelo Relacional, nos encontramos con que los conceptos matemáticos tienen un equivalente informal.

Concepto formal	Equivalente informal
relación	tabla
atributo	columna
tupla	fila
grado	número de columnas
cardinalidad	número de filas

Por último, mencionar que si bien la percepción del usuario de la "forma" de sus datos es la *tabla*, nada tiene que ver con su almacenaje físico. La representación de los datos en los dispositivos físicos del ordenador no es en forma de tablas sino registros estructurados de una manera más o menos eficiente (o, en general, cualquier otro método). La tabla es, por tanto, una abstracción de su representación física la cual es desconocida por el usuario final.

III3.5.abstracciones.

Las clases de objetos se representan por medio de relaciones (tablas). Veamos cuál es la mecánica de definición de esas relaciones.

El proceso de abstracción de clases de objetos utiliza tres mecanismos concretos, tal como mencionamos anteriormente:

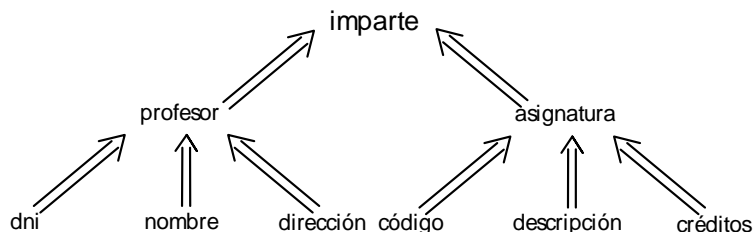
- **clasificación:** La descripción de dominios y la definición de atributos que toman valores en esos dominios.
- **agregación:** La definición de relaciones.
- **generalización:** La definición de relaciones como subtipos de otra relación.

Veremos, con ejemplos sencillos, como trata el modelo relacional estos mecanismos de abstracción. Como ya hemos dicho, la clasificación consiste en la identificación de objetos y los dominios sobre los que trabajan.

dni	∈	domDni =	{ c / c es cadena(8) }
nombre	∈	domNom =	{ c / c es cadena(30) }
dirección	∈	domDir =	{ c / c es cadena(50) }

código \in $domCod = \{ c / c \text{ es cadena}(6) \}$
 descripción \in $domNom$
 créditos \in $domCdt = \{ n / n \text{ es real y } n > 0.0 \}$

La agregación puede verse como la composición de objetos de mayor nivel a partir de los identificados en la clasificación.



El esquema de la base de datos (el conjunto de todos los esquemas de relación), según las agregaciones anteriores, sería el siguiente⁸:

PROFESOR (dni:domDni, nombre:domNombre, dirección:domDir)

ASIGNATURA (código: domCod, descripción: domNom, créditos: domCdt)

IMPARTE (profesor:domDni, asignatura:domCod)

La simple comparación de valores entre las relaciones nos permite saber qué profesores imparten qué asignaturas, como se puede ver en las extensiones de las relaciones que se muestran seguidamente.

PROFESOR

dni	nombre	dirección
211	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

ASIGNATURA

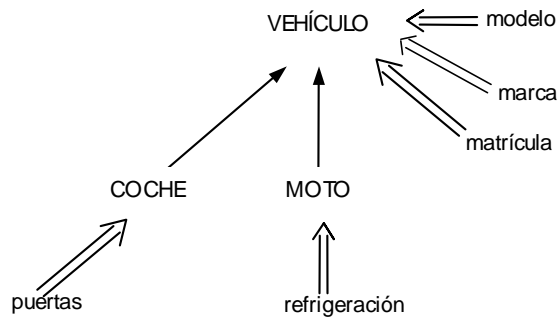
código	descripción	créditos
BD1	Bases de datos 1	9
BD2	Bases de datos 2	6
FP2	Fundamentos de programación 2	3

IMPARTE

profesor	asignatura
211	BD1
321	BD1
211	BD2

⁸ Más adelante, por las restricciones de cardinalidad de correspondencia entre clases, se justificará esta representación.

Sea la generalización de vehículos que se propone a continuación:



El esquema resultante sería el siguiente:

VEHÍCULO (matrícula : domMat, marca : domMarca, modelo : domMod)

COCHE (mat : domMat, puertas : domPp)

MOTO (mat : domMat, refrigeración : domRfr)

VEHÍCULO			
matrícula	marca	modelo	
A-0000-AA	Xeat	Kordoba	
A-1111-BB	Susuki	Fantom	
0-0000-0	Ferrari	Testarrossa	

COCHE		MOTO	
mat	puertas	mat	refrigeración
0-0000-0	3	A-1111-BB	agua
A-0000-AA	5		

Por la estructuración de las relaciones, sabemos que el vehículo con matrícula A-0000-AA es un coche de marca Xeat y modelo Kordoba y que tiene 5 puertas. El vehículo A-1111-BB es una moto Susuki Fantom refrigerada por agua.

Los atributos comunes a todos los tipos se mantienen en la relación vehículo, mientras que el número de puertas únicamente es aplicable a los coches, como el tipo de refrigeración sólo lo es para las motos.

Más adelante veremos como se completan los esquemas de relación para dotarles de una semántica adecuada y preservar la integridad de los datos.

III4. restricciones semánticas

Con lo visto hasta ahora no somos capaces de representar fielmente todas las características del sistema de información a mecanizar, ni

hemos conseguido que la tabla se comporte como una relación: ¿cómo evitamos la duplicación de tuplas en las relaciones?

Otro ejemplo: podemos introducir información en las tablas definidas como se proponía en el punto 3, pero no controlamos la integridad de los datos. Podríamos decir, insertando una nueva tupla en la relación IMPARTE, que Lucía se responsabiliza de la docencia de XYZ, pero tal código de asignatura no se encuentra en la relación ASIGNATURA: ¿cuál es esa asignatura, cuántos créditos tiene?

Vamos a ver, a continuación, qué mecanismos utiliza el modelo relacional para reflejar todas las restricciones semánticas que afectan a un sistema de información representado mediante una base de datos relacional.

Los conceptos del modelo relacional a desarrollar son los siguientes:

- dominio
- valor nulo
- clave primarias y alternativa
- clave ajena e integridad referencial

III4.1. Restricciones de dominio

Todo atributo tiene un dominio asociado (un tipo de datos), un conjunto de valores sobre los que toma valor. Los atributos pueden contener nulos o tener prohibida esta posibilidad.

Un **valor nulo** es ausencia de información, un dato que se desconoce. No tiene nada que ver con valores 0 o cadenas de longitud 0, puesto que estos son valores conocidos. Es un concepto fundamental de la teoría del modelo relacional pero que está lejos de haberse solucionado satisfactoriamente.

Por ejemplo, de una persona podemos no conocer su teléfono. Sin embargo, esto no implica que esa persona no posea ese servicio sino, simplemente, que no disponemos de esa información. Por otro lado, el valor NULO no se puede comparar con otros valores; únicamente podemos saber si un atributo es NULO o no lo es. Así, si el sueldo de un empleado es desconocido, obviamente, no podemos interrogar a la base de datos sobre si esa persona es la que más cobra de toda la plantilla.

III4.2. restricciones de identificación

Como ya hemos dicho, no existen dos tuplas iguales en una relación. Es obvio, pues, que podemos encontrar un conjunto de atributos que

por su combinación de valores nos identifique unívocamente una determinada tupla de la relación (o fila de una tabla)⁹.

Pensemos en una relación que contenga información sobre PERSONAS. Parece obvio que nunca nos encontraremos el caso de dos personas con el mismo D.N.I.: si éste es un atributo de la relación, podremos diferenciar cada tupla (cada *persona*) de las demás. Estamos buscando, pues, un atributo (o subconjunto de atributos) que nos permita especificar una única tupla.

Clave candidata

Una clave candidata es un subconjunto de atributos que cumple que cualquier combinación de sus valores es única; no existirán dos filas con valores iguales en las mismas columnas que constituyen tal subconjunto. Además, no debe contener atributos que no sean relevantes para esa identificación.

Sea R una relación con atributos A_1, A_2, \dots, A_n . Se dice que un subconjunto de esos atributos $K = \{A_i, A_j, \dots, A_k\}$ es una *clave candidata* si satisface las dos propiedades siguientes: *identificación única e irreducibilidad*.

Identificación única

No hay dos tuplas en R , T_m y T_n tales que:

$$\begin{aligned} A_{im} &= A_{in} \\ A_{jm} &= A_{jn} \\ \dots A_{km} &= A_{kn} \end{aligned}$$

Para cada tupla, la combinación de valores de los atributos que forman la clave candidata es única, no se repite para ninguna otra tupla.

K es irreducible

Ninguno de los atributos de K puede eliminarse sin que se deje de cumplir la propiedad anterior, o lo que es lo mismo, ningún subconjunto de K puede diferenciar cada tupla del resto de tuplas de la relación.

Supongamos una relación R con atributos a, b, c , y d . Si **(a,d)** es clave candidata entonces¹⁰:

(a) no lo es, ni **(d)**

(a, b, d) no lo es, ni **(a, c, d)**, ni **(a, b, c, d)**

(b) sí puede serlo, y **(c)**

(c, d) sí puede serlo, y **(a, b)**, **(a, c)**, **(b, c)**

⁹ En general, no vamos a diferenciar términos tales como tupla de relación y fila, o atributo y columna: véase la discusión del punto anterior.

¹⁰ Téngase en cuenta que el orden de las columnas no importa: (a, d) es lo mismo que (d, a) .

Sea la relación SUMINISTRA, representada por la tabla que se muestra a continuación, que asocia unos proveedores (representados por su D.N.I.) con unos artículos (código de artículo) que el primero vende a un precio determinado, dato éste último que se refleja en la columna que representa al atributo *importe*. Veamos ahora cuáles podrían ser las claves candidatas.

dni	numartículo	importe
21455855	0001	1.200.000
21455855	0002	500.000
10154123	0001	25.000
14456789	0001	135.500

- { dni } no puede ser clave candidata porque viola la primera propiedad: hay dos tuplas con el valor de dni = 21455855
- Lo mismo ocurre con { numartículo }.
- Sin embargo { dni, numartículo } sí que posee valores distintos para cada tupla ('21455855,0001' es distinto de '21455855,0002', ...) y, además, ninguno de los dos atributos, como hemos dicho, puede ser clave por separado. Ésta sí sería clave candidata.
- { dni, num, importe } no sería clave candidata porque viola la segunda propiedad: existe un subconjunto de éste, { dni, numartículo }, que ya lo es.

No obstante el ejemplo, no debe pensarse en la búsqueda de las claves candidatas a partir del conjunto de tuplas que contiene la relación en ese instante, sino por el concepto al que representan. La relación ALUMNO puede dar la casualidad que no contenga ninguna tupla con el nombre repetido, pero es obvio que en la realidad es el número de expediente (*exp*) la clave candidata, y no el atributo *nombre* que viola el principio de identificación única.

Toda relación tiene al menos una clave candidata, ya que el conjunto de todos sus atributos siempre cumple la primera propiedad (por la definición de relación, no existen tuplas duplicadas), de forma que si este conjunto no es clave es porque existe un subconjunto que sí lo es.

Clave Primaria y Claves Alternativas

Una **clave primaria** es una de las claves candidatas. Es la que vamos a utilizar para identificar cada tupla de la relación, y es decisión del analista/diseñador decidir cual de entre todas, si hay más de una, es la que va a hacer el papel de primaria en el sistema. Siempre existirá una clave primaria puesto que siempre existe al menos una clave candidata.

Una **clave alternativa** es una clave candidata no primaria.

Una consecuencia de la definición de clave candidata, la identificación única de cada tupla, es la conocida como **integridad de clave**, que dice que ningún atributo de una clave candidata puede contener valores nulos. Si el *nulo* significa valor desconocido, parece un contrasentido que una clave candidata (o parte de ella) sea nula, puesto que entonces el objeto al que queremos identificar nos es desconocido. Si permitiéramos nulos en el dni de un empleado que se

llame Juan, ¿cómo podemos diferenciarlo de otro empleado que se llame de la misma manera? ¿Son la misma persona o son distintas? ¿Cómo podemos almacenar información de una persona a la que no conocemos?

Con esto aseguramos que siempre vamos a poder identificar todas y cada una de las tuplas de una determinada relación; no tendría sentido una clave nula ya que nos faltaría precisamente la información que la diferencia de las demás tuplas¹¹.

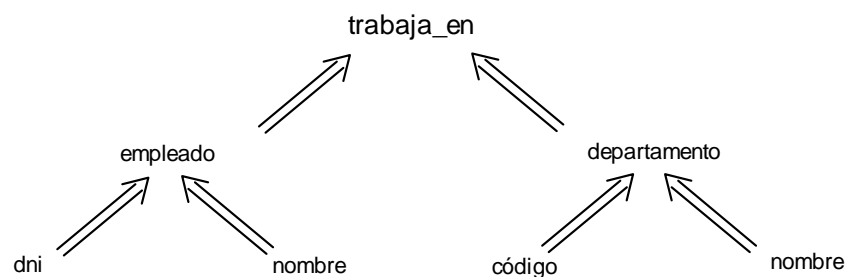
Aunque pueda parecerlo, la definición de las claves primarias y alternativas no se emplea en mejorar el rendimiento del sistema a la hora de recuperar datos de nuestra BD. Los dos conceptos anteriores se utilizan, básicamente, para mantener en las tablas la propiedad de las Relaciones Matemáticas de no duplicidad entre sus tuplas, y para expresar restricciones de correspondencia entre clases de objetos también agregadas. En otras palabras, las claves nos sirven para representar la semántica de nuestro esquema conceptual.

No tiene, por tanto, el sentido de una organización indexada (aún cuando utilice esta para implementar los mecanismos de identificación), por poner un ejemplo, en un fichero convencional, puesto que pretendemos dejar al SGBD las tareas de la administración física de los datos.

III4.3.restricciones de correspondencia

La agregación y la generalización no están totalmente soportadas por el modelo relacional ya que el modelo carece de recursos para representar todas las restricciones de correspondencias entre clases de este tipo de abstracciones. No obstante, veremos las herramientas que nos proporciona la teoría relacional con respecto a estos mecanismos de abstracción.

Las **claves ajenas** sirven para asociar tuplas de relaciones distintas (o de una relación consigo misma). Es un mecanismo utilizado para expresar agregaciones de objetos ya agregados.



¹¹ Tradicionalmente, la integridad de clave se aplicaba únicamente a la clave primaria. Aunque la bibliografía no es clara a este respecto, en nuestra opinión parece un contrasentido no permitir nulos en la clave primaria pero si en el resto de candidatas, puesto que toda clave candidata debería ser capaz de identificar todas, sin excepción, las tuplas de una relación.

Sea el árbol de agregaciones anterior, donde definimos las cardinalidades siguientes:

$\text{Card}(\text{empleado}, \text{trabaja_en}) = (0,1)$

$\text{Card}(\text{departamento}, \text{trabaja_en}) = (0, n)$

¿Cómo podemos expresar, dentro del modelo relacional, que un empleado trabaja en un determinado departamento dentro de una empresa? Una solución es anotar, como una característica más del empleado, cuál es ese departamento en el que desempeña su labor. Tal información está representada por dos relaciones: EMPLEADO y DEPARTAMENTO, donde se han definido los atributos necesarios para cada uno: número de empleado, nombre, y código de departamento al que está asignado para la primera, y código y descripción para la segunda.

EMPLEADO

número	nombre	departamento
001	Antonio	DEP1
002	José	
003	Susana	DEP2
004	Pilar	DEP1

DEPARTAMENTO

código	nombre
DEP1	Ventas
DEP2	Investigación
DEP3	Contabilidad

En el modelo relacional, una **clave ajena** es un subconjunto de atributos de una relación de tal forma que el valor de todos ellos debe coincidir con el valor de la clave primaria de una tupla de otra relación, en el caso de tener algún valor.

El atributo *departamento* de la tabla EMPLEADO actúa como clave ajena, y referencia, por su valor en cada tupla, una determinada fila de la tabla DEPARTAMENTO. Con este mecanismo de representación estamos diciendo que el empleado *Antonio*, de clave *001*, trabaja en el departamento de *Ventas*, lo mismo que *Pilar*, mientras *Susana* lo hace en el de *Investigación*.

Así, tal y como está definida la clave ajena podemos deducir (con la información de que disponemos acerca de la definición de las tablas) que un departamento tiene asignados ninguno o varios empleados, y que un empleado no necesariamente está asignado a un departamento en concreto (el empleado *002* no está asignado a ningún departamento, y el departamento *DEP3* no tiene empleados).

Integridad Referencial

Toda clave ajena perteneciente a una relación *S* que referencie a la clave primaria de otra relación *R* (*R* y *S* no han de ser necesariamente distintas) debe cumplir que todo valor no nulo en dicha clave ajena ha de ser exactamente igual al valor de clave primaria de alguna tupla de *R*.

Todo SGBD basado en el Modelo Relacional que posea mecanismos de claves ajenas debe garantizar todas las restricciones de integridad que se puedan expresar en él; en particular, la *integridad referencial*.

En este caso, debe asegurar que una clave ajena de una relación, o bien no referencia a nada (la tupla no está relacionada con otra), o bien referencia a una tupla existente en otra relación. Nunca podrá tener valores que no se encuentren en la clave primaria de alguna tupla de otra relación.

¿Cómo puede el SGBD garantizar la integridad referencial? Suponiendo que disponga de toda la información necesaria sobre claves (primarias y ajenas, al menos) se encuentra con el problema de las actualizaciones y borrados de claves primarias referenciadas por alguna clave ajena.

Sea el caso de la relación entre EMPLEADO y DEPARTAMENTO, antes descrita. El operador pretende borrar de la tabla DEPARTAMENTOS la tupla del "Dpto. de Investigación". Si así lo hiciere, en EMPLEADOS, la tupla de "Susana" tendría su clave ajena apuntando a un valor de clave primaria que no existe en DEPARTAMENTOS, lo que viola la integridad referencial.

También se puede dar el caso de que dicho Dpto. de Investigación cambie de clave primaria y pase a identificarse como "DEP5". Nuevamente, si no tomamos las medidas oportunas, estaríamos violando la integridad referencial. En definitiva, para actualizaciones y borrados de tuplas referenciadas por claves ajenas en otras tablas, el SGBD debe conocer, por las especificaciones oportunas en el esquema de la BD, cual de las siguientes estrategias debe utilizar para garantizar la restricción:

- Rechazar
- Anular
- Propagar

RECHAZAR

El SGBD sólo permitirá la operación en caso de que no produzca problemas. En nuestro ejemplo no dejaría que realizásemos la operación.

ANULAR

El SGBD pondrá a nulos todas las claves ajenas que hagan referencia a la clave primaria que sufre la operación. En nuestro ejemplo, si pretendemos borrar la tupla de "Investigación", la tupla de "Susana" quedaría con el atributo DEPARTAMENTO a valor nulo.

EMPLEADO

número	nombre	departament o
1	Antonio	DEP1
2	José	
3	Susana	
4	Pilar	DEP1

DEPARTAMENTO

código	nombre
DEP1	Ventas
DEP3	Contabilidad

Supongamos, ahora, que para actualizar el dpto. de Ventas a un nuevo código, cambiarlo de “DEP1” a “DEP4”, adoptamos la misma estrategia. Obtendríamos:

EMPLEADO

número	nombre	departamento
001	Antonio	
002	José	
003	Susana	
004	Pilar	

DEPARTAMENTO

código	nombre
DEP4	Ventas
DEP2	Investigación

PROPAGAR

También conocida como “en cascada” porque reproduce la operación sobre todas las tuplas que hagan referencia a la clave primaria. Si partimos, nuevamente, del ejemplo original y borramos la tupla del dpto. de Investigación, la tupla de “Susana” también sería borrada:

EMPLEADO

número	nombre	departamento
001	Antonio	DEP1
002	José	
004	Pilar	DEP1

DEPARTAMENTO

código	nombre
DEP1	Ventas
DEP3	Contabilidad

Y en el caso de la actualización antes mencionada de Ventas:

EMPLEADO

número	nombre	departamento
001	Antonio	DEP4
002	José	
004	Pilar	DEP4

DEPARTAMENTO

código	nombre
DEP4	Ventas
DEP3	Contabilidad

Por ejemplo, en un lenguaje de definición de datos ficticio podríamos definir la estructura de la tabla EMPLEADO como sigue:

```

CREAR TABLA EMPLEADO (
    número carácter(3), nombre carácter(20), departamento
    carácter(4),
    CLAVE PRIMARIA ( número ),
    CLAVE AJENA ( departamento ) REFERENCIA
    DEPARTAMENTO
    NULOS PERMITIDOS
    BORRADOS EN DEPARTAMENTO RECHAZAR
    ACTUALIZACIONES EN DEPARTAMENTO PROPAGAR
);
  
```

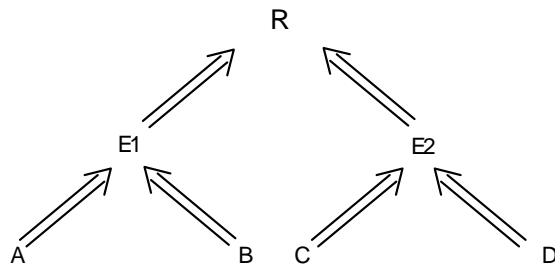
Nótese que para cada una de las operaciones se han definido métodos distintos para mantener la integridad referencial. Además, dicha definición ha de hacerse con todas y cada una de las claves ajenas aún estando en la misma relación:

```

CREAR TABLA SUMINISTRA (
    prov carácter(4), art carácter(5), precio entero largo,
    CLAVE PRIMARIA ( prov, art ),
    CLAVE AJENA ( prov ) REFERENCIA PROVEEDOR
        BORRADOS EN PROVEEDOR RECHAZAR
        ACTUALIZACIONES EN PROVEEDOR PROPAGAR );
    CLAVE AJENA ( art ) REFERENCIA ARTÍCULO
        BORRADOS EN ARTÍCULO PROPAGAR
        ACTUALIZACIONES EN ARTÍCULO PROPAGAR );
    
```

restricciones de cardinalidad

Las restricciones de cardinalidad limitan el número de tuplas de varias relaciones que se pueden asociar entre sí.



La representación de las siguientes restricciones de cardinalidad se hace mediante la definición apropiada de claves ajenas, tantas como sean necesarias, y de la definición de claves primarias. Más que saber cómo traducir un esquema conceptual (en el modelo de datos que sea) a un esquema lógico relacional, interesa comprender cómo, mediante tablas y claves, se puede “simular” una relación entre entidades.

$Card(E1, R) = (0, n)$

$Card(E2, R) = (0, 1)$

La forma de representar estas cardinalidades entre E1 y E2 es definir una tabla para cada objeto y añadir una columna a la tabla E2 que actúe como clave ajena hacia la tabla E1:

<p>E1 (A : domA, B : domB, C : domC) Clave Primaria: A</p> <p>E2 (D : domD, E : domE, aE1 : domA) Clave Primaria: D Clave Ajena: aE1 → E1</p>
--

Este es el mecanismo utilizado para relacionar EMPLEADO y DEPARTAMENTO en un ejemplo anterior.

El número de columnas que son clave ajena debe coincidir con el número de columnas de la clave primaria de la tabla con la que se relaciona y, además, coincidir en dominios para poder compararlas.

Una clave ajena, en principio, no es una clave candidata dentro de la tabla en la que se define, y por lo tanto admite, en principio, duplicados y valores nulos. Por eso podíamos tener empleados que no estuvieran asignados a ningún departamento, y varios empleados con el mismo valor de código de departamento en la columna que actúa como clave ajena.

Es importante, también, darse cuenta de que se respetan las cardinalidades mínimas. En concreto, existen departamentos que no están asociados a ningún empleado, y el hecho ya comentado de empleados sin departamento.

$Card(E1, R) = (0, n)$

$Card(E2, R) = (1, 1)$

La representación es la misma que en el caso anterior pero obligando a que la clave ajena tenga siempre valor (que sea de *Valor No Nulo*):

E1 (A : domA, B : domB, C : domC) Clave Primaria: A E2 (D : domD, E : domE, aE1 : domA) Clave Primaria: D Clave Ajena: aE1 → E1 VNN
--

Este sería el caso, aplicándolo a la agregación TRABAJA_EN, en el que todos los empleados, sin excepción, están asignados a un departamento.

$Card(E1, R) = (0, n)$

$Card(E2, R) = (0, n)$

En este caso, se define una tercera tabla con un número de columnas igual a la suma del número de columnas que son clave primaria de las entidades relacionadas.

E1 (A : domA, B : domB, C : domC) Clave Primaria: A E2 (D : domD, E : domE) Clave Primaria: D	R (aE1 : domA, aE2 : domD) Clave Primaria: (aE1, aE2) Clave Ajena: aE1 → E1 Clave Ajena: aE2 → E2
--	---

Éste es el caso de la relación SUMINISTRA, suponiendo que existieran dos tablas más, PROVEEDOR y ARTÍCULO, donde la clave primaria de la primera es el D.N.I. del vendedor y el código de artículo la de la segunda.

Es evidente que las parejas (dni, numartículo) no se repiten nunca: un vendedor no vende un mismo artículo a dos precios distintos. Sin embargo, si nos fijamos en las claves ajenas, como tales, el proveedor 21455855 aparece dos veces en su columna, y lo mismo ocurre para el artículo 0001.

Las cardinalidades mínimas se respetan, ya que puede haber vendedores de la tabla PROVEEDOR, o piezas de PIEZA, que no aparezcan referenciados en la tabla SUMINISTRA.

$$\text{Card}(E1, R) = (0, 1)$$

$$\text{Card}(E2, R) = (0, 1)$$

En este caso, también se define una tercera tabla, pero una de las claves ajenas actúa ahora como primaria y la otra como alternativa.

E1 (A : domA, B : domB, C : domC)	R (aE1 : domA, aE2 : domD)
Clave Primaria: A	Clave Primaria: aE1
E2 (D : domD, E : domE)	Clave Alternativa: aE2
Clave Primaria: D	Clave Ajena: aE1 → E1
	Clave Ajena: aE2 → E2

Pensemos, por ejemplo, en una agregación SER_JEFE_DE entre EMPLEADO y DEPARTAMENTO en la que uno de los empleados es el director de un único departamento, y todo departamento, a su vez, tiene un único jefe.

Veamos ahora los casos con cardinalidad mínima en uno o los dos objetos:

$$\text{Card}(E1, R) = (0, 1)$$

$$\text{Card}(E2, R) = (1, 1)$$

E1 (A : domA, B : domB, C : domC)
Clave Primaria: A
E2 (D : domD, E : domE, aE1 : domA)
Clave Primaria: D
Clave Alternativa: aE1
Clave Ajena: aE1 → E1

$$\text{Card}(E1, R) = (1, 1)$$

$$\text{Card}(E2, R) = (1, 1)$$

R (A : domA, B : domB, C : domC, D : domD, E : domE)
Clave Primaria: A
Clave Alternativa: D

Sin embargo, el modelo relacional no es capaz de representar todas las restricciones de cardinalidad, como puedan ser, entre otras, las siguientes¹²:

$$\text{Card}(E1, R) = (0, n)$$

$$\text{Card}(E2, R) = (1, n)$$

$$\text{Card}(E1, R) = (1, n)$$

$$\text{Card}(E2, R) = (0, 1)$$

$$\text{Card}(E1, R) = (2, n)$$

$$\text{Card}(E2, R) = (0, 3)$$

¹² No es exactamente cierto, lo que ocurre es que ciertas representaciones que pretendan resolver estas restricciones de cardinalidad introducen redundancia o facilitan la inconsistencia de la base de datos. La correcta interpretación de esta afirmación se da en el tema de Teoría de la Normalización.

Un caso particular de restricción de cardinalidad es el siguiente:

E1 (A : domA, B : domB, C : domC) Clave Primaria: A
E2 (aE1 : domA, D : domD, E : domE) Clave Primaria: (aE1, D) Clave Ajena: aE1 → E1

Es el caso, por ejemplo, de calles que pertenecen a distintas localidades. La *Avda. de la Constitución*, con tal denominación, se encuentra en varias poblaciones, pero no es la misma avenida la de *Elda* que la de *Monóvar*: aún llamándose igual, son dos objetos diferentes físicamente. El problema está en que el nombre de la calle no es clave candidata, y debe apoyarse en el nombre de la población para distinguirse de las otras calles de nuestra base de datos. Ahora éstas se denominan, respectivamente, *<Avda. de la Constitución, Elda>* y *<Avenida de la Constitución, Monóvar>*¹³. Teniendo esto en cuenta, pese a lo que pudiera pensarse por la declaración de la clave primaria de E2, las cardinalidades de ambos objetos en la agregación que los asocia son:

Card(E1, R) = (0, n)

Card(E2, R) = (1, 1)

Según el ejemplo, E1 se correspondería con la población y E2 con la calle. Como se puede ver, es fácil distinguir este tipo de agregación por la definición de una clave primaria que, en parte y no totalmente, es al mismo tiempo clave ajena hacia otra relación. Cuando una clave primaria es enteramente una clave ajena estamos ante una generalización, como se desarrolla a continuación.

generalización y propiedades de cobertura

La generalización se representa, en el modelo relacional, mediante la utilización de una clave ajena que es, al mismo tiempo, clave primaria de la relación.

VEHÍCULO (matrícula : domMat, marca : domMarca, modelo : domMod) Clave primaria: matrícula	
COCHE (mat : domMat, puertas : domPp) Clave Primaria: mat Clave Ajena: mat → VEHÍCULO	MOTO (mat : domMat, refrigeración : domRfr) Clave Primaria: mat Clave Ajena: mat → VEHÍCULO

En el modelo relacional no podemos representar las propiedades de cobertura de la generalización. En realidad sería más propio decir que únicamente podemos reflejar generalizaciones *parciales* y *solapadas*,

¹³ Claramente, se trata de un ejemplo; no hemos comprobado que tal avenida se encuentre en dichas poblaciones.

puesto que no existe ningún método para las otras combinaciones de características.

III5. operadores

La tabla, o relación en su equivalente matemático, es la estructura básica del Modelo Relacional. Para poder crear y manipular tablas necesitamos un lenguaje de definición de datos (LDD) y un lenguaje de manipulación de datos (LMD).

Con el LDD, aparte de describir las tablas, también se pueden definir vistas, esquemas externos, autorizaciones, restricciones de integridad, etc.

Por otro lado, sobre la información almacenada en las tablas queremos resolver un determinado *requerimiento*, esto es, formular una pregunta a nuestra BD de la que obtendremos una respuesta (como pueda ser “¿cuáles son los vendedores de Alicante?”, o “¿cuál es el precio de suministro para la pieza X ofertado por cada vendedor?”).

A nivel formal, sobre el Modelo Relacional existen tres propuestas de LMD, en forma de lenguajes de especificación, basados cada uno en una determinada notación. Todas fueron desarrolladas por Codd:

- Álgebra Relacional.
- Cálculo Relacional de Tuplas.
- Cálculo Relacional de Dominios.

El *álgebra* es un lenguaje con operadores explícitos y, mediante sus expresiones, se representa la forma de construir nuevas relaciones. El *cálculo* emplea fórmulas lógicas que se evalúan a cierto o falso, y son la definición de la relación en términos de otras, qué condiciones cumple o qué características debe tener la información dentro de la relación resultado.

Se puede decir que el álgebra expresa la forma de resolver el problema, mientras que el cálculo define cuál es el problema a resolver. Por ejemplo:

$$\text{ALUMNO}[\text{exp}, \text{nombre}] \otimes \text{SEMATRICULA}[\text{cod}, \text{exp}]$$

Este ejemplo del álgebra utiliza los operadores proyección ($[]$), y concatenación natural (\otimes) sobre las relaciones ALUMNO y SEMATRICULA, y obtendría la información de los alumnos y las asignaturas en las que se ha matriculado por la evaluación de izquierda a derecha de los operadores, teniendo en cuenta las reglas de precedencia de los mismos. Son expresiones al estilo de las utilizadas comúnmente en los cálculos matemáticos.

$$\{ \text{xexp}, \text{xnombre}, \text{xcod} \mid \text{ALUMNO}(\text{exp}: \text{xexp}, \text{nombre}: \text{xnombre}) \wedge \text{SEMATRICULA}(\text{cod}: \text{xcod}, \text{exp}: \text{xexp}) \}$$

La anterior es una expresión con el mismo resultado que la de álgebra. Es una fórmula en cálculo relacional de dominios que devuelve todos aquellos valores de expediente, nombre de alumno y código de asignatura que hacen cierta la fórmula. Las fórmulas se basan en el cálculo de predicados de primer orden.

Como hemos dicho, son lenguajes de especificación, frente a los denominados navegacionales, porque *especifican* la información a recuperar en vez de obtener registros por su posición dentro de la estructura de datos. El Álgebra expresa los requerimientos mediante la aplicación directa de ciertos operadores definidos a tal fin sobre las relaciones. Los demás utilizan una notación lógica basada en el Cálculo de Predicados de Primer Orden, donde las consultas son fórmulas lógicas que se evalúan a cierto o falso; las tuplas contenidas en la relación (tabla) resultado de una consulta, sustituidas en la fórmula una a una, son las que hacen que se evalúe ésta a cierto.

Estos lenguajes son *cerrados*, en el sentido de que el resultado de aplicar un operador a una o más tablas ya existentes en nuestra BD será una nueva tabla que podremos consultar o utilizar para una nueva operación.

Los tres lenguajes tienen idéntica capacidad expresiva; un requerimiento escrito en uno de ellos siempre se puede traducir a cualquiera de los otros dos lenguajes, y son objeto de tratamiento en profundidad en temas posteriores.

La completitud relacional es la medida de lo "relacionalmente adecuado" que es un lenguaje. Se dice que un LMD es **completo relacionalmente** si tiene, al menos, la misma capacidad expresiva que el álgebra relacional, si puede definir cualquier relación resultado del álgebra (si ese lenguaje puede "hacer" las mismas cosas, independientemente de que ofrezca otras capacidades). SQL cumple esta propiedad.

Por contra, los tres lenguajes mencionados no son *computacionalmente completos* (no pueden calcular todas las funciones computables) puesto que no poseen, entre otras, funciones de agregación que calculen sumas, medias, etc.

III.6. otras características

vistas

En un SGBD relacional se diferencia según la temporalidad de las tablas. Denominamos **tablas base** a aquellas que forman el esquema de nuestra BD. Son las que definimos en esquema lógico¹⁴ como

¹⁴ Veremos en otro tema que una de las partes de la arquitectura de un Sistema de Gestión de Base de Datos es el esquema lógico, o lo que es lo mismo, la definición completa de las tablas que constituyen las distintas Bases de Datos a gestionar.

representación de los objetos de nuestro sistema de información. Se dice que son relaciones con nombre.

Las tablas derivadas son relaciones con o sin nombre, definidas a partir de tablas base u otras relaciones derivadas: tablas temporales, cursores, etc. Las **vistas** son tablas derivadas con nombre que sirven para mantener una consulta que se realiza frecuentemente. Supongamos, por ejemplo, que una información de frecuente uso es la lista de vendedores de la provincia de Alicante; la definición de esta vista, en SQL, sería:

```
CREATE VIEW provali AS
SELECT numvend, nomvend, nombrecomer
FROM vendedor WHERE provincia = "ALICANTE"
```

Téngase en cuenta que la vista no es una copia de una parte de la base de datos sino una "ventana" o un filtro que se aplica a la misma cada vez que se accede a la vista. En este sentido es una relación más, y se puede operar con ella (siempre que se tengan los permisos adecuados) con inserciones, borrados, etc. Se puede ver como una tabla virtual que trabaja sobre las tablas base.

Las ventajas de la utilización de vistas, entre otras, se enumeran a continuación:

- Proporcionan independencia lógica, ya que modificaciones del esquema lógico tales como añadir atributos, añadir nuevas tablas o reestructurar el orden de las columnas no les afectan.
- Permite que distintos usuarios vean la misma información de formas diferentes
- Son filtros que limitan la cantidad de información a manejar a sólo aquella que se necesita realmente.
- Los datos que no se incluyen en la vista no se pueden alterar y, por tanto, las vistas son un medio de preservar la seguridad de los datos.

optimización de consultas

Ya hemos dicho que los LMD relacionales son de especificación y no procedurales. Esto implica que únicamente le decimos al sistema *qué información necesitamos y no cómo conseguirla*. Bajo un lenguaje de programación clásico, podríamos afinar nuestros algoritmos hasta obtener el más eficiente a la hora de recuperar los datos. En un SGBD relacional la tarea de buscar el método más eficiente de consulta corre a cargo del optimizador.

En los otros modelos clásicos (jerárquico y red) el usuario era el encargado de definir la navegación a través de los registros definidos como estructura de datos (en realidad el programa de aplicación escrito en COBOL u otro lenguaje similar). En el modelo relacional esa navegación es automática y transparente para el usuario. El **optimizador de consultas** es el encargado de planificar la navegación automática a través de los datos almacenados físicamente. No olvidemos que, aunque el usuario "ve" tablas únicamente, los datos, a nivel interno, tienen una forma totalmente distinta y en general

desconocida para el usuario de la BD. El SGBD debe explorar el dispositivo de almacenamiento hasta encontrar la información requerida y, además, de la forma más eficiente posible.

En este sentido, el optimizador ha de decidir que estrategia sigue para resolver la consulta, que subconsultas realizar primero, que herramientas del sistema (como puedan ser los índices) utilizar, etc. Para tal decisión ha de tener en cuenta los siguientes condicionantes:

- qué tablas están involucradas en la consulta
- qué tamaño tienen
- qué índices hay definidos para ellas (en principio todas tendrán clave primaria que se implementa mediante un índice)
- cómo está dispuesta la información físicamente
- qué operadores se utilizan en la consulta
- información estadística almacenada por el propio sistema

diccionario de datos

Un SGBD implementa el diccionario de datos mediante lo que se conoce como **catálogo** de la BD. Es la metainformación (información sobre los datos almacenados) donde se describen los distintos esquemas de la BD (lógico, externos, interno) y las correspondencias entre ellos.

Almacena descriptores de los objetos de interés para el sistema: tablas, columnas y su tipo de datos, índices, restricciones de clave, claves ajenas, estrategias para esas claves ajenas, permisos, vistas, ... Absolutamente todo lo referente a la organización de la BD está almacenado en el catálogo. El optimizador, por ejemplo, sería un de los "usuarios" habituales del diccionario de datos.

Una de las características más importantes del catálogo es que está almacenado en forma de tablas, de tal forma que su acceso es idéntico al de cualquier tabla base de nuestro esquema lógico.

III7. conclusiones

El modelo relacional es el modelo de datos de una gran parte de los SGBD actuales. Su estructura básica es la relación, concepto matemático basado en la teoría de conjuntos que, adaptado convenientemente al medio físico en el que se va a utilizar, el ordenador, recibe el nombre de tabla.

Con la tabla vamos a ser capaces de representar los conceptos, objetos, etc. que se relacionan entre sí para conformar un determinado sistema de información. Esta tabla se define en función de sus columnas (atributos de la clase de objetos representada), donde cada

una de ellas pertenece o toma valores de un dominio (tipo de datos) concreto.

La forma de construir o definir una tabla pasa por tres mecanismos de abstracción generales a todos los modelos de datos: la clasificación, la agregación y la generalización.

Como no basta con reflejar los objetos en sí, sino que necesitamos ver como interactúan unos con otros, debemos especificar las restricciones semánticas que les afectan, tanto a ellos mismos como clases de objetos como a su relación con otras clases de objetos.

Entre las restricciones semánticas soportadas por el modelo están las de dominio, de identificación y las de correspondencia entre clases de objetos, que incluyen las de cardinalidad, dependencia de identificador y propiedades de cobertura de la generalización.

Las herramientas que proporciona el modelo para expresar estas restricciones semánticas son las claves candidatas (primaria y alternativas), claves ajenas, y atributos de valor no nulo.

Específicamente sobre claves ajenas, el SGBD debe conocer las estrategias que deben aplicarse en borrados y actualizaciones para mantener la integridad referencial de la BD.

Los operadores que permiten crear y manejar una base de datos se definen en los lenguajes de definición y manipulación de datos adecuados (que se verán más adelante) cuyo reflejo comercial es el estándar SQL.

Por último, como características propias de los SGBD relacionales se describen brevemente las vistas y el optimizador de consultas. Las primeras sirven para definir “tablas virtuales” que ayudan a la seguridad del sistema y el segundo es uno de los principales factores del éxito del modelo por su transparencia para el usuario.

BIBLIOGRAFÍA

[CELMA2002]
[DATE2001]

IV ÁLGEBRA RELACIONAL

Codd propuso tres lenguajes de especificación para el modelo relacional como base teórica de cualquier lenguaje que quisiera cumplir con los requisitos formales del modelo. Estos lenguajes no pueden ser explotados comercialmente, al menos tal y como los definió Codd, porque adolecen de falta de operadores: carecen de operadores aritméticos simples (sumas, restas, etc.), o de manipulación de cadenas de caracteres, por poner dos ejemplos "escandalosos". El propósito de estos lenguajes no es el de definir y manejar bases de datos relaciones, tan sólo constituyen una declaración de los mínimos requeridos para cualquier lenguaje de manipulación de datos que se quiera etiquetar a sí mismo como "relacional". En otras palabras, cualquier lenguaje de manipulación y definición de datos en bases de datos relacionales ha de poseer la potencia suficiente como para "hacer", como mínimo, lo que pueden "hacer" los lenguajes de Codd.

El primero de ellos, al menos por el orden en el que se van a introducir en esta asignatura, es el *álgebra relacional*. Recibe este nombre precisamente por su carácter algebraico: incluye un conjunto de operadores (ocho, concretamente) cuyos operandos son relaciones y el resultado de la operación es otra relación, del mismo modo que cuando sumamos dos enteros obtenemos otro número entero.

En primer lugar se definirán una serie de conceptos necesarios para definir los operadores del álgebra relacional y para, finalmente, trabajar con ellos. A continuación se describirán, uno por uno, los ocho operadores propuestos por Codd:

Unión	\cup	Selección	donde
Intersección	\cap	Proyección	[]
Diferencia	$-$	Concatenación Natural	∞
Producto Cartesiano	\times	División	\div

IV1. Conceptos previos.

Al describir las propiedades de cada operador se van a utilizar una serie de términos que debemos definir previamente.

En primer lugar se presentará una adaptación del concepto de relación matemática en la que se vuelve a hacer uso de la ordenación de las componentes de una tupla. El resto, son expresiones o reformulaciones de conceptos ya presentes en la definición del modelo.

Los conceptos a definir son:

- Relación
- Esquema de relación
- Nombres Cualificados de Atributo
- Alias de una relación
- Relación Nominada
- Relación Derivada
- Relaciones Compatibles
- Operación conmutativa
- Operación asociativa

relación

El AR hace uso del orden de las componentes de las tuplas para definir operadores y propiedades de los operadores. En realidad, se trata de retomar la definición original de la relación matemática como el subconjunto de un producto cartesiano de n dominios, de tal forma que las tuplas resultado de ese producto cumplan y cumplen que

Las tuplas son listas de valores (conjunto ordenado) tal que el i -ésimo valor pertenece al i -ésimo dominio.

Vamos a combinar la definición anterior de tupla con la adaptación que en su momento introdujimos a la relación matemática para adecuarla al objetivo final que es una base de datos. Utilizaremos al mismo tiempo los nombres de atributos y el orden de las componentes en una tupla:

El conjunto de nombres de atributos es un conjunto ordenado.

Las tuplas son listas de valores (conjunto ordenado) tal que el i -ésimo valor pertenece al i -ésimo dominio asociado al i -ésimo nombre de atributo.

A partir de ahora, los operadores pueden utilizar tanto el nombre simbólico de un atributo como su orden dentro de la tupla.

esquema de relación

Es la descripción formal de la relación con sus atributos y dominios asociados. En realidad se aplica únicamente a las relaciones nominadas, aquellas descritas en el esquema lógico relacional.

$$R(A_1:D_1, A_2:D_2, \dots, A_n:D_n)$$

donde: R es el nombre de la relación

A_i es el nombre del atributo

D_i es el nombre del dominio asociado a A_i

nombres cualificados de atributo

Es, por decirlo así, el nombre completo de un atributo, por ejemplo $R.A_i$, el atributo A_i de la relación R . Su uso evita la ambigüedad de dos atributos en dos tablas distintas con el mismo nombre.

En general, nos referimos a los atributos por su nombre sin especificar la relación a la que pertenecen. No obstante, es habitual que en distintas relaciones, y sobre todo en las relaciones derivadas (los resultados de operar con relaciones nominadas), nos podamos encontrar nombres de atributo coincidentes en relaciones distintas. La forma de diferenciar unos de otros es utilizar los nombres cualificados: “*alumno.nombre*”, “*asignatura.nombre*”.

En definitiva, se pueden utilizar indistintamente, siempre y cuando no se produzcan ambigüedades, las dos formas ya conocidas de referirse a un atributo:

- nombre cualificado: $R.A_i$
- nombre no cualificado: A_i

alias de una relación

Nombre alternativo para una relación. Dada una relación R se define un alias mediante la declaración:

definir alias S para R

Entonces la relación puede referenciarse tanto por R como por S , y los nombres cualificados de atributos $R.A_i$ o $S.A_i$.

relación nominada

Es toda relación definida en el esquema lógico relacional. En otras palabras, las que constituyen nuestra base de datos.

relación derivada

Es aquella que se obtiene como resultado de una expresión del Álgebra Relacional.

Una relación derivada no tiene nombre ni alias. Así pues, los nombres de los atributos de ésta se obtendrán a partir de los nombres cualificados de atributos de las relaciones operando, y si existe ambigüedad se utilizarán los alias.

Las reglas que rigen en los operadores para la asignación de nombres a los atributos de relaciones derivadas se verán con cada uno de ellos.

relaciones compatibles

Dos relaciones son compatibles si el grado de ambas es el mismo y los dominios asociados a los i-ésimos atributos de cada una son iguales.

$$R(A_1:D_1, A_2:D_2, \dots, A_n:D_n)$$

$$S(B_1:E_1, B_2:E_2, \dots, B_m:E_m)$$

R y S son compatibles si y sólo si:

- 1) $n = m$
- 2) $\forall i \ D_i = E_i \ (1 \leq i \leq n)$

Dicho de otra forma, el número de atributos ha de ser el mismo en ambas relaciones y, además, los dominios han de ser los mismos para atributos de la misma posición.

operación conmutativa

Una operación es conmutativa¹⁵ si se cumple que

$$A \otimes B = B \otimes A$$

operación asociativa

Una operación es asociativa si se cumple que

$$(A \otimes B) \otimes C = A \otimes (B \otimes C)$$

operadores

Los operadores del Álgebra Relacional (bajo el punto de vista de Codd) se dividen en dos grupos:

<u>de la teoría de conjuntos</u>	<u>relacionales</u>
UNIÓN	SELECCIÓN
INTERSECCIÓN	PROYECCIÓN
DIFERENCIA	DIVISIÓN
PRODUCTO CARTESIANO	CONCATENACIÓN NATURAL

¹⁵En la descripción de los operadores, algunos se dice que cumplen la propiedad conmutativa, refiriéndonos exclusivamente al conjunto de tuplas de la relación derivada, y no al conjunto de atributos de dicha relación resultado, cuyos nombres cualificados dependen del orden de las relaciones operando.

Como el resultado de una expresión en Álgebra Relacional es otra relación, ésta puede participar como operando a su vez, permitiendo la construcción de expresiones anidadas.

$R \otimes S = D_1$ $D_1 \otimes O = D_2$	$R \otimes S \otimes O = D_2$
--	-------------------------------

La única regla de precedencia entre operadores es la utilización de paréntesis, en cualquier caso las expresiones se evalúan de izquierda a derecha de tal forma que el resultado de una operación es el operando de la siguiente operación a su derecha:

$R \otimes S = D_1$ $O \otimes D_1 = D_2$	$O \otimes (R \otimes S) = D_2$
--	---------------------------------

Otra clasificación se basa en la propia definición de los operadores: Son **operadores básicos** (o primitivos) la *unión*, *diferencia*, *producto cartesiano*, *selección* y *proyección*, y **operadores derivados** la *intersección*, la *división* y la *concatenación natural* ya se definen a partir de la combinación de varios operadores básicos.

Dicho de otra forma, los cinco operadores básicos definen la potencia expresiva del lenguaje, con ellos se pueden realizar todas las operaciones que permite el álgebra relacional, mientras que los tres derivados se pueden ver como “atajos” o “resúmenes” de varias operaciones básicas reunidas en un único operador.

IV2. definición informal de los operadores

En primer lugar, se pretende dar una visión de cómo operan y que resultado obtienen los operadores antes mencionados. Posteriormente, se dará la definición formal de todos ellos como referencia del lenguaje.

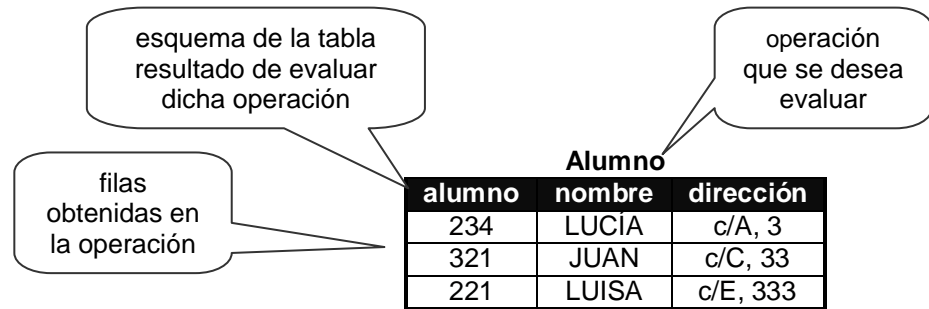
Cuando trabajamos con una base de datos relacional, si de manipulación de datos estamos hablando, la recuperación de información desde las tablas es un proceso habitual y necesario.

En álgebra relacional, si queremos recuperar el contenido completo de una relación, por ejemplo todos los alumnos almacenados en mi base de datos, una vez identificada la tabla de la que extraer los datos, la tabla alumno, la evaluación del nombre de la relación nos devuelve todas sus filas:

Alumno		
alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

```
select *
from alumno
```

Si a un intérprete de AR teórico le ordenamos simplemente “alumno”, el resultado es la tabla de 3 filas anterior. Por razones de claridad los ejemplos incluyen la operación, el esquema de la relación resultado y las filas obtenidas, según se detalla en el gráfico siguiente.



A partir de ahora, los ejemplos muestran el contenido de las tablas utilizadas y el resultado de cada uno de los operadores. También se darán las expresiones correspondientes en SQL como ayuda para la comprensión de cada operación.

selección

En muchas ocasiones sólo nos interesan algunos individuos de una relación, aquellos que poseen unas determinadas características: “clientes de la provincia de Alicante”, “artículos que cuestan mas de 500 euros”.

El operador selección (“**donde condición**”) recupera tuplas de una relación que cumplen una determinada condición.

Alumno		
alumno	nombre	dirección
577	YÉNIFER	c/C, 33
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

Alumno donde dirección = 'c/C, 33'

alumno	nombre	dirección
577	YÉNIFER	c/C, 33
321	JUAN	c/C, 33

```
select *
from alumno
where dirección = 'c/C, 33'
```

proyección

Si la selección recupera filas, la proyección ([*columna(s)*]) recupera atributos.

Alumno		
alumno	nombre	dirección
577	YÉNIFER	c/C, 33

234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

Alumno [alumno, nombre]

alumno	nombre
577	YÉNIFER
234	LUCÍA
321	JUAN
221	LUISA

```
select alumno, nombre
from alumno
```

La selección me permite recuperar aquellos individuos que me interesan y la proyección elimina aquellos datos de los individuos que no necesito. Como ya se ha dicho, se pueden realizar varias operaciones consecutivas, como se muestra a continuación:

Alumno donde dirección = 'c/C, 33' [alumno, nombre]

alumno	nombre
577	YÉNIFER
321	JUAN

```
select alumno, nombre
from alumno
where dirección = 'c/C, 33'
```

La proyección podría eliminar la o las columnas definidas como clave candidata y podría dar lugar a duplicados entre las filas (o más bien nuestra percepción de la relación nos puede inducir a pensar que se puedan mostrar filas iguales). Como el resultado de esta operación es también una tabla, ésta está sujeta a las mismas restricciones del modelo que cualquier otra. Se puede pensar que el AR realiza la proyección en dos pasos: la proyección como tal y una eliminación de duplicados, si es que se producen.

Si proyectáramos sobre la columna dirección, la tabla alumno contiene dos filas con los mismos valores. Sin embargo, el resultado de tal operación es el siguiente:

Alumno [dirección]

dirección
c/C, 33
c/A, 3
c/E, 333

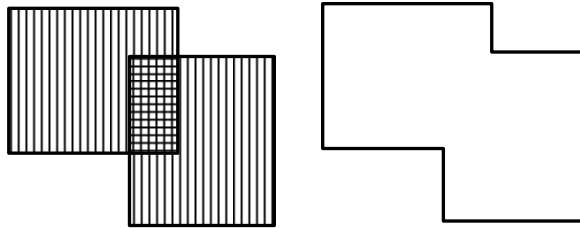
```
select dirección16
from alumno
```

¹⁶ En realidad, es habitual que los SGBD relacionales y su implementación de SQL, ante expresiones como esta devuelvan filas duplicadas. El modificador *distinct* realiza ese segundo paso al que hemos hecho referencia en el álgebra relacional. A partir de ahora, aunque no aparezca en los ejemplos, se entenderá que todas las órdenes *select* incluyen este modificador: "select **distinct** dirección from alumno where dirección = 'c/C, 33' "

Obviamente, esto es aplicable a cualquier operación que hagamos puesto que el resultado siempre va a ser una tabla derivada.

unión

La unión de dos relaciones da como resultado otra relación que contiene las tuplas de las dos. Como en una relación no existen tuplas duplicadas (por ser un conjunto, por estar definida por un producto cartesiano), sería más exacto describir la unión de dos relaciones A y B como otra relación C que contiene las tuplas de A que no están en B, las tuplas de B que no están en A, y las tuplas que están en A y B a la vez.



Alumno		
alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

Profesor		
profesor	nombre	dirección
522	JOSÉ	c/F, 32
778	EVA	c/F, 51
221	LUISA	c/E, 333

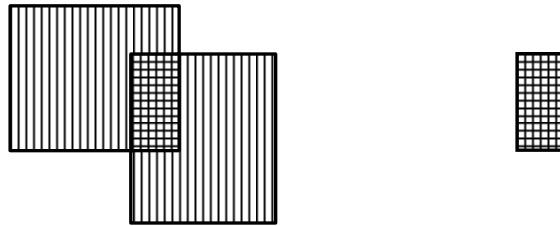
Alumno \cup Profesor		
alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333
522	JOSÉ	c/F, 32
778	EVA	c/F, 51

```
select * from alumno
union
select * from profesor
```

La unión sólo se puede realizar si A y B son compatibles y, evidentemente, es conmutativa y asociativa (ver las definiciones anteriores)

intersección

La intersección de dos relaciones da como resultado otra relación que contiene las tuplas comunes a las dos primeras.

**Alumno**

alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

Profesor

profesor	nombre	dirección
522	JOSÉ	c/F, 32
778	EVA	c/F, 51
221	LUISA	c/E, 333

Alumno \cap Profesor

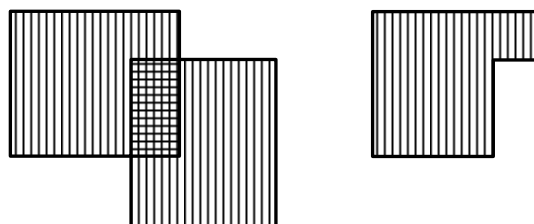
alumno	nombre	dirección
221	LUISA	c/E, 333

```
select * from alumno
intersect
select * from profesor
```

Al igual que la unión, la intersección sólo se puede realizar si A y B son compatibles y es conmutativa y asociativa.

diferencia

La diferencia de dos relaciones se define como aquellas tuplas que están en la primera pero no en la segunda. Nótese que este operador ya no es conmutativo, importa el orden de los operandos.



Alumno

alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

Profesor

profesor	nombre	dirección
522	JOSÉ	c/F, 32
778	EVA	c/F, 51
221	LUISA	c/E, 333

Alumno - Profesor

alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33

```
select * from alumno
minus
select * from profesor
```

La diferencia sólo se puede realizar si A y B son compatibles.

producto cartesiano

Este operador se utiliza para generar todas las posibles combinaciones de las tuplas de una relación con todas y cada una de las tuplas de otra relación; se obtienen todas las combinaciones posibles de filas entre dos tablas.

Alumno

alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

Profesor

profesor	nombre	dirección
522	JOSÉ	c/F, 32
778	EVA	c/F, 51
221	LUISA	c/E, 333

Alumno × Profesor

alumno	nombre	dirección	profesor	nombre	dirección
234	LUCÍA	c/A, 3	522	JOSÉ	c/F, 32
234	LUCÍA	c/A, 3	778	EVA	c/F, 51
234	LUCÍA	c/A, 3	221	LUISA	c/E, 333
321	JUAN	c/C, 33	522	JOSÉ	c/F, 32
321	JUAN	c/C, 33	778	EVA	c/F, 51
321	JUAN	c/C, 33	221	LUISA	c/E, 333
221	LUISA	c/E, 333	522	JOSÉ	c/F, 32
221	LUISA	c/E, 333	778	EVA	c/F, 51
221	LUISA	c/E, 333	221	LUISA	c/E, 333

```
select * from alumno, profesor
```

concatenación natural

Este operador se utiliza para combinar información de dos tablas que comparten un atributo común. En este caso quedará más claro un ejemplo en el que la base de datos refleje una relación entre, por ejemplo, profesores y departamentos a los que están adscritos. La relación entre los dos se representa por una columna en la tabla profesor que indica el código de departamento en el que trabaja.

La concatenación natural utiliza los atributos que tienen el mismo nombre y, por supuesto, están definidos sobre el mismo dominio.

Profesor				Departamento	
profesor	nombre	dirección	cod	cod	departamento
522	JOSÉ	c/F, 32	LSI	LSI	LENGUAJES
778	EVA	c/F, 51	LSI	CCIA	CIENCIAS
221	LUISA	c/E, 333	FI	FI	FILOLOGÍA

Profesor ⋈ Departamento

profesor	nombre	dirección	cod	departamento
522	JOSÉ	c/F, 32	LSI	LENGUAJES
778	EVA	c/F, 51	LSI	LENGUAJES
221	LUISA	c/E, 333	FI	FILOLOGÍA

```
select p.*, d.departamento
from profesor p, departamento d
where p.cod = d.cod17
```

Si no existen 2 filas, una en cada tabla con ese valor idéntico que las une, el resultado de la concatenación es vacío (no devuelve ninguna tupla). En nuestro ejemplo sería el caso de que ningún profesor trabajara en ningún departamento (si la columna Profesor.cod almacenara nulos para todas las filas en un determinado estado de la base de datos).

El problema se presenta cuando más de una columna comparte tanto nombre como dominio: el operador intentará unir las filas que comparten los mismos valores en todos esos atributos. Cambiando un poco el ejemplo anterior,

¹⁷ El operador concatenación natural lleva implícita la condición de la cláusula *where*. En realidad, esta expresión en SQL es la descomposición en primitivas de la concatenación natural que se verá más adelante: un producto cartesiano más una selección y una proyección.

podemos pensar que el esquema de *Departamento* tiene como nombre de la columna de descripción *nombre* en vez de *departamento*. La concatenación natural sólo devolverá aquellos profesores y departamentos que cumplan que el profesor trabaja en él y que ambos tienen el mismo nombre.

Profesor				Departamento	
profesor	nombre	dirección	cod	cod	nombre
522	LENGUAJES	c/F, 32	LSI	LSI	LENGUAJES
778	EVA	c/F, 51	LSI	CCIA	CIENCIAS
221	LUISA	c/E, 333	FI	FI	FILOLOGÍA

Profesor \bowtie Departamento			
profesor	nombre	dirección	cod
522	LENGUAJES	c/F, 32	LSI

```
select p.*,d.departamento
from profesor p, departamento d
where p.cod = d.cod and p.nombre =
d.nombre
```

Cuando no se desea este comportamiento, si queremos que la unión de filas se realice únicamente por las columnas etiquetadas como *cod*, estamos obligados a utilizar una combinación de otros operadores. De hecho, la concatenación natural es uno de los operadores derivados, y se puede expresar en función de los operadores básicos selección, proyección y producto cartesiano.

división

Este operador es el de aplicación menos habitual, se utiliza para consultas del tipo “alumnos matriculados en **todas** las asignaturas”. Es, además, el operador con más restricciones a la hora de construir una expresión en AR. El dividendo debe tener más atributos que el divisor, dividendo y divisor han de compartir uno o varios atributos, éstos han de ser los últimos del dividendo y los únicos del divisor y estar en el mismo orden. Los atributos no comunes son la información que se obtiene de la división. En la práctica, sea necesario o no, se suele utilizar la proyección para asegurar que se cumplen todas estas reglas.

En nuestro ejemplo, “alumnos matriculados en **todas** las asignaturas”, usamos la tabla matriculado que es la que representa qué alumnos están matriculados en qué asignaturas. Todas las asignaturas son, evidentemente, las asignaturas almacenadas en la tabla asignatura¹⁸: el atributo común a ambas relaciones es “asignatura”, y la información que queremos obtener es “alumno” (que después se puede concatenar con la tabla Alumno para obtener su “nombre” y “dirección”).

¹⁸ En realidad, debemos identificar ese “todos” al qué se refiere, a qué tabla y a qué datos, puede no ser trivial: “alumnos matriculados en **todas las asignaturas de menos de 6 créditos**”, por ejemplo.

Alumno		
alumno	nombre	dirección
234	LUCÍA	c/A, 3
321	JUAN	c/C, 33
221	LUISA	c/E, 333

Asignatura	
asignatura	nombre
BD1	Bases1
BD2	Bases2
FP2	Prog2

Matriculado	
alumno	asignatura
234	BD1
234	BD2
234	FP2
321	BD1
321	FP2

Matriculado [alumno, asignatura] % (Asignatura [asignatura])

alumno

234

```
select alumno
from matriculado m
where not exists
(select * from asignatura a
  where not exists
    (select * from matriculado m2
     where m2.alumno = m.alumno
       and m2.asignatura =
         a.asignatura))19
```

La razón de que la división tenga unas reglas de composición tan estrictas está en que éste es otro de los operadores derivados, ya que se puede realizar la misma operación con una combinación de diferencias y proyecciones.

IV2.1. el problema de los nombres de los atributos en las relaciones derivadas

Cuando operamos en AR de dos tablas obtenemos una tercera tabla resultado. Esta tabla cumple todas las restricciones del modelo relacional y, por tanto, tiene un esquema de relación que es generado automáticamente. Dicho de otra forma, las relaciones derivadas también tienen nombre para sus atributos pero estos nombres se asignan directamente de los operandos y según el operador utilizado y el orden de tales operandos.

Por ejemplo, la unión, intersección y diferencia dan como resultado una tabla que tiene los mismos nombres de columna que la primera tabla operando.

Alumno (alumno, nombre, dirección)

Profesor (profesor, nombre, dirección)

Alumno \cup Profesor

T(Alumno.alumno, Alumno.nombre, Alumno.dirección)

¹⁹ La expresión “seleccionar aquellos alumnos que están matriculados en **todas** las asignaturas” se puede resolver en SQL si cambiamos “un poco” el enunciado: “seleccionar aquellos alumnos que cumplen que **no existe** ninguna asignatura en la que **no** estén matriculados”

Pero para la concatenación natural los nombres se asignan de diferente forma: todos los nombres de atributo de la primera relación más los no “comunes” de la segunda.

Profesor (profesor, nombre, dirección, cod)

Departamento (cod, descripción)

Profesor \bowtie Departamento

T(**Profesor**.profesor, **Profesor**.nombre,
Profesor.dirección, **Profesor**.cod,
Departamento.descripción)

El producto cartesiano produce tablas con todos los atributos de las dos relaciones operando, pero respetando el orden en que se ha operado:

Alumno (alumno, nombre, dirección)

Profesor (profesor, nombre, dirección)

Alumno \times Profesor

T(**Alumno**.alumno, **Alumno**.nombre,
Alumno.dirección, **Profesor**.profesor,
Profesor.nombre, **Profesor**.dirección)

La división produce una tabla con los atributos no comunes de la primera relación:

Matriculado [alumno, asignatura]

% (Asignatura [asignatura])

T(**Matriculado**.alumno)

Finalmente, la selección y la proyección, al ser operadores con un único operando, la tabla resultado tiene todos (si es una selección) o algunos (si es una proyección) de los atributos de la relación operando.

Alumno donde dirección='c/C, 33'

T(**Alumno**.alumno, **Alumno**.nombre, **Alumno**.dirección)

Alumno [dirección]

T(**Alumno**.dirección)

Es fundamental tener claro cuales son los atributos que resultan de cada operación ya que las expresiones del AR son una secuencia de operaciones cuyo resultado es el operando de la siguiente. La evaluación de las expresiones se realiza de izquierda a derecha salvo si se utilizan paréntesis que alteran el orden de evaluación. Las siguientes expresiones son correctas:

Alumno \times Profesor [Alumno.nombre]

Matriculado [alumno, asignatura]

% (Asignatura [asignatura])

Alumno donde dirección = 'c/C, 33' [alumno, nombre]

Pero las mostradas a continuación son incorrectas:

Alumno \times Profesor [nombre]

hay dos columnas “nombre” en la relación derivada.

Matriculado [alumno, asignatura] % Asignatura [asignatura]

al evaluar de izquierda a derecha la aplicación de los operadores sería proyección, división y, finalmente, proyección. Los atributos del divisor serían todos los de Asignatura, y no son consistentes con las restricciones del operador. Antes de dividir, se debería proyectar sobre Asignatura.

Alumno [alumno, nombre] donde dirección = 'c/C, 33'

si proyectamos antes de seleccionar estamos eliminando precisamente la columna dirección que es la que utiliza la selección en la condición de filtro.

IV2.2.operaciones primitivas.

Las únicas operaciones consideradas como primitivas son la *selección*, *proyección*, *unión*, *producto cartesiano*, y *diferencia*. Así, *concatenación natural*, *intersección* y *división* se pueden expresar en función de las primitivas mencionadas antes:

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

$$R \infty S = ((R \times S) \text{ DONDE } R.B_1=S.B_1 \text{ y... y } R.B_m=S.B_m) \\ [R.A_1, \dots, R.A_n, R.B_1, \dots, R.B_m, S.C_1, \dots, S.C_p]$$

donde los B_i son los atributos comunes a las dos relaciones, los A_i los no comunes de R y los C_i los no comunes de S.

$$R \div S = R[B] - ((R[B] \times S) - R)[B]$$

donde B es el conjunto de atributos no comunes de R.

IV2.3.uso del alias de una relación

Se puede definir un alias para una relación en cualquier ocasión pero su uso está más justificado por operaciones como la siguiente

Matriculado \times Matriculado

Un producto de una tabla por si misma es a veces necesario para ciertas consultas de conteo simple ("alumnos que se han matriculado de al menos 2 asignaturas"). El problema reside en que el esquema de la relación derivada sería:

$T(\text{Matriculado.alumno}, \text{Matriculado.asignatura},$
 $\text{Matriculado.alumno}, \text{Matriculado.asignatura}$)

Evidentemente, en una tabla no pueden existir dos columnas distintas con el mismo nombre (aunque sí en dos tablas distintas). La solución consiste en utilizar uno o dos alias para la relación Matriculado:

definir alias M para Matriculado
 Matriculado \times M

$T(\text{Matriculado.alumno}, \text{Matriculado.asignatura},$

IV3. Resumen de los operadores del Álgebra Relacional

Supongamos que R y S son dos relaciones.

selección

Selecciona aquellas tuplas que cumplen una determinada condición lógica.

Los nombres de los atributos de la relación derivada son los mismos que los de la relación operando.

proyección

Elimina los atributos no especificados.

Los nombres de atributo son los de la lista de parámetros

producto cartesiano

Es la combinación de todas y cada una de las tuplas de la primera relación con todas y cada una de las de la segunda.

Es asociativo y conmutativo, si no tenemos en cuenta el nombre y el orden de los atributos.

Los nombres de atributo son, en este orden, todos los de R y todos los de S.

concatenación natural

Se combinan aquellas tuplas de R y S cuyos valores coinciden para un grupo de atributos comunes. Por atributos comunes entendemos aquellos que tienen el mismo nombre y están definidos en el mismo dominio.

Es asociativo y conmutativo, si no tenemos en cuenta el nombre y el orden de los atributos.

Los nombres de atributo son, en este orden, todos los de R y todos los no comunes de S.

diferencia

Todas las tuplas que están en R pero no en S.

R y S han de ser compatibles.

Los nombres de atributo son todos los de R.

unión

Todas las tuplas que están en R o en S (o en ambas).

R y S han de ser compatibles.

Los nombres de atributo son todos los de R.

intersección

Todas las tuplas que están en R y en S a la vez.

R y S han de ser compatibles.

Los nombres de atributo son todos los de R.

división

Son las tuplas de R que están combinadas que están combinadas con todas las de S. En este caso se da que los atributos de S se encuentran también en R pero ordenados de una manera especial:

- a) el orden de los atributos comunes es igual en R y en S, y además,
- b) en R son los últimos (si “leemos” el esquema de R de izquierda a derecha)

Los nombres de atributo son todos los no comunes de R

IV4. ejemplos

El esquema que se muestra a continuación es el que se va a utilizar para resolver todas las consultas que se plantean a modo de ejemplo. Se especifican los dominios asociados a cada atributo puesto que las comparaciones entre atributos sólo se pueden realizar si sus dominios son idénticos.

ASIGNATURAS (cod_asg: *dCod*, nombre: *dNom*, curso: *dCur*, t: *dHora*, p: *dHora*, l: *dHora*)

ALUMNOS (exp: *dExp*, nombre: *dNom*, dir: *dDir*, ciudad: *dCiudad*, estudios: *dEstudios*)

NOTAS (exp: *dExp*, cod_asg: *dCod*, nota: *dNota*)²⁰

asignaturas

cod_asg	nombre	curso
BD	bases de datos	3
EIN	estructuras	2
AFO	análisis	2
FP	programación	1

alumnos

exp	nombre	estudios
1	pepe	cou
2	ana	cou
3	luisa	fp
4	juan	cou
5	pedro	cou
6	pilar	cou

notas

exp	cod_asg	nota
1	BD	7
1	EIN	5
1	FP	5
1	AFO	6
2	FP	5
2	EIN	6
3	FP	1

²⁰ Nótese que no se ha definido ninguna clave candidata ni ninguna clave ajena. El AR, no hace uso de las definiciones de clave candidata, clave ajena o valor no nulo. Los operadores de AR se basan en comparaciones de valores y, por tanto, la información necesaria y suficiente es la mostrada en este esquema. Por eso, claves ajenas, si se definen, y claves candidatas, definidas obligatoriamente, son aquí meramente una información útil a la hora de escribir una expresión concreta.

En condiciones normales el esquema de base de datos será conforme al modelo relacional, y esquemas de este tipo, incompletos y erróneos según el modelo relacional, no se volverán a proponer.

1) Nombre de los alumnos matriculados en todas las asignaturas de 2º.

Puesto que es un requerimiento del tipo “unos que hacen algo con *todos* los otros”, es con bastante probabilidad una división. Nuestro primer problema es decidir quién hace de dividendo y quién de divisor:

dividendo: “alumnos que se han matriculado de...”

divisor: “asignaturas de segundo”

Así, las tablas a utilizar son:

dividendo: NOTAS

divisor: ASIGNATURAS donde curso = 2

Una vez identificados los datos con los que vamos a trabajar, debemos “arreglar” las expresiones para que cumplan las restricciones de la división:

- dividendo y divisor han de tener atributos comunes, es decir, con el mismo nombre y dominio asociado
- esos atributos han de ser los únicos del divisor, y
- deben estar en el mismo orden en dividendo y divisor, y
- deben ser los últimos del dividendo

De esta forma los atributos no comunes del dividendo van a conformar la tabla derivada de la operación.

dividendo: NOTAS [exp, cod_asg]

divisor: ASIGNATURAS donde curso = 2 [cod_asg]

Sólo falta unir las dos expresiones con el operador de división. Como no se ha establecido ningún orden de precedencia entre los operadores del AR, las expresiones se evalúan de izquierda a derecha excepto si se utilizan los paréntesis, en nuestro caso para “adelantar” la proyección del divisor y asegurar la consistencia de la división.

NOTAS [exp, cod_asg]

÷

(ASIGNATURAS donde curso = 2 [cod_asg])

exp	cod_asg	÷	cod_asg	=	NOTAS.exp
1	BD		EIN		1
1	EIN		AFO		
1	FP				
1	AFO				
2	FP				
2	EIN				
3	FP				

El resultado son todos los números de expediente de aquellos alumnos matriculados en *todo* segundo. Si se concatena con alumnos podemos saber el nombre de esos alumnos.

NOTAS [exp, cod_asg]
 \div (ASIGNATURAS donde curso = 2 [cod_asg])
 ∞ ALUMNOS [nombre]

2) Nombre de los alumnos que sólo se hayan matriculado de asignaturas de 1º.

Las consultas que se refieren a un subconjunto de datos exclusivamente suelen expresarse mediante la palabra “sólo” o “solamente”. Esto quiere decir que no basta simplemente con que una condición se cumpla para ese subconjunto sino que se debe cumplir únicamente para él.

En nuestro caso, no es lo mismo recuperar los “alumnos que se han matriculado de asignaturas de primero” que los “alumnos que se han matriculado *únicamente* de asignaturas de primero”.

En el primer caso se incluyen los alumnos que, aparte de tener alguna asignatura de primero, también pueden tener asignaturas de otros cursos. La siguiente expresión obtiene el número de expediente y el nombre de esos alumnos:

ALUMNOS [exp, nombre] ∞ NOTAS
 ∞ (ASIGNATURAS donde curso = 1) [cod_asg])

En la expresión anterior se ha tenido cuidado de proyectar primero el resultado de la selección sobre asignaturas para evitar que la concatenación se produzca también por nombre de alumno y de asignatura (recuperaríamos alumnos que se llamaran igual que alguna asignatura de primer curso)

¿Cómo obtener los alumnos que sólo se han matriculado de alguna asignatura de primero? Debemos pensar en conjuntos y cambiar un poco la consulta, lo que queremos obtener son alumnos que no se han matriculado de otras asignaturas: “alumnos que se han matriculado de alguna asignatura **menos** los alumnos que se han matriculado de asignaturas que **no** son de primero”.

alumnos matriculados de alguna asignatura

NOTAS [exp]

alumnos matriculados de alguna asignatura que no es de primero

ASIGNATURAS donde curso \neq 1 ∞ NOTAS [exp]

Así contruidos los dos conjuntos sólo falta hacer la resta, nuevamente utilizando los paréntesis para asegurar que las dos subexpresiones generan relaciones compatibles:

NOTAS [exp]
 $-$ (ASIGNATURAS donde curso \neq 1 ∞ NOTAS [exp])

Como nos pedían el nombre de los alumnos que cumplían esta condición concatenamos con alumno para obtener los nombres y ya tenemos la expresión completa

NOTAS [exp] $-$ (ASIGNATURAS donde curso \neq 1 ∞ NOTAS
 [exp])
 ∞ alumnos [nombre]

Nótese que hemos utilizado la tabla NOTAS y no ALUMNOS para la resta. La siguiente expresión daría como resultado “alumnos que se han matriculado sólo de asignaturas de primero o de ninguna asignatura”:

ALUMNOS [exp]
 – (ASIGNATURAS donde curso \neq 1 \cap NOTAS [exp])

3) Número de expediente y nombre de los alumnos que han aprobado todo primero y están matriculados en alguna de tercero.

Aquí el “truco” consiste en dividir la consulta en varias consultas más sencillas: “alumnos que han aprobado todo primero” y “alumnos matriculados en alguna de tercero”. Una vez que hayamos resuelto estas subconsultas sólo tenemos que realizar la intersección para cumplir con la conjunción de condiciones.

Alumnos matriculados en alguna asignatura de tercer curso

ASIGNATURAS donde curso = 3 \cap NOTAS [exp]

Alumnos que han aprobado todo primero

Estamos ante una división en la que el dividendo son “los alumnos que han aprobado...” y el divisor “... todas las asignaturas de primero”

NOTAS donde nota \geq 5 [exp, cod_asg]
 \div (ASIGNATURAS donde curso = 1 [cod_asg])

Como las relaciones derivadas de las dos expresiones anteriores son compatibles podemos combinarlas con el operador intersección y concatenar, finalmente, con alumno para obtener el nombre (poniendo especial cuidado con los paréntesis):

ASIGNATURAS donde curso = 3 \cap NOTAS [exp]
 \cap
 (NOTAS donde nota \geq 5 [exp, cod_asg]
 \div (ASIGNATURAS donde curso = 1 [cod_asg]))
 \cap ALUMNOS [exp, nombre]

4) Nombre de los alumnos que sólo han aprobado una asignatura.

En AR se pueden hacer cuentas sencillas (sería más correcto decir que “contar” en AR resulta trabajoso) utilizando el producto cartesiano y el orden de los atributos de la relación derivada.

Supongamos que hallamos el producto cartesiano de NOTAS por si misma

definir alias N1 para NOTAS
 definir alias N2 para NOTAS²¹
 N1 \times N2

²¹ Con un único alias es suficiente para diferenciar los nombres de atributo de cada uno de los operandos pero se ha hecho así por claridad.

El producto obtiene todos los posibles pares de notas como se muestra en la tabla que acompaña a esta consulta. De todos los pares nos interesan aquellos que se refieren al mismo alumno pero a asignaturas distintas.

$N1 \times N2$
 donde $(N1.exp = N2.exp \text{ y } N1.cod_asig \neq N2.cod_asig)$

Tengamos claro que la tabla resultado contiene filas que contienen dos números de expediente iguales, dos códigos de asignatura diferentes. Ya sólo nos falta preguntar si ha aprobado las dos para conocer cuales son los *alumnos que han aprobado más de una asignatura*:

$N1 \times N2$
 donde $(N1.exp = N2.exp$
 y $N1.cod_asig \neq N2.cod_asig$
 y $N1.nota \geq 5$ y $N2.nota \geq 5)$ $[N1.exp]$

Si estos alumnos los restamos de los que han aprobado alguna asignatura tendremos aquellos *alumnos que sólo han aprobado una única asignatura*:

NOTAS donde $nota \geq 5$ $[exp]$
 -
 $(N1 \times N2$
 donde $(N1.exp = N2.exp$
 y $N1.cod_asig \neq N2.cod_asig$
 y $N1.nota \geq 5$ y $N2.nota \geq 5)$ $[N1.exp])$

Concatenando con la tabla alumnos obtenemos el nombre de esos alumnos y la expresión completa sería:

definir alias N1 para NOTAS
 definir alias N2 para NOTAS

NOTAS donde $nota \geq 5$ $[exp]$ -
 $(N1 \times N2$ donde $(N1.exp = N2.exp$ y $N1.cod_asig \neq$
 $N2.cod_asig$ y $N1.nota \geq 5$ y $N2.nota \geq 5)$ $[N1.exp])$
 \Join Alumnos $[nombre]$

Con los datos del ejemplo que estamos manejando la tabla resultado sería vacía puesto que no hay alumnos que cumplan la condición (haber aprobado algo y sólo una asignatura)

producto cartesiano de NOTAS por si misma y filas donde está la información de los alumnos que han aprobado más de una asignatura; proyectando por N1.exp (o por N2.exp) obtenemos esos alumnos.

N1.exp	N1.cod_asg	N1.nota	N2.exp	N2.cod_asg	N2.nota
1	BD	7	1	BD	7
1	BD	7	1	EIN	5
1	BD	7	1	FP	5
1	BD	7	1	AFO	6
1	BD	7	2	FP	5
1	BD	7	2	EIN	6
1	BD	7	3	FP	1
1	EIN	5	1	BD	7
1	EIN	5	1	EIN	5
1	EIN	5	1	FP	5
1	EIN	5	1	AFO	6
1	EIN	5	2	FP	5
1	EIN	5	2	EIN	6
1	EIN	5	3	FP	1
1	FP	5	1	BD	7
1	FP	5	1	EIN	5
1	FP	5	1	FP	5
1	FP	5	1	AFO	6
1	FP	5	2	FP	5
1	FP	5	2	EIN	6
1	FP	5	3	FP	1
1	AFO	6	1	BD	7
1	AFO	6	1	EIN	5
1	AFO	6	1	FP	5
1	AFO	6	1	AFO	6
1	AFO	6	2	FP	5
1	AFO	6	2	EIN	6
1	AFO	6	3	FP	1
2	FP	5	1	BD	7
2	FP	5	1	EIN	5
2	FP	5	1	FP	5
2	FP	5	1	AFO	6
2	FP	5	2	FP	5
2	FP	5	2	EIN	6
2	FP	5	3	FP	1
2	EIN	6	1	BD	7
2	EIN	6	1	EIN	5
2	EIN	6	1	FP	5
2	EIN	6	1	AFO	6
2	EIN	6	2	FP	5
2	EIN	6	2	EIN	6
2	EIN	6	3	FP	1
3	FP	1	1	BD	7
3	FP	1	1	EIN	5
3	FP	1	1	FP	5
3	FP	1	1	AFO	6
3	FP	1	2	FP	5
3	FP	1	2	EIN	6
3	FP	1	3	FP	1

Algunos consejos.

Las siguientes afirmaciones no tienen el carácter de verdades absolutas sino que son guías que, en función del requerimiento a resolver, pueden servir para “intuir” la solución final.

- La concatenación se usa para obtener una ocurrencia que cumple una condición alguna vez, o para obtener datos relacionados.
- La diferencia se utiliza en enunciados del tipo “... sólo...”, o para eliminar ocurrencias que no cumplen una condición.
- La división en enunciados del tipo “... todos...”
- La intersección en enunciados con dos condiciones complejas sobre un mismo concepto: “... $c()$ y $c()$...” (para condiciones sencillas suele ser suficiente con una selección)
- La unión: “... $c()$ o $c()$...”
- El producto cartesiano en comparaciones de tuplas de una relación con otras tuplas de la misma relación para obtener “primeros” o “últimos”. También cuando la concatenación natural no se puede realizar porque hay más atributos comunes de los deseados o por tener diferentes nombres.

IV5. Referencia

A continuación se detallan las definiciones formales de los operadores antes descritos y sus propiedades.

Sean $R(A_1:D_1, A_2:D_2, \dots, A_n:D_n)$ y $S(B_1:E_1, B_2:E_2, \dots, B_m:E_m)$ dos esquemas de relación.

producto cartesiano	operación definición atributos de la relación derivada	$R \times S$ $R \times S = \{ t \mid t = rs \text{ y } \forall r \in R \forall s \in S \}$ $R.A_1, R.A_2, \dots, R.A_n, S.B_1, S.B_2, \dots, S.B_m$
proyección	operación definición atributos de la relación derivada	$R[B]$ Sea $B = \{ B_1 = A_o, B_2 = A_p, \dots, B_k = A_q \}$ una lista de nombres de atributos de R $R[B] = \{ t \mid t = \langle v_1, v_2, \dots, v_k \rangle \text{ y } \exists r \in R \text{ que cumple que } v_i \text{ es igual al valor de } B_i \text{ en } r, i = 1, 2, \dots, k \}$ $R.A_o, R.A_p, \dots, R.A_q$

selección	operación definición	<p><i>R DONDE f(t)</i> Sea $f(t)$ es una expresión lógica sobre alguno o todos los atributos de R</p> <p>$R \text{ DONDE } f(t) = \{ t \mid t \in R, f(t)=\text{cierto} \}$</p>
	atributos de la relación derivada	$R.A_1, R.A_2, \dots, R.A_n$
	construcción de $f(t)$	<p>Sea α un operador de comparación: $<, >, \leq, \geq, =, \neq$.</p> <p>Conectivas lógicas $\text{Y}, \text{O}, \text{NO}$</p> <p>Fórmulas bien formadas (FBF): $A \alpha a$ es una FBF donde $\exists i(A = R.A_i \text{ y } a \in D_i)$ $A \alpha B$ es una FBF donde $\exists i(A = R.A_i \text{ y } \exists j(B = S.B_j \text{ y } D_i = E_j))$ Si f y g son FBF entonces $f \text{ Y } g$ y $f \text{ O } g$ también lo son. Si f es una FBF entonces también lo son (f) y $\text{NO } f$. Ninguna fórmula que no se construya según las reglas anteriores es una FBF.</p> <p>Evaluación de $f(t)$ Dada $t \in R$, $f(t)$ es cierta para esa tupla si al sustituir en f cada nombre de atributo por su correspondiente valor en t, f se evalúa a cierto.</p> <p>Orden de precedencia de los operadores: $()$ NO $<, >, \leq, \geq, =, \neq$ Y, O</p>

diferencia	operación definición	<p>$R - S$ $R - S = \{ t \mid t \in R \text{ y } t \notin S \}$</p>
	atributos de la relación derivada	$R.A_1, R.A_2, \dots, R.A_n$

concatenación natural	operación	$R \bowtie S$
	definición	Sean $C = \{ j \mid \exists_j (A_j = B_j \text{ y } D_j = E_j) \}$ $f(t) = \text{cierto si } \forall j \in C (A_j:v_j = B_j:v_j)$ $R \bowtie S = R \times S \text{ DONDE } f(t) [A_1, \dots, A_n, B_{n+1}, \dots, B_{n+k}] \forall j = n+1, \dots, n+k j \notin C$
	atributos de la relación derivada	$R.A_1, R.A_2, \dots, R.A_n, \{S.B_k\} \forall k \notin C$
	propiedades	conmutativa, asociativa ²²
unión	operación	$R \cup S$
	definición	$R \cup S = \{ t \mid t \in R \text{ o } t \in S \}$
	atributos de la relación derivada	$R.A_1, R.A_2, \dots, R.A_n$
	propiedades	conmutativa, asociativa
intersección	operación	$R \cap S$
	definición	$R \cap S = \{ t \mid t \in R \text{ y } t \in S \} = R - (R - S)$
	atributos de la relación derivada	$R.A_1, R.A_2, \dots, R.A_n$
	propiedades	conmutativa, asociativa
división	operación	$R \div S$
	definición	Sean $C = \{ j \mid \exists_j (A_j = B_j \text{ y } D_j = E_j) \}$ $A = \{ A_i:D_i \mid i \notin C \}$ $R \div S = R[A] - ((R[A] \times S) - R)[A]$
	atributos de la relación derivada	$R.A_1, R.A_2, \dots, R.A_k \forall i = 1, 2, \dots, k i \notin C$

BIBLIOGRAFÍA

[ELMASRI & NAVATHE 2001]

[CELMA 1997]

[DATE 2000]

²² Siempre que se dice que una operación en AR es asociativa o conmutativa se entiende que nos estamos refiriendo a las tuplas de la tabla derivada ya que ni los nombres de atributo ni su orden dentro de las tuplas son los mismos según sea el orden de los operandos.

V INTRODUCCIÓN AL DISEÑO DE BASES DE DATOS RELACIONALES

La teoría de la normalización va perdiendo peso con el paso de los años como herramienta de diseño de bases de datos relacionales en favor de modelos de datos más ricos en su representación, como pueda ser el Entidad-Relación de Chen que veremos en temas posteriores.

Aún así, los conceptos que se van a desarrollar aquí son totalmente válidos desde el punto de vista de comprobar lo correcto que es un esquema de base de datos relacional, en el sentido de que no se produzcan redundancias innecesarias entre los datos a almacenar.

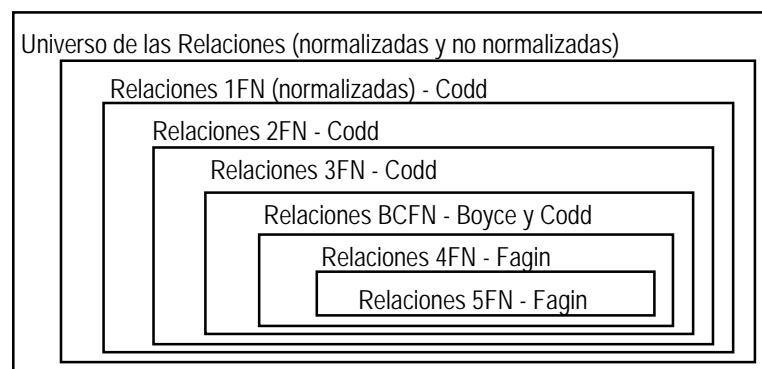
Por otra parte, ayudará a la mejor comprensión del Modelo Relacional y, en general, a justificar la estructuración en tablas (relaciones en su denominación formal) interrelacionadas mediante claves ajenas de los sistemas de información a mecanizar mediante técnicas de bases de datos.

V1. Introducción

Cuando trabajamos con una base de datos relacional, los esquemas de las distintas relaciones que la constituyen nos indican que “cada dato tiene su lugar”. Pero, ¿qué ocurre si se modifican estas estructuras lógicas?. Muchas veces es tan obvio que un dato debe de almacenarse en una de las relaciones y no en otra que se nos escapa la respuesta a porqué es así.

La teoría de la normalización es en esencia una expresión formal de ideas sencillas con una aplicación muy práctica en el área del diseño de bases de datos, ya **que conducen a una correcta elección del esquema de la base de datos.**

Ya que el diseño de la base de datos es el diseño de los esquemas que la componen, debemos recordar que lo que nos interesa son las propiedades de los datos que siempre se cumplen y no las que se cumplen por casualidad en un instante de tiempo. El propósito del esquema es capturar aquellas propiedades que siempre son verdaderas, es decir nos vamos a ocupar de **la cabecera o esquema** de la relación independientemente del cuerpo o valores variables con el tiempo.



El punto fundamental es que algunas relaciones pueden presentar aspectos indeseables y la teoría de la normalización nos permite identificar esos casos y nos muestra la forma de convertir esas relaciones a una forma más deseable. Para ello, **la teoría de la normalización tiene como fundamento el concepto de formas normales.** Se dice que **una relación está en una determinada forma normal si satisface un cierto conjunto de restricciones.** Cuantas más restricciones satisface “más deseable” es la forma de la relación.

Las formas normales que se han definido se muestran en el gráfico anterior junto con el nombre de quien (o quienes) las han definido. Como da idea la ilustración, cada conjunto de relaciones normalizadas en una determinada forma, además de las suyas propias, cumple con las restricciones impuestas por la forma normal anterior. En otras palabras, la quinta forma normal es más restrictiva que la cuarta, la cuarta que la de Boyce-Codd, y así sucesivamente.

Se debe aclarar que formalmente cuando se habla de **relaciones normalizadas** nos estamos refiriendo a relaciones que están en **primera forma normal (1FN)**, es decir, los términos “normalizada” y “1FN” significan exactamente lo mismo.

Además de ver la definición de las distintas formas normales se introduce la idea de un **procedimiento de normalización adicional**, con el que una

relación en una cierta forma normal, por ejemplo en 2FN, se puede convertir en un conjunto de relaciones en una forma más deseable, 3FN, y así sucesivamente. Es decir, este procedimiento es la **reducción sucesiva de un conjunto dado de relaciones a una forma más deseable**. Es importante destacar que este procedimiento es **reversible**, siempre es posible tomar la salida del procedimiento y convertirla en la entrada, lo que significa que **no se pierde información** durante el proceso de normalización adicional.

Las tres formas normales originales de Codd y la forma normal de Boyce/Codd se asientan en el concepto de **dependencia funcional**. Se dará primero una definición este concepto para poder definir posteriormente cuando una relación está en una cierta forma normal.

Ejemplo propuesto

El ejemplo que se va a seguir para aclarar todas las definiciones trata de forma muy reducida de una empresa que tiene una serie de empleados, de los que se conoce su N.I.F., su nombre, su dirección, un teléfono de contacto y su año de nacimiento. Esta empresa tiene una serie de proyectos identificados unívocamente por un código y de los que se conoce su nombre, su presupuesto, su fecha de inicio, su fecha de finalización, la ciudad donde se desarrolla y el grado de idoneidad del proyecto, éste tiene que ver con la facilidad de desplazamiento desde la ciudad donde se encuentra la sede de la empresa hasta la ciudad donde se desarrolla el proyecto y depende no sólo de los kilómetros sino también de los medios de comunicación que existan entre ellos según un estudio hecho por la propia empresa. A los empleados los pueden destinar a trabajar en distintos proyectos durante un período de tiempo establecido pudiendo ir alternando su trabajo en distintos proyectos.

Si partimos de que vamos a trabajar con un SGBD Relacional y abordamos la fase de diseño conceptual usando el modelo EER, un posible diagrama que recogería nuestro sistema de información sería el mostrado a continuación en el diagrama.

Si realizamos el diseño lógico partiendo de este diagrama, el esquema de la base de datos que obtendremos prescindiendo de los dominios asociados a los atributos es el que se recoge a continuación:

EMPLEADO (N.I.F., nombre, dirección, fecha-nacimiento)

Clave primaria: N.I.F.

CIUDAD(ciudad, idóneo)

Clave primaria: ciudad

PROYECTO (código, nombre, presupuesto, inicio, finalización, ciudad)

Clave primaria: código

Clave ajena: ciudad → CIUDAD

TRABAJAR (N.I.F., código, desde, hasta)

Clave primaria: (N.I.F., código, desde)

Clave ajena: N.I.F. → EMPLEADO

Clave ajena: código → PROYECTO

Pero, ¿es completamente correcta la forma en que pensamos almacenar la información? Hay cosas que tenemos muy claro que “están en su sitio” porque parece “natural”. Por ejemplo, parece lógico que si, además,

queremos almacenar el teléfono personal de cada empleado, este dato debe estar en la tabla EMPLEADO.

Supongamos que no, que el teléfono del empleado lo incluimos en la tabla TRABAJAR.

TRABAJAR (N.I.F., **teléfono**, código, desde, hasta)
 Clave primaria: (N.I.F., código, desde)
 Clave ajena: N.I.F. → EMPLEADO
 Clave ajena: código → PROYECTO

Una posible ocurrencia de la tabla TRABAJAR sería:

N.I.F.	TELÉFONO	CÓDIGO	DESDE	HASTA
22446688A	5632224	A111	10/10/92	20/12/92
22446688A	5632224	A112	09/01/93	12/03/93
22446688A	5632224	A112	20/03/93	14/03/93
22446688A	5632224	A116	15/08/93	24/11/93
11116666B	5211111	A112	09/01/93	20/05/93
11116666B	5211111	A114	07/07/93	27/10/93

Como vemos, el teléfono de contacto de un empleado se va a repetir tantas veces como proyectos esté asignado. Esta redundancia puede provocar información inconsistente si no se manejan con cuidado las posibles actualizaciones del teléfono de contacto de un empleado.

Este problema surge porque de todas las columnas de la tabla TRABAJAR el teléfono de contacto de un empleado lo podemos conocer sabiendo su N.I.F. pero por ejemplo la fecha en la que comienza a trabajar en un proyecto no depende sólo de su N.I.F. sino que necesitamos conocer también el código del proyecto al que nos referimos.

En general, la redundancia introducida por un mal diseño de las relaciones provoca lo que se conoce como **anomalías de actualización**, es decir, problemas con las tres operaciones básicas: inserción, borrado y modificación.

INSERCIÓN

Para almacenar el teléfono de un empleado debemos asignarlo, obligatoriamente a un proyecto (recordemos que la clave primaria incluye el código del empleado).

N.I.F.	TELÉFONO	CÓDIGO	DESDE	HASTA
22446688A	965632224	A111	10/10/92	20/12/92
22446688A	965632224	A112	09/01/93	12/03/93
22446688A	965632224	A112	20/03/93	14/03/93
22446688A	965632224	A116	15/08/93	24/11/93
11116666B	965211111	A112	09/01/93	20/05/93
11116666B	965211111	A114	07/07/93	27/10/93
21555777C	965227820	?	?	?

Además, puede que, para un mismo empleado, tengamos varios teléfonos distintos, cuando debería ser único para cada uno.

N.I.F.	TELÉFONO	CÓDIGO	DESDE	HASTA
22446688A	965632224	A111	10/10/92	20/12/92
22446688A	965632224	A112	09/01/93	12/03/93

22446688A	965632224	A112	20/03/93	14/03/93
22446688A	965632224	A116	15/08/93	24/11/93
11116666B	965211111	A112	09/01/93	20/05/93
11116666B	965211111	A114	07/07/93	27/10/93
11116666B	965212222	A111	10/10/92	20/12/92

BORRADO

Podría eliminarse información innecesariamente.

Si un empleado trabajaba en un único proyecto y esa relación ha terminado, lo lógico es eliminar la tupla que representa la relación entre ambos. Sin embargo, entre la información a borrar de la base de datos está el teléfono del empleado, dato éste que perderíamos innecesariamente.

N.I.F.	TELÉFONO	CÓDIGO	DESDE	HASTA
22446688A	965632224	A111	10/10/92	20/12/92
22446688A	965632224	A112	09/01/93	12/03/93
22446688A	965632224	A112	20/03/93	14/03/93
22446688A	965632224	A116	15/08/93	24/11/93
11116666B	965211111	A112	09/01/93	20/05/93

MODIFICACIÓN

Proceso costoso computacionalmente y posibilidad de introducir inconsistencias en la base de datos.

La redundancia de información que presenta la relación TRABAJAR, hace que si un empleado cambia de teléfono tengamos que modificar este dato en todas las tuplas de ese empleado.

Todo esto no ocurriría si el teléfono hubiera sido definido como un atributo más de la relación EMPLEADO en vez de en TRABAJAR.

Por tanto, parece que no es adecuado mantener toda la información en la misma tabla. Los problemas surgen porque ciertos atributos no dependen directa o completamente de las claves candidatas.

La normalización es un proceso por el cual se modifica la estructura de la base de datos buscando la eliminación de la redundancia innecesaria, a obtener un conjunto de tablas con una estructura más deseable.

Vamos a definir lo que se entiende por "dependencia".²³

²³ En realidad, todo son consideraciones de diseño de una base de datos, más o menos cercanas a la realidad pero, finalmente, decisiones del diseñador. Éste busca el conjunto de relaciones entre los datos más apropiado para el uso final de esa base de datos. Sin ser habitual, por características propias y excepcionales del sistema de información a representar, podría llegar a decidirse que el teléfono es único para cada dirección.

V2. dependencia funcional

Dada una relación R , el atributo Y de R depende funcionalmente del atributo X de R -en símbolos, $R.X \rightarrow R.Y$ y se lee “ $R.X$ determina funcionalmente a $R.Y$ ”- si y sólo si, como mucho, un solo valor Y en R está asociado a cada valor X en R (en cualquier momento dado). Los atributos X e Y pueden ser compuestos.

Por ejemplo en la relación EMPLEADO, los atributos nombre, dirección, teléfono de contacto y fecha de nacimiento dependen funcionalmente del atributo N.I.F., porque dado un valor específico de N.I.F., existe sólo un valor correspondiente de nombre, dirección, teléfono y fecha de nacimiento. En símbolos:

EMPLEADO.NIF \rightarrow EMPLEADO.NOMBRE
 EMPLEADO.NIF \rightarrow EMPLEADO.DIRECCIÓN
 EMPLEADO.NIF \rightarrow EMPLEADO.TELÉFONO
 EMPLEADO.NIF \rightarrow EMPLEADO.FECHA

o, de forma más concisa:

EMPLEADO.NIF \rightarrow EMPLEADO.(NOMBRE, DIRECCION, TELEFONO, FECHA)

Sin embargo, dada una dirección no ocurre que sólo haya un empleado con esa dirección, puede que más de un empleado viva en la misma casa²⁴.

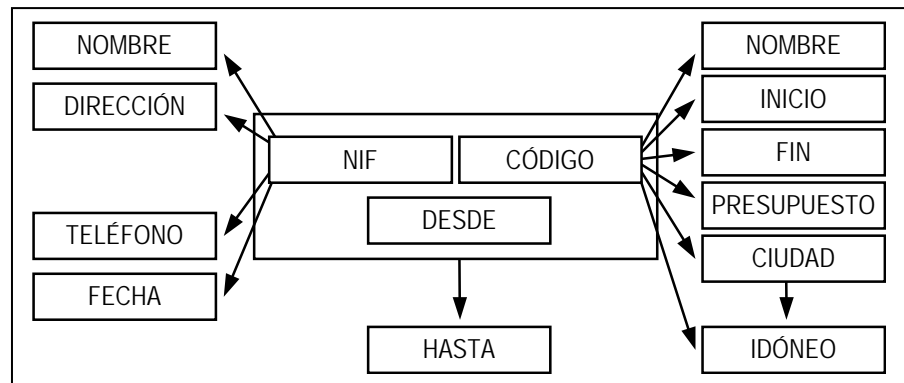
~~EMPLEADO.DIRECCION \rightarrow EMPLEADO.NIF~~

Por decirlo de otra forma:

Dada una relación R , el atributo Y depende funcionalmente del atributo X de R , si y sólo si, siempre que dos tuplas de R concuerden en su valor de X , deben por fuerza concordar en su valor de Y .

De forma gráfica, las dependencias funcionales se representan como arcos dirigidos cuyo inicio es el atributo (o atributos) del que depende el atributo al que apunta. Así, y como ya se ha dicho antes de NIF dependen funcionalmente tanto NOMBRE, DIRECCIÓN, TELÉFONO, como FECHA, como se observa en el siguiente **diagrama de dependencias funcionales**:

²⁴ Nuevamente hay que tener en cuenta que las dependencias funcionales las decide el diseñador de la base de datos: en el sistema de información que estamos manejando no existen restricciones del tipo “sólo hay un empleado, como mucho, por cada dirección”, o similares. Pero en otro sistema de información sí podría ser necesario representar estas restricciones.

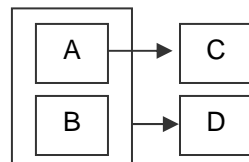


Nótese que todas estas dependencias funcionales están dentro del esquema de base de datos que estamos manejando como ejemplo, aún cuando los atributos se encuentren repartidos entre varias tablas relacionadas con las correspondientes claves ajenas. Más adelante, con las formas normales, se justificará porqué es este el esquema y no otro.

DEPENDENCIA FUNCIONAL COMPLETA

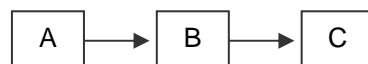
Se dice que **el atributo Y de la relación R es por completo dependiente funcionalmente del atributo X de la relación R** si depende funcionalmente de X y no depende funcionalmente de ningún subconjunto propio de X (es decir, no existe un subconjunto propio Z de los atributos componentes de X tales que Y sea funcionalmente dependiente de Z).

En el siguiente diagrama, C depende de forma completa de A, pero no de (A, B). D sí depende de forma completa de (A,B).

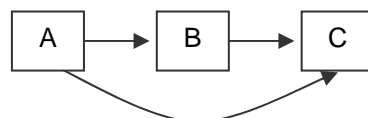


DEPENDENCIA FUNCIONAL TRANSITIVA

Sean X, Y, y Z atributos de la relación R. Si Z depende funcionalmente de Y, e Y depende funcionalmente de X, se dice que **el atributo Z depende funcionalmente del atributo X de forma transitiva**.



Aunque no se represente, existe otro arco entre A y C puesto que esa es, precisamente, la propiedad de transitividad. En ocasiones aparecerá explícitamente como en el siguiente, pero ambos diagramas son absolutamente equivalentes:



V3. formas normales

Las que vamos a desarrollar aquí son las primeras formas normales que definió Codd. Posteriormente se vio que eran insuficientes para tratar algunos casos especiales, básicamente aquellos que manejaban varias claves candidatas. De hecho las definiciones que se dan a continuación hablan de Clave Primaria y no de Clave Candidata.

Por otra parte, para la mayoría de las relaciones que se describen en una base de datos, suele ser suficiente normalizar hasta tercera forma normal (3FN) para conseguir un diseño correcto de la BD.

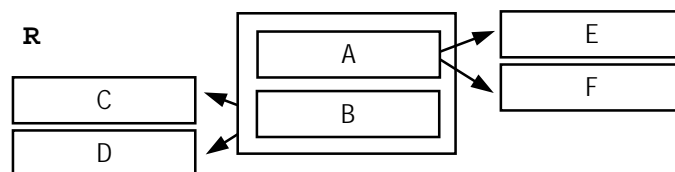
V3.1. primera forma normal

Nosotros partimos de la idea de que por la estructura de las tablas cada tupla únicamente admite un valor para cada uno de los atributos. Esto es lo que se expresa en la definición de primera forma normal.

Una relación está en primera forma normal (1FN) si y sólo si todos los dominios simples subyacentes contienen sólo valores atómicos.

V3.2. segunda forma normal

Una relación está en segunda forma normal (2FN) si y sólo si está en 1FN y todos los atributos no clave dependen por completo de cualquier clave candidata.²⁵



Este diagrama representa una relación con la siguiente definición:

$R(A, B, C, D, E, F)$ CP(A, B)

La elección de la clave primaria es evidente: el único conjunto de atributos del que dependen funcionalmente todos los demás es (A, B). Si hubiéramos elegido (A) sólo podríamos "llegar" a E y a F, con (B) a ninguno, y (A, B, C), por ejemplo, no puede ser clave candidata porque (A, B) ya lo es.

En el diagrama de dependencias funcionales anterior podemos observar como los atributos C y D dependen funcionalmente de forma completa de la

²⁵ Al hablar de atributos no clave o *no primos* de una relación se hace referencia a los que no pertenecen a ninguna de las claves candidatas de la relación.

clave primaria (A, B). Pero E y F sólo dependen de A, por lo que decimos que esta relación no se encuentra en 2FN.

El proceso para pasar de una relación en una determinada forma normal a un conjunto de relaciones en una forma normal más deseable, *procedimiento de normalización*, tiene como **operador de descomposición la proyección** (el de recomposición es la *concatenación natural*).

El primer paso del procedimiento de normalización es transformar una relación que sólo está en 1FN a varias en 2FN (algunas estarán además en tercera o más). Esto hace que tengamos que sacar proyecciones para eliminar las dependencias funcionales no completas de la clave primaria (si una relación está en 1FN pero no está en 2FN su clave primaria es compuesta).

Dada una relación **R** en 1FN:

R (A, B, C, D, E, F)
Clave primaria: (A, B)

Ya hemos comentado que son E y F los atributos que no dependen funcionalmente de forma completa de la clave primaria (A, B); se recomienda sustituir R por sus dos proyecciones **R1** y **R2**.

R1 (A, E, F)
Clave primaria: A

R2 (A, B, C, D)
Clave primaria: (A, B)
Clave ajena: A → R1

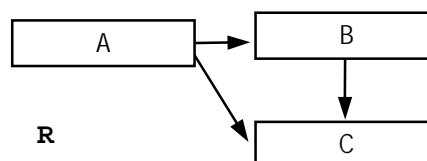
Ambas relaciones, R1 y R2 parten de una misma relación original: R. Es evidente que si, antes, toda la información estaba asociada por pertenecer a una única relación, al efectuar la proyección de atributos, las relaciones resultantes han de relacionarse por alguna clave ajena.

V3.3. tercera forma normal

Una relación está en tercera forma normal (3FN) si y sólo si está en 2FN y todos los atributos no clave dependen de manera no transitiva de cualquier clave candidata.

Hablamos de dependencia transitiva cuando ocurre:

$A \rightarrow B, B \rightarrow C$ y, por tanto, $A \rightarrow C$



Siguiendo con el proceso de normalización veamos cuales son los pasos a seguir para descomponer una relación **R** que sólo está en 2FN en otras

relaciones más deseables en 3FN. Las proyecciones irán encaminadas a eliminar las dependencias transitivas.

R (A, B, C)
Clave primaria: A

En realidad, la DF que provoca las anomalías en las actualizaciones es $R.B \rightarrow R.C$.

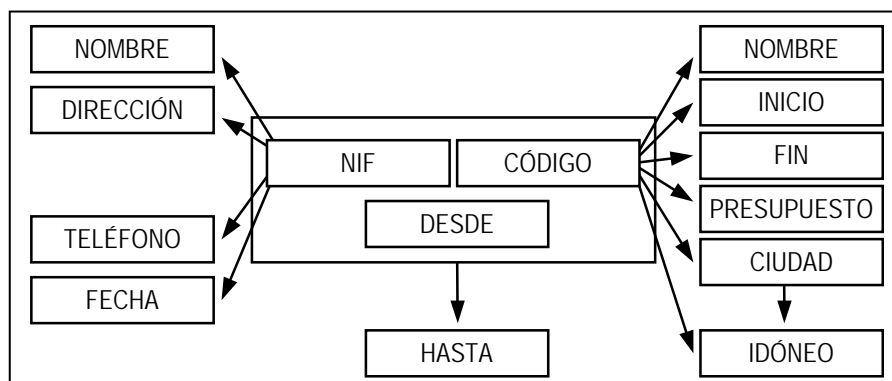
Se recomienda sustituir **R** por sus dos proyecciones **R1** y **R2**:

R1 (B, C) **R2 (A, B)**
Clave primaria: B Clave primaria: A
Clave ajena: $B \rightarrow R1$

Estas dos relaciones se encuentran en 3FN.

V3.4. Un ejemplo

Volvamos al diagrama de dependencias funcionales de empleados y proyectos.



En primer lugar debemos definir la primera relación:

$R(NIF, nombreE, dirección, fecha-nacimiento, código, nombreP, presupuesto, inicio, finalización, ciudad, idóneo, desde, hasta)$ ²⁶
CP: (código, NIF, desde)

Todas las DF de NIF y todas las de CÓDIGO son DF no completas, lo que provoca la definición de dos nuevas tablas, ya que R no está en 2FN:

$R(NIF, código, desde, hasta)$
CP: (código, NIF, desde)
CAj: (código) \rightarrow PROYECTO
CAj: (NIF) \rightarrow EMPLEADO

²⁶ Tenemos dos atributos llamados "nombre" por lo que, en esta primera relación, los vamos a diferenciar como "nombreE" y "nombreP".

EMPLEADO (NIF, nombre, dirección, fecha-nacimiento)
 CP: (NIF)
 PROYECTO (código, nombre, presupuesto, inicio, finalización, ciudad, idóneo)
 CP: (código)

La relación R (que podemos renombrar como TRABAJAR) está en 2FN y 3FN puesto que la única DF que contiene es (código, NIF, desde) → hasta.

No obstante, hay que comprobar las dos nuevas tablas. EMPLEADO tiene, también, una única dependencia funcional: NIF → (nombre, dirección, fecha-nacimiento), por lo tanto está en 3FN, pero PROYECTO tiene:

Código → (nombre, presupuesto, inicio, finalización, ciudad, idóneo)
 Ciudad → idóneo

PROYECTO no está en 3FN y debemos proyectar una nueva tabla:

CIUDAD(ciudad, idóneo)
 CP: (ciudad)
 PROYECTO (código, nombre, presupuesto, inicio, finalización, ciudad)
 CP: (código)
 CAj: (ciudad) → CIUDAD

Ahora ya está el esquema completo:

EMPLEADO (NIF, nombre, dirección, fecha-nacimiento) CP: (NIF) CIUDAD(ciudad, idóneo) CP: (ciudad) PROYECTO (código, nombre, presupuesto, inicio, finalización, ciudad) CP: (código) CAj: (ciudad) → CIUDAD TRABAJAR(NIF, código, desde, hasta) CP: (código, NIF, desde) CAj: (código) → PROYECTO CAj: (NIF) → EMPLEADO
--

V4. forma normal de boyce-codd

Esta forma normal se definió para resolver las anomalías de actualización provocadas por un caso muy específico, que se da cuando:

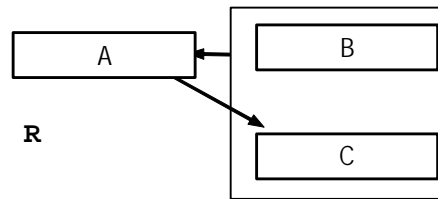
- La relación tiene varias claves candidatas
- Las claves candidatas están solapadas en algún subconjunto de atributos de las dos.

Si cualquiera de estas dos condiciones no se cumpliera, normalizar hasta 3FN sería suficiente.

Una relación está en forma normal de Boyce-Codd (FNBC) si y sólo si todo determinante es una clave candidata.

Definimos **determinante** como un conjunto de atributos del que depende funcionalmente por completo algún otro atributo.

Supongamos la siguiente relación y sus atributos, sus claves candidatas y las dependencias funcionales observadas:



Los determinantes en esta relación son (A) y (B,C) puesto que de todos ellos depende funcionalmente de forma completa un conjunto de atributos.

Las claves candidatas son (A,B) y (B,C) ya que de estos conjuntos de atributos depende el resto.

Por eso, R no está en FNBC. Para normalizar y conseguir que todo determinante sea clave candidata debemos proyectar $(A) \rightarrow (C)$, la dependencia que provoca que la relación no esté en FNBC, a otra relación, quedando entonces:

R1 (A, B)

Clave primaria: (A, B)

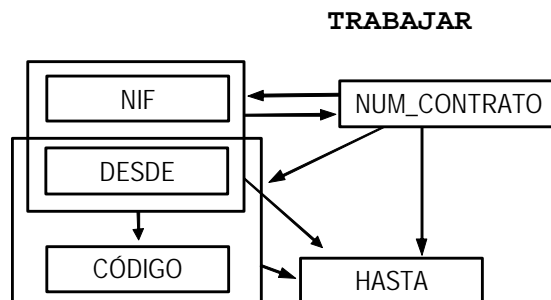
Clave ajena: $(A) \rightarrow \mathbf{R2}$

R2 (A, C)

Clave primaria: (A)

V5. Un ejemplo

Supongamos una relación trabajar que debe representar las siguientes restricciones semánticas:



- un proyecto se descompone en varios períodos de ejecución, desde-hasta.
- un empleado se asigna a los proyectos por períodos de ejecución y cada asignación constituye un número de contrato único para la empresa.
- un empleado no puede estar trabajando en dos proyectos al mismo tiempo.
- si a un empleado se le contrata para trabajar en varios períodos de un proyecto, aunque sean consecutivos, significa varias asignaciones y varios números de contrato.

- un empleado es contratado para cubrir períodos completos de los distintos proyectos, nunca parciales.

Volvamos ahora a la relación TRABAJAR pero teniendo en cuenta que hay una característica más, el número de contrato.

TRABAJAR (NUM_CONTRATO, NIF, CÓDIGO, DESDE, HASTA)

¿Cuáles serían las *claves candidatas* de esta relación?

Al modificar los requisitos del sistema de información, ahora es (NIF, DESDE) una clave candidata. Otra posible clave candidata de la relación es (NUM_CONTRATO). Por lo tanto, sólo es necesario comprobar la 3FN puesto que las claves no están solapadas.

Aclarado este punto, las únicas dependencias funcionales completas que encontramos en la relación son:

NUM_CONTRATO → (NIF, DESDE, CÓDIGO, HASTA)
 (NIF, DESDE) → (NUM_CONTRATO, CÓDIGO, HASTA)
 (CÓDIGO, DESDE) → HASTA

Por lo tanto, todos los atributos no primos depende de forma completa de cualquier clave candidata: la relación está en 2FN.

Sin embargo, el atributo HASTA depende de forma transitiva de (NUM_CONTRATO): la relación no está en 3FN. Proyectando esta DF en una nueva tabla obtenemos:

PERÍODOS (CÓDIGO, DESDE, HASTA)
 Clave Primaria: (CÓDIGO, DESDE)

TRABAJA (NUM_CONTRATO, NIF, DESDE, CÓDIGO)
 Clave Primaria: (NIF, DESDE)
 Clave Alternativa: NUM_CONTRATO
 Clave Ajena: (CÓDIGO, DESDE) → PERÍODOS

VI LA PERSPECTIVA LÓGICA DEL MODELO RELACIONAL

1. INTRODUCCIÓN
2. CÁLCULO DE PREDICADOS DE PRIMER ORDEN
3. UNA BASE DE DATOS COMO UNA INTERPRETACIÓN DE UN LPO
4. FÓRMULAS SEGURAS
5. CÁLCULO RELACIONAL

Hemos visto que el álgebra relacional consiste en la especificación de una secuencia de operaciones sobre las tablas de la base de datos de tal forma que obtenemos una nueva relación resultado. Manejamos una lista de operadores de los que conocemos, básicamente, su resultado. Por ejemplo, la diferencia dará como resultado todas las filas del primer operando que no pertenezcan también al segundo. O la selección, que obtiene las tuplas que cumplen una determinada condición.

Imaginemos, ahora, que vamos a trabajar con expresiones que, dependiendo de sus argumentos, se evalúan a cierto o falso, del tipo *CLIENTE(x)* en la que si un valor está actualmente almacenado en la tabla CLIENTE, sustituyendo *x* con ese valor el resultado de la expresión es *verdadero*. Imaginemos, también, que un módulo de consultas es capaz de probar todos los valores posibles y mostrarnos en pantalla aquellos que hacen cierta la expresión anterior: ya tenemos un listado de clientes.

Aún más, podemos pensar en otra expresión tal como *CLIENTE(x)* y *x.ciudad='Alicante'*. Si sustituimos todos los valores posibles en *x*, los que pertenezcan a la tabla y su atributo ciudad contenga 'Alicante' serán los que hagan cierta esta expresión. Estamos dando por supuesto, claro está, que *x* es una tupla. Estamos expresando, mediante una fórmula qué características ha de tener la información esperada, "qué queremos obtener".

Sólo necesitamos unas reglas para escribir estas expresiones, una sintaxis para comunicarnos con ese módulo de consultas imaginario y los criterios por los que se obtendrán valores de cierto o falso. En este tema veremos una nueva forma de describir una base de datos usando conceptos del cálculo de predicados de primer orden.

VI1. introducción

Además del álgebra relacional (AR), Codd desarrolló dos lenguajes de especificación de bases de datos relacionales basados en el **cálculo de predicados de primer orden** (CPPO). Estos lenguajes son el **cálculo relacional orientado a tuplas** y el **cálculo relacional orientado a dominios**.

El AR es cómodo para nosotros, en cierta medida, puesto que la forma de escribir consultas se asemeja a escribir expresiones aritméticas. Además, sus operadores se pueden alinear (más o menos) con las principales cláusulas de la orden *select* de SQL.

Pues bien, con los cálculos relacionales, se describe el conjunto de información deseado mediante *fórmulas lógicas de primer orden*. Así, la consulta se convierte en algo así como “qué valores de la base de datos hacen cierta la fórmula”²⁷.

Lo que necesitamos es una guía (un “manual”) de cómo construir las fórmulas y cómo evaluarlas, cuando son ciertas y cuando falsas. Nuevamente, Codd hace uso de una teoría matemática ya existente, el CPPO. Si para el álgebra relacional realizó una adaptación del modelo relacional a la teoría de conjuntos (la definición de ciertos operadores, básicamente), para el cálculo relacional esta adaptación consiste en la definición de una interpretación de un lenguaje de primer orden y del concepto de fórmula segura.

Este tema trata del fundamento matemático que hay detrás del cálculo relacional orientado a tuplas y del orientado a dominios, ese conocimiento previo necesario para poder escribir fórmulas y saber cómo obtener su resultado.

Dicho de otro modo, cómo podemos describir una base de datos relacional en términos del CPPO. Para ello, debemos disponer de un **lenguaje de primer orden** y de una **interpretación** de ese lenguaje. El lenguaje de primer orden lo constituyen los símbolos que se utilizan al escribir las fórmulas, y la interpretación la asignación de valores a esos símbolos, lo que nos permite definir cuando una fórmula se evalúa a cierto o falso.

VI2. cálculo de predicados de primer orden

El CPPO, y en general la lógica, nos permite hacer deducciones sobre un universo de discurso. Por ello, es imprescindible disponer de un lenguaje preciso que nos permita describir aquellos aspectos relevantes de la realidad objeto de estudio. Este lenguaje, que llamaremos **lenguaje de primer orden** (LPO), consta de unos símbolos y unas reglas precisas para combinarlos en expresiones sintácticamente correctas, en fórmulas. Una **interpretación** de un LPO, define el valor de verdad de tales fórmulas.

²⁷ En realidad, simplemente “qué valores hacen cierta la fórmula”; al adaptar el modelo relacional al CPPO nos vamos a asegurar de que los datos con los que trabajamos son los de la ocurrencia de la base de datos (se verá en fórmulas seguras)

Podemos utilizar nuestro idioma para ilustrar estos dos conceptos. Una frase escrita contiene, o puede contener, nombres, verbos, adjetivos, adverbios, artículos, pronombres, etc. Algunas palabras tienen carga semántica (significan algo, se pueden encontrar en un diccionario) y otras son modificadores o conectores. Por ejemplo: “este señor dice tonterías” es una frase correctamente escrita según nuestra gramática, pero no lo es “dice señor este tonterías”. Pensemos ahora en un ciudadano de la ciudad de Taipei: salvo que, por esas casualidades, aprendiera español en algún momento de su vida, ni tan siquiera la primera frase tiene sentido para él. Necesita conocer las palabras y, tanto o más importante, qué significan.

Asimilemos, pues, el LPO a las palabras y las reglas para construir frases, y la interpretación a un diccionario que nos dice qué significan esas palabras (pueden tener varios significados) y qué sentido tienen en una frase concreta. Por ejemplo, aunque según la sintaxis es correcto decir “este árbol dice tonterías”, la interpretación nos diría que no tiene sentido, que los árboles no hablan.

Nuestro objetivo final es ver como podemos adaptar el modelo relacional y verlo como una interpretación de un lenguaje de primer orden. Así pues, estamos obligados a conocer que es un LPO y que es una interpretación.

VI2.1.lenguaje de primer orden

Un lenguaje de primer orden es un conjunto de símbolos, un **alfabeto**, y unas reglas que nos dicen que unas secuencias de símbolos son correctas y otras no (unas están “bien escritas” y otras no). Los símbolos podemos dividirlos en dos tipos, aquellos a los que se les asigna valor y aquellos que actúan como conectores o modificadores. Como veremos seguidamente, variables, constantes, predicados y funciones pertenecen al primer tipo, y cuantificadores, conectivas lógicas, y símbolos de comparación y puntuación al segundo. La forma de combinar estos símbolos correctamente se concreta en el concepto de **fórmula bien formada**, basado en la definición de **fórmula atómica**, y éste a su vez en la de **término**.

alfabeto

Un lenguaje de primer orden, L , viene definido por un par (A, F) , donde A es un alfabeto de símbolos y F el conjunto de todas las expresiones sintácticamente correctas (fórmulas bien formadas) que se pueden construir utilizando los símbolos de A .

Como ya se ha dicho, el alfabeto A contiene las siguientes clases de símbolos:

- Variables
- Constantes
- Símbolos de función

La declaración de la cantidad de argumentos de una función se hace mediante puntos separados por comas. Por ejemplo, $f(. , .)$ representa a una función de 2 parámetros.

- Símbolos de predicado

Al igual que para las funciones se ha de indicar la cantidad de argumentos.

- Símbolos de comparación: $<$ $>$ $<=$ $>=$ $<>$
- Símbolos de puntuación: $($ $)$ $,$
- Conectivas lógicas: \neg \wedge \vee \rightarrow
Negación, conjunción, disyunción e implicación.
- Cuantificadores: \exists (existencial), \forall (universal)

F es el conjunto de todas las expresiones que se pueden construir combinando los símbolos anteriores. La definición de **fórmula bien formada** establece estas reglas de combinación de símbolos pero, dado que esta definición se basa en ellos, antes es necesario introducir los conceptos de **término** y **fórmula atómica**.

término

Un término se define recursivamente como sigue:

- Un símbolo de constante es un término.
- Un símbolo de variable es un término.
- Si f es un símbolo de función de n argumentos y t_1, t_2, \dots, t_n son términos, entonces $f(t_1, t_2, \dots, t_n)$ es un término.
- Nada más es un término.

fórmula atómica

Si P es un símbolo de predicado de n argumentos y t_1, t_2, \dots, t_n son términos, entonces $P(t_1, t_2, \dots, t_n)$ es una fórmula atómica o átomo.

fórmula bien formada

Una fórmula bien formada (FBF) se define recursivamente aplicando las siguientes reglas:

- Una fórmula atómica es una FBF.
- Si F y G son FBF's, entonces también lo son:

$$\neg F$$

$$F \wedge G$$

$$F \vee G$$

$$F \rightarrow G$$
- Si x es un símbolo de variable y F es una FBF, entonces también lo son:

$$\forall x F$$

$$\exists x F$$
- Si F es una FBF, entonces también lo es (F) .
- Nada más es una FBF.

En definitiva, una FBF se construye conectando fórmulas atómicas y estas últimas mediante símbolos de predicado y términos. El objetivo de estas definiciones es establecer las reglas de “escritura”. Por ejemplo, $\forall x P(x) \rightarrow Q(x)$ es una FBF, pero las siguientes no lo son:

$$x \forall P(x) \rightarrow Q(x)$$

$$\forall x \rightarrow Q(x)$$

$$\forall (x P(x) \rightarrow (x))$$

Nótese que en ningún momento hemos hablado de “resultados”, simplemente estamos aprendiendo a “escribir” pero todavía no sabemos lo que “decimos” (para esto necesitamos la interpretación).

fórmulas abiertas y cerradas

Este concepto nos será muy útil cuando, finalmente, interroguemos a la base de datos. Adelantándonos al punto donde se expone su aplicación, digamos que las **fórmulas abiertas** son las que obtienen una lista de valores (un resultado en pantalla, por entendernos), y valores de cierto o falso las **cerradas**, con lo que se pueden utilizar éstas para especificar restricciones de integridad del tipo “los clientes de Alicante no pueden tener descuento”.

La diferencia entre unas y otras se basa en la utilización de los cuantificadores existencial y universal. Cuando una variable está en el alcance de un cuantificador se dice que es una **ocurrencia de variable ligada**, en contraposición con la ocurrencia de variable **libre**, no afectada por cuantificador alguno.

El alcance de un cuantificador es la FBF que se encuentra inmediatamente a su derecha. En la expresión $\forall x P(x) \rightarrow Q(x)$, la primera FBF a la derecha del cuantificador es $P(x)$, por lo que este es el alcance del cuantificador universal, y $Q(x)$ queda fuera de él.

A partir de ahora, supongamos que x e y son símbolos de variable y que $P(\cdot)$, $Q(\cdot)$, $R(\cdot, \cdot)$ son símbolos de predicado. Las siguientes FBF ilustrarán los conceptos de formulas abiertas y cerradas.

$$\forall x P(x) \rightarrow Q(x)$$

La primera ocurrencia de la variable x , la que aparece como argumento del predicado $P(\cdot)$, es ligada, esto es, está bajo el alcance del cuantificador universal, pero la segunda ocurrencia es libre.

Nótese que cuando un símbolo de variable se utiliza, al mismo tiempo, en ocurrencias libres y ligadas, su evaluación cuando se defina la interpretación se hará como si utilizáramos símbolos distintos. Es decir, que la fórmula podría haberse escrito igualmente como $\forall x P(x) \rightarrow Q(y)$.

$$\forall x (P(x) \rightarrow Q(x)) \vee \neg P(x)$$

En esta fórmula, y por la acción de los paréntesis, las dos primeras ocurrencias de x están bajo el alcance del cuantificador, y la interpretación que se defina las considerará como la misma variable, mientras que la tercera y última es libre y considerada una variable distinta. Al igual que antes, la fórmula $\forall x (P(x) \rightarrow Q(x)) \vee \neg P(y)$ será equivalente.

$$\exists x (R(x, y) \vee P(x))$$

Ahora todas las ocurrencias de x son ligadas mientras que la de y es libre.

$$\exists x \exists y (R(x, y) \vee P(x))$$

Ahora todas las ocurrencias de variable son ligadas. Para entendernos, si pudiéramos sustituir cada FBF por un símbolo, el alcance del segundo cuantificador es $\exists x \exists y F$, y el alcance del primero es $\exists x F'$.

Esta clasificación de las ocurrencias de una variable nos permite distinguir entre **fórmulas cerradas**, aquellas que no tienen ocurrencias libres de variable, y **fórmulas abiertas**, con al menos una ocurrencia libre de variable.

De los ejemplos anteriores, las tres primeras fórmulas son ejemplos de fórmula abierta, y la cuarta de fórmula cerrada.

VI2.2. interpretación

Un lenguaje de primer orden nos permite efectuar deducciones sobre un **universo de discurso**, siendo éste el conjunto de todos los valores posibles. Sin embargo, aunque sabemos construir expresiones, debemos dotar a cada símbolo del lenguaje de un “contenido”, establecer los valores que definen la evaluación a *cierto* o *falso* de las fórmulas.

En este sentido pretendemos que:

- las **constantes** denoten objetos (individuos) de ese universo de discurso.
- los **predicados** denoten propiedades sobre los objetos del universo de discurso.
- las **fórmulas bien formadas** sean enunciados o sentencias sobre el universo.

Dicho de otra forma, dado un lenguaje de primer orden $L=(A, F)$, el objetivo es la asignación a cada símbolo del alfabeto A de un valor del universo de discurso de forma que, utilizando esta asignación como base, podamos definir el valor de verdad de cualquier fórmula de dicho lenguaje. Para ello introducimos el concepto de **interpretación**.

interpretación

Una interpretación I de un lenguaje de primer orden, $L=(A, F)$, es una cuádrupla (D, K, H, E) donde:

- D es un conjunto no vacío, llamado dominio de I , en el que las variables de A toman valores, y que constituye el **universo de discurso**.
- K es una aplicación que asigna a cada símbolo de constante un elemento de D .

- H es una aplicación que asigna a cada símbolo de función n -aria una función de D^n en D .
- E es una aplicación que asigna a cada símbolo de predicado n -ario una relación sobre D^n (esta relación se denomina extensión del predicado).

Para ilustrar la definición anterior, vamos a suponer que D es el conjunto de los números enteros. Supongamos, también, que $L=(A, F)$ está bien definido (tiene símbolos de constante, de variable, de predicado, etc.).

De forma general, una constante es un valor del dominio. Otra cosa es el símbolo que utilizamos para representarlo. En nuestra vida cotidiana utilizamos ideas intangibles, manejamos objetos, nos relacionamos con personas, y cuando pretendemos comunicar a otros esos conceptos o referirnos a esas cosas o individuos utilizamos símbolos, palabras, que representan a esas ideas, objetos o personas, justo lo que estamos haciendo nosotros al escribir este texto y al leerlo.

Todos sabemos, hablando de los enteros, lo que significa el “uno” o el “dos”. Para referirnos a ellos escribimos ‘1’ y ‘2’. El número entero 1 es un concepto, una idea, que para poder ser manejado e interpretado de igual forma por todos, está representado por un símbolo (gráfico en nuestro caso) que es el carácter ‘1’. Luego, antes de poder utilizar estos caracteres en una expresión aritmética como $1 + 2$, se han asignado los valores del dominio a los símbolos de constante. Lo que pasa es que no somos conscientes de esa asignación, la hemos aprendido desde pequeños en la escuela y es la que se maneja en todo el mundo porque necesitamos comunicarnos y ser entendidos.

K define esa asignación. Si suponemos que el conjunto de símbolos de constante está compuesto por $\{1, 2, 3, 4, \dots\}$, $K(1) = 1$, $K(2) = 2$, $K(3) = 3$, ..., o de forma general:

$$K = \{ (c, d), c \text{ es un símbolo de constante, } d \text{ es un entero} \}$$

Es decir, el argumento de K es un símbolo y el resultado un valor del dominio (cuando escribimos ‘1’ estamos representando al entero 1). Por eso mismo, ya que la interpretación la definimos nosotros, también podíamos haber establecido que $K(1) = 2$ y $K(2) = 1$.

Obviamente, si así fuera necesario, el conjunto de constantes podría ser otro, por ejemplo $\{a, b, c\}$, y K definido como $\{(a,1), (b,2), (c,3)\}$ (a representa al entero 1 , b al 2 , y c al 3).

H y E realizan la misma tarea para los símbolos de función y de predicado. Si $f(.,.)$ es un símbolo de función podríamos asignarle una función de D^2 en D , con una expresión general como por ejemplo $f(x,y)=x+y$, o de forma explícita como $f(1,1)=2, f(1,2)=3, f(2,1)=3, \dots$

La asignación de extensión a un símbolo de predicado consiste en establecer los valores que hacen cierto el predicado (como se verá más adelante), por ejemplo para un predicado de un argumento $P(.)$ el conjunto de valores $\{1, 3, 5\}$ o para otro de dos argumentos $Q(.,.)$ el conjunto $\{(1,1), (1,2), (2,2), (2,4)\}$.

De esta forma, $P(1)$ y $Q(1,2)$ se evalúan a cierto, mientras que $P(5)$ y $Q(3,3)$ a falso. También $Q(f(1,1),2)$ se evalúa a cierto.

VI2.3. evaluación de fórmulas

En realidad, no debemos olvidar que nuestro objetivo final es hacer deducciones acerca del universo de discurso, y esto se consigue evaluando

las fórmulas que se precisen. Es decir, dada una fórmula, ésta será cierta o falsa, o informará sobre qué elementos del universo de discurso la hacen cierta.

Si nos adelantamos una vez más, su aplicación a bases de datos relacionales es clara: mediante fórmulas queremos saber si los datos almacenados en ella cumplen una cierta condición o no, o recuperar un conjunto de información.

Por evaluar fórmulas entendemos obtener una respuesta según la interpretación que se haya definido. Las reglas de evaluación se distinguen según el tipo de fórmula, cerrada o abierta. Cuando la fórmula es cerrada, el resultado es cierto o falso, y cuando es abierta, devuelve una lista de valores. En todos los casos, se asume el orden de precedencia de los operadores que se muestran en el siguiente cuadro, de tal forma que fórmulas entre paréntesis serán evaluadas en primer lugar, y fórmulas con conjunciones, disyunciones o implicaciones serán las últimas en evaluarse:

()
< , > , <= , >= , <>
\forall , \exists
\neg
\wedge , \vee , \rightarrow

evaluación de fórmulas cerradas

La evaluación de una fórmula cerrada G en una interpretación da como resultado el valor *cierto* o *falso*, y se define como sigue:

- Si G es una fórmula atómica, $G = P(t_1, t_2, \dots, t_n)$, y (a_1, a_2, \dots, a_n) una tupla tal que a_i sustituye a t_i , entonces G se evalúa a *cierto* si la tupla está en la extensión que la interpretación asigna a P , y a *falso* en caso contrario.
- Si G contiene alguna conectiva lógica, se evalúa de acuerdo a las siguiente tabla (F y H son fórmulas bien formadas).

F	H	$F \wedge H$	$F \vee H$	$F \rightarrow H$	$\neg F$
cierto	cierto	cierto	cierto	cierto	falso
falso	cierto	falso	cierto	cierto	cierto
cierto	falso	falso	cierto	falso	
falso	falso	falso	falso	cierto	

- Si $G = \forall x F$ entonces G es *cierta* si, al sustituir la variable x en F por todos los elementos del dominio, F es cierta siempre, y G es *falsa* en cualquier otro caso.
- Si $G = \exists x F$ entonces G es *cierta* si F es cierta al menos para un elemento del dominio que sustituya la variable x en F , y G es *falsa* en cualquier otro caso.

evaluación de fórmulas abiertas

- El resultado de evaluar una fórmula abierta G con n ($n > 0$) variables libres en una interpretación, es una relación n -aria, R_G , definida sobre el dominio de la interpretación D .

Cada tupla de esta relación es tal que, al sustituir las variables libres por las correspondientes componentes de la tupla, la fórmula cerrada que resulta es cierta en la interpretación. Si la relación R_G

coincide con D^n la fórmula se evalúa simplemente a *cierto*; si R_G no contiene ninguna tupla, entonces la fórmula se evalúa a *falso*.

Resumiendo, con el LPO definimos los símbolos y la forma de las fórmulas que los utilizan, y con la interpretación establecemos los valores que definirán el resultado de esas fórmulas.

VI2.4.un ejemplo

Sea A el alfabeto de un lenguaje de primer orden:

Variables = $\{x, y\}$

Constantes = $\{a, b, c\}$

Predicados = $\{P(\cdot), Q(\cdot), R(\cdot, \cdot)\}$

Sea $I(D, K, H, E)$ definida como sigue

$D = \{1, 2, 3\}$

$K(a) = 1, K(b) = 2, K(c) = 3$

$E(P) = \{2\}, E(Q) = \{1, 3\}, E(R) = \{(1,1), (1,2), (1,3)\}$

F1	$Q(x)$
----	--------

Esta es una fórmula abierta por lo que el proceso de evaluación consiste en sustituir la variable x por todos y cada uno de los valores del dominio y comprobar si la fórmula cerrada correspondiente es cierta o falsa, que es lo mismo que decir que el resultado de esta fórmula son todos los valores pertenecientes a la extensión del predicado.

Como $Q(1)$ y $Q(3)$ se evalúan a cierto y $Q(2)$ a falso, el resultado de la formula consiste en dos tuplas de una componente:

$$Q(x) = \{1, 3\}$$

F2	$Q(x) \wedge \forall y R(x, y)$
----	---------------------------------

Esta es otra fórmula abierta pero se combinan variables ligadas y libres. Por la precedencia de operadores, podemos resumir la fórmula como $F \wedge G$, por lo que ambas subfórmulas han de ser ciertas para que toda la fórmula lo sea; los únicos valores que hacen cierto $Q(x)$ son el 1 y el 3.

$$Q(1) \wedge R(1, 1) = \text{cierto}$$

$$Q(1) \wedge R(1, 2) = \text{cierto}$$

$$Q(1) \wedge R(1, 3) = \text{cierto}$$

Cuando x vale 1, se cumple que en R hay una tupla para todos los elementos del dominio en la que la primera componente es 1. Luego el valor 1 hace cierta la fórmula $Q(x) \wedge \forall y R(x, y)$.

$$Q(3) \wedge R(3, 1) = \text{falso}$$

Puesto que el par $(3,1)$ no pertenece a la extensión de R , ya hemos encontrado un valor de y para el que la fórmula se

hace falsa, luego ya no se cumple para todo valor de y , y toda la fórmula se evalúa a falso.

El resultado final de la fórmula $Q(x) \wedge \forall y R(x, y)$ es una única tupla:

$$Q(x) \wedge \forall y R(x, y) = \{1\}$$

G1	$\forall x(P(x) \rightarrow \neg Q(x))$
----	---

Esta fórmula es cerrada ya que las dos ocurrencias de la variable x están en el alcance del cuantificador universal, por lo que la fórmula se describiría como que todo valor que esté en P no lo está en Q . Podemos decir que vamos a trabajar con una única variable. Así, debemos comprobar todos los valores del dominio, y si para todos ellos la fórmula se evalúa a cierto, es que la fórmula es cierta.

$$(P(1) \rightarrow \neg Q(1)) = (\text{falso} \rightarrow \neg \text{cierto}) = (\text{falso} \rightarrow \text{falso}) = \text{cierto}$$

$$(P(2) \rightarrow \neg Q(2)) = (\text{cierto} \rightarrow \neg \text{falso}) = (\text{cierto} \rightarrow \text{cierto}) = \text{cierto}$$

$$(P(3) \rightarrow \neg Q(3)) = (\text{falso} \rightarrow \neg \text{cierto}) = (\text{falso} \rightarrow \text{falso}) = \text{cierto}$$

$$\forall x(P(x) \rightarrow \neg Q(x)) = \text{cierto}$$

G2	$\forall x(P(x) \rightarrow \exists y R(x, y))$
----	---

Para $x=1$ y $x=3$ el antecedente de la implicación es falso por lo que, valga lo que valga $\exists y R(x, y)$, el resultado es cierto.

No existe ningún par en R cuya primera componente sea 2, $\exists y R(2, y)$ es falso por lo que la fórmula $P(2) \rightarrow \exists y R(2, y)$ se evalúa a falso.

Como no se cumple para todo valor de x , la fórmula se evalúa a falso:

$$\forall x(P(x) \rightarrow \exists y R(x, y)) = \text{falso}$$

G3	$\exists x(P(x) \vee \exists y R(x, y))$
----	--

Se puede comprobar que para $x=1$, la fórmula se evalúa a cierto. Si se cumple para un valor ya podemos decir que el resultado es:

$$\exists x(P(x) \vee \exists y R(x, y)) = \text{cierto}$$

VI2.5. definición de modelo

Dada una interpretación I y un conjunto de fórmulas bien formadas T , se dice que **I es modelo para T** si y sólo si todas las fórmulas de T se evalúan a cierto en I .

En el ejemplo anterior, si únicamente consideramos las fórmulas cerradas $G1$, $G2$ y $G3$ la interpretación no es modelo para ellas ya que $G2$ se evalúa a falso. Sin embargo la interpretación sí es modelo para las fórmulas $G1$ y $G3$.

VI3. una base de datos relacional como una interpretación de un lenguaje de primer orden.

VI3.1. otro enfoque de las bases de datos relacionales

Ya disponemos de las herramientas necesarias para efectuar la adaptación del modelo relacional al CPPO. La idea central es probar que una base de datos puede ser interrogada mediante fórmulas lógicas, fórmulas bien formadas. Lo que a continuación se va a exponer es otro punto de vista, otra forma de “presentar” las bases de datos relacionales.

Si para construir y evaluar una FBF necesitamos una interpretación y un LPO, para todo esquema de base de datos relacional (BDR) y para cada estado de base de datos tenemos que asegurarnos que ambos, interpretación y lenguaje, están bien definidos.

De hecho, cuando trabajamos con una BDR, cada operación provoca un nuevo estado de base de datos mientras que la estructura, el esquema de base de datos permanece inalterado. El LPO se basará en el esquema y cada estado es una interpretación distinta.

Por ejemplo, supongamos una BDR muy simple, con una tabla de departamentos y otra de personas que trabajan en esos departamentos de tal forma que un empleado trabaja como máximo en un departamento y estos pueden tener tantos empleados como necesiten (relación uno a muchos). El esquema de base de datos, entre otras cosas, define los dominios en los que los atributos toman valores. Supongamos que todos los atributos son cadenas de 20 caracteres. Todas las combinaciones de caracteres en cadenas de 0 a 20 elementos son susceptibles de ser almacenadas. Éste es nuestro universo de discurso, el dominio, y cada una de esas combinaciones tendrá su correspondiente símbolo de constante.

Por otro lado, necesitamos predicados para poder determinar qué valores “están” y qué valores “no están” en la BD. Si cada tabla, departamento y persona, se define como un predicado, ya disponemos del LPO, ya podemos escribir fórmulas.

Para trabajar normalmente con la BD, esas fórmulas deben devolver un resultado, deben poder evaluarse, necesitamos una interpretación del LPO. Si el esquema de BD es estático, la información almacenada cambia constantemente con las inserciones, borrados y modificaciones. Por tanto, una consulta que nos da un resultado en un estado de la BD, podría darnos otro en otra ocurrencia.

Como hemos visto, la interpretación es la herramienta que rellena los símbolos del LPO, da valor a las constantes y a los símbolos de predicado. En el caso de las constantes es simplemente asegurarnos de que existe una correspondencia entre el símbolo y el valor del dominio. Volviendo a nuestra pequeña base de datos las constantes ya están definidas y son siempre las mismas.

Los predicados son los que van a definir, realmente, la ocurrencia de la base de datos, los datos que están almacenados en cada instante en las tablas. Si

suponemos que cada estado de la BD define una interpretación, ya estamos en disposición de evaluar cualquier fórmula.

Insistimos en que, simplemente, se trata de otro enfoque de una base de datos relacional, lo que antes llamábamos relación (tabla) ahora se llama predicado, lo que antes eran dominios ahora es universo de discurso... como se muestra en las equivalencias que se muestran a continuación.

dominios:	universo del discurso
valor de un dominio:	constante
relación:	predicado
extensión de la relación:	extensión del predicado
atributos:	argumentos de predicado
consulta:	FBF abierta
restricción de integridad:	FBF cerrada
estado de base de datos:	interpretación

Ahora, en vez de utilizar el modelo relacional para describir la BD, vamos usar una interpretación de un LPO:

Todo esquema de base de datos relacional define un lenguaje de primer orden y cada estado de base de datos una interpretación.

Resumiendo, siempre podemos utilizar el CPPO para describir e interrogar a una base de datos relacional. Se entiende que en cada estado de base de datos disponemos de:

- universo del discurso
Sea D la unión de todos los dominios que aparecen en el esquema de BD.
- constantes
Por cada elemento de D, se introduce un símbolo de constante y nada más que uno. Para simplificar, escogeremos como símbolos de constante los símbolos que denotan a los elementos de D; a cada símbolo se le asigna “su” elemento.²⁸
- predicados
Por cada esquema de relación de n atributos en el esquema de la BD, se introduce un símbolo de predicado de n argumentos. Para simplificar, escogeremos como símbolos de predicado los nombres de las relaciones. A cada símbolo se le asigna la extensión de “su” relación.

En este alfabeto, por simplicidad, no incluimos los símbolos de función²⁹. Los símbolos de variables se asume que serán tantos como sean necesarios, y las conectivas lógicas, especiales y los cuantificadores son los mismos que los definidos en el apartado anterior. Así mismo, las FBF's de este lenguaje se construyen con las mismas reglas que se han presentado en dicho apartado.

Puesto que disponemos en todo momento de esa interpretación, podemos construir fórmulas lógicas y evaluarlas. Estas fórmulas pueden ser abiertas o

²⁸ Es otra vez la discusión del lenguaje de primer orden: desde un punto de vista formal, una cosa es el símbolo '1' y otra cosa el entero 1; una cosa es el símbolo 'Pepe' y otra el valor del dominio de las cadenas de 10 caracteres Pepe. Tenemos conceptos y símbolos que representan a esos conceptos. Nosotros, puesto que utilizamos símbolos para comunicarnos, no somos conscientes de esa asignación, la asumimos el día en que “aprendimos a hablar”.

²⁹ Aunque se podrían utilizar si fuera necesario. Al igual que las constantes, son en cierto modo estáticas ya que tendrían la misma definición en todos los estados de base de datos (en todas las interpretaciones).

cerradas. Recordemos que las restricciones de integridad son condiciones que se desean de los datos, un control de su almacenamiento. Se entiende que integridad referencial y de clave son mantenidas por los sistemas de gestión de base de datos. Pero también podemos desear introducir restricciones de existencia que el esquema no puede reflejar como, por ejemplo, que los departamentos han de tener al menos un empleado.

¿Cómo podemos estar seguros de que nuestra BD, en un estado concreto, es correcta, cumple con todas las restricciones de integridad? Aquí es cuando recuperamos el concepto de modelo. Si la interpretación (el estado de la BD) es modelo para el conjunto de fórmulas cerradas con las que expresamos esas restricciones, es que los datos están correctamente almacenados.

VI3.2.un ejemplo

Partiendo del siguiente esquema, vamos a ilustrar la formalización lógica de una BDR, esto es, la descripción de todos sus posibles estados. A partir de esta descripción, y dada la ocurrencia concreta, se propondrán fórmulas abiertas y cerradas, asumiendo que las primeras representan consultas que han de devolver un conjunto de tuplas, y las segundas un conjunto de restricciones de integridad.

Esquema de la base de datos:

CONDUCTOR (número : cadena(2), añoNac : entero)

Clave Primaria: número

VEHÍCULO (matrícula : cadena(9), marca : cadena(30), añoFab : entero)

Clave Primaria: matrícula

CONDUCE (conductor : cadena(2), vehículo : cadena(9))

Clave Primaria: (conductor, vehículo)

Clave Ajena: conductor → CONDUCTOR

Clave Ajena: vehículo → VEHÍCULO

Restricciones de integridad:

R1: Los vehículos sólo pueden ser conducidos por conductores (integridad referencial de la relación CONDUCE respecto a la relación CONDUCTOR).

R2: Los conductores sólo conducen vehículos" (I.R. de CONDUCE respecto de VEHÍCULO)

R3: Todo vehículo tiene al menos un conductor

Extensión de las relaciones:

CONDUCTOR

número	añoNac
C1	1950
C2	1950
C3	1972
C4	1970

VEHÍCULO

matrícula	marca	añoFab
A-0000-A	SEAT	1980
A-1111-BM	SEAT	1990
A-2222-CB	VOLKSWAGEN	1994
A-3333-CN	AUDI	1995

CONDUCE

conductor	vehículo
C1	A-0000-A
C2	A-0000-A
C1	A-2222-CB
C4	A-3333-CN

Formalmente, un esquema base de datos relacional y su ocurrencia se describen, desde el punto de vista del CPPO, como:

Sea $L = (A, F)$ e $I(D, K, H, E)$, donde

- símbolos de constante:

$$C = \{ a : a \text{ es una cadena}(2) \} \cup \{ a : a \text{ es una cadena}(9) \} \cup \{ a : a \text{ es una cadena}(30) \} \cup \{ a : a \text{ es un entero} \}$$

- símbolos de predicados:

$$P = \{ \text{CONDUCTOR}(\dots), \text{VEHÍCULO}(\dots), \text{CONDUCE}(\dots) \}$$

- dominio de la interpretación:

$$D = \{ a : a \text{ es una cadena}(2) \} \cup \{ a : a \text{ es una cadena}(9) \} \cup \{ a : a \text{ es una cadena}(30) \} \cup \{ a : a \text{ es un entero} \}$$

- Asignación a las constantes:

$$K : C \rightarrow D \text{ tal que } K = \{ (c, d) : c \in C \text{ y } d \in D \text{ y } c=d \}^{30}$$

- Asignación a los predicados:

$$E(\text{CONDUCTOR}) = \text{Ext}(\text{CONDUCTOR})$$

$$E(\text{VEHÍCULO}) = \text{Ext}(\text{VEHÍCULO})$$

$$E(\text{CONDUCE}) = \text{Ext}(\text{CONDUCE})$$

Se supone que todo posible estado de base de datos se puede describir de esta forma, por lo que en todo momento podemos evaluar una fórmula escrita en este LPO en esta interpretación. Una BD así formalizada, puede interrogarse con naturalidad mediante fórmulas bien formadas del lenguaje que tiene asociado de manera que, el resultado de evaluar esas fórmulas en la interpretación asociada a esa base de datos proporciona la contestación a la pregunta.

¿Qué conductores han nacido en 1950?

F1	CONDUCTOR(x, 1950)
----	----------------------

x
C1
C2

¿Qué vehículos son conducidos por el conductor número C2?

F2	CONDUCE('C2', y)
----	------------------

y
A-0000-A

¿Qué conductores no conducen ningún vehículo?

F3	$\exists y (\text{CONDUCTOR}(x, y) \wedge \neg \exists z (\text{CONDUCE}(x, z))$
----	--

x
C3

³⁰ Puesto que c es un símbolo de constante y d es un valor del dominio, la expresión $c=d$ quiere decir que "c se escribe igual que d"

Por otra parte, como ya se ha comentado, las restricciones de integridad representan propiedades semánticas que deben cumplirse para que la ocurrencia de la base de datos sea válida. Con la formalización que se ha presentado, estas restricciones de integridad pueden expresarse mediante un conjunto de fórmulas cerradas de modo que la interpretación asociada a la base de datos debe ser modelo de ese conjunto de fórmulas.

“los vehículos sólo pueden ser conducidos por conductores”

$$F4 \quad \forall x \forall y (\text{CONDUCE}(x, y) \rightarrow \exists z \text{CONDUCTOR}(x, z))$$

se evalúa a *cierto*

“Los conductores sólo conducen vehículos”

$$F5 \quad \forall y \forall x (\text{CONDUCE}(x, y) \rightarrow \exists z \exists t \text{VEHÍCULO}(y, z, t))$$

cierto

“todo vehículo tiene al menos un conductor”

$$F6 \quad \forall x \forall y \forall z (\text{VEHÍCULO}(x, y, z) \rightarrow \exists t \text{CONDUCTOR}(t, x))$$

falso

Si suponemos que F4, F5 y F6 es el conjunto de fórmulas que representan restricciones del sistema, la interpretación no es modelo para este conjunto de fórmulas, o lo que es lo mismo, nuestra ocurrencia del esquema no es consistente con las restricciones explícitas del problema: la base de datos no es correcta.

VI4. fórmulas seguras

Como se ha visto en el apartado anterior, en una base de datos vista como una interpretación de un lenguaje de primer orden, las preguntas se expresan mediante fórmulas abiertas, y las restricciones de integridad se expresan con fórmulas cerradas de ese lenguaje.

Sin embargo nuestra base de datos es un subconjunto de toda la información que se puede llegar a almacenar. Supongamos que tenemos una tabla de clientes (con atributos “D.N.I.” y “cuenta bancaria”): es obvio que sólo voy a almacenar “mis clientes” no “todos los posibles clientes”.

Supongamos también, que todos “mis clientes” tienen cuenta bancaria (es una restricción de integridad de mi base de datos). El problema al construir fórmulas en CPPO es que estamos haciendo deducciones sobre todo el universo de discurso. Luego no es lo mismo decir que “todos mis clientes tienen cuenta bancaria” que “todos los clientes tienen cuenta bancaria”. En este segundo caso, ¿cómo puedo asegurarlo si no lo “conozco”, no está almacenado en mi base de datos?

Es más, expresiones como $\neg \text{CLIENTE}(x, y)$ son correctas y tienen respuesta: “clientes que no están en la tabla CLIENTE” o “clientes desconocidos”. Pero claro, clientes desconocidos no son esas personas que aún no son mis clientes, sino todas las posibles combinaciones de “D.N.I.” y “cuenta bancaria”, tanto si se corresponden a personas reales como si no. No olvidemos que CLIENTE es un subconjunto del producto cartesiano de los dominios de sus atributos: todos los clientes son todas las tuplas de ese producto cartesiano.

Aparte de que, en bases de datos, no parece que haya ninguna justificación para preguntas de este tipo (la información que me interesa es la que está almacenada; ¿para qué necesito a los conductores que no “conozco”?), en el

peor de los casos, y siempre a nivel teórico, los dominios son potencialmente infinitos. Una pregunta así, y no olvidemos que nuestra intención final es trabajar con un ordenador y un sistema de gestión de bases de datos, llevaría a tener que apagar la máquina ya que nunca pararía de mostrar todas las infinitas tuplas de la relación.

Por todo ello, es necesario evitar tales expresiones, restringiendo el conjunto de fórmulas permitido a un subconjunto de F formado por aquellas fórmulas que se denominan **fórmulas seguras**. Podemos concretar el objetivo de usar sólo fórmulas seguras para interrogar a bases de datos relacionales en que pretendemos preguntar únicamente sobre información almacenada, evitar expresiones que estén afectadas por los datos que no están en mi base de datos.

Antes de definir la propiedad de seguridad de una fórmula es necesario introducir el concepto de dominio de una fórmula.

dominio de una fórmula

Dada una fórmula G que contiene los símbolos de predicado P_1, P_2, \dots, P_k y los símbolos de constante a_1, a_2, \dots, a_n , entonces el dominio de G , **Dom(G)**, es el siguiente conjunto:

$$\text{Dom}(G) = \{ K(a_1), K(a_2), \dots, K(a_n) \} \cup \text{Ext}(P_1) \cup \text{Ext}(P_2) \cup \text{Ext}(P_k)$$

El dominio de una fórmula, dicho en otras palabras, es el conjunto de valores que “maneja” la fórmula, las extensiones de los predicados y las constantes utilizados en la fórmula.

Ya que las relaciones asociadas a cada símbolo de predicado son conjuntos finitos, también $\text{Dom}(G)$ lo es.

fórmula segura

Una fórmula G es segura si, independientemente de la interpretación que se esté considerando, satisface las siguientes tres reglas:

- Si G tiene n variables libres (x_1, x_2, \dots, x_n) y (a_1, a_2, \dots, a_n) es una tupla tal que al sustituir cada x_i por a_i la fórmula que resulta es cierta, entonces $a_i \in \text{Dom}(G)$.
- Para cada subfórmula de G de la forma $\exists x F$, si a es un valor tal que al sustituir x por a en F la fórmula se hace cierta, entonces $a \in \text{Dom}(F)$.
- Para cada subfórmula de G de la forma $\forall x F$, si a es un valor tal que al sustituir x por a en F la fórmula se hace falsa³¹, entonces $a \in \text{Dom}(F)$.

Intuitivamente, una fórmula es segura si su evaluación depende sólo de las extensiones de las relaciones asociadas a los símbolos de predicado que en ella aparecen (y de las constantes utilizadas). Para entendernos, debemos asegurarnos de que los valores que hacen ciertas las fórmulas abiertas y las cerradas por cuantificadores existenciales, y los que hacen falsas las cerradas por cuantificadores universales, pertenecen a la base a la base de

³¹ ¿Porqué para el cuantificador universal sólo miramos los valores que hacen falsa la fórmula?
 $\forall x F$ es igual a $\neg \exists x \neg F$

datos. De una manera un poco menos formal, para que una fórmula G sea segura debe cumplirse que:

- Los valores de variables libres que no pertenecen al dominio de la fórmula no hacen cierta la fórmula.
- Los valores de variables ligadas por cuantificadores existenciales que no pertenecen al dominio de la fórmula no hacen cierta la subfórmula (el alcance del cuantificador).
- Los valores de variables ligadas por cuantificadores universales que no pertenecen al dominio de la fórmula no hacen cierta la subfórmula (el alcance del cuantificador).

Por ejemplo, $\text{CLIENTE}(x, y)$ es una fórmula segura puesto que sólo nos va a devolver valores almacenados en la tabla CLIENTE , pero $\neg\text{CLIENTE}(x, y)$ precisamente se hace cierta para aquellos valores que no están en la tabla, es una fórmula no segura.

Fórmulas seguras y no seguras es una clasificación en fórmulas aplicables a bases de datos relacionales y fórmulas que no se deben utilizar. La definición de fórmula segura es independiente de la interpretación que se utilice. Es evidente que una consulta que es segura en un estado de una base de datos, cuando cambie ese estado (cuando insertamos datos nuevos, cuando eliminamos, cuando modificamos) debe seguir siendo segura. Como veremos a continuación, sabremos que una fórmula es segura o no sin necesidad de utilizar valores concretos, una fórmula si es segura, siempre lo será, sea cual sea el contenido de mi base de datos.

G1	$\forall x (P(x) \rightarrow Q(x, y))$
	¿Es G1 una fórmula segura?

La forma de proceder pasa por verificar todas y cada una de las variables, teniendo en cuenta que si para una de ellas llegamos a la conclusión de que la fórmula no es segura, no hace falta continuar con el resto de las variables.

Para cada variable primero hemos de establecer la fórmula que vamos a examinar, y dependiendo de si es libre, ligada por un cuantificador existencial o ligada por un universal, comprobar que cumple con la definición de fórmula segura.

En este caso hay dos variables, x e y :

Empecemos por la variable y . Al ser libre, la fórmula a evaluar es toda G1.

Los valores de y que debemos comprobar son los que hacen cierta la fórmula, todo valor de y que haga cierta la fórmula debe pertenecer al dominio. G1 se hace cierta cuando son ciertos $P(x)$ y $Q(x, y)$ o cuando $P(x)$ es falso³².

Si suponemos que la extensión de $P(.)$ no tiene tuplas (representaría a una tabla vacía), la fórmula G1 será cierta valga lo que valga $Q(x, y)$. Concretamente, sería cierta para valores de y que no pertenecen al dominio de la fórmula, por lo que la fórmula no es segura³³. Ya no haría falta comprobar la variable x .

³² Véase la tabla de verdad del operador implicación.

³³ Nótese que la definición de fórmula segura es muy restrictiva: esta fórmula sólo da problemas cuando la extensión de $P(.)$ está vacía, es decir, para ciertos estados de la base de datos no

En realidad, lo que queremos constatar es que los valores que no pertenecen al dominio de la fórmula no la hacen cierta. Simplificando, y por entendernos, los valores no almacenados en nuestra base de datos no deberían aparecer en el resultado de la consulta.

G2	$\forall x (Q(x, y) \rightarrow P(x))$
----	--

La fórmula no es segura ya que cualquier valor de y que no pertenezca a su dominio hace falso $Q(x,y)$, y la fórmula sería cierta para todo x (de dentro de y de fuera del dominio).

G3	$\forall x \forall y (\text{CONDUCTOR}(x, y) \rightarrow \exists z \text{ CONDUCE}(x, z))$
----	--

variable x: ligada por un cuantificador universal. La fórmula a evaluar es toda G3.

Cualquier valor de x que no pertenezca al dominio de la fórmula hace falso $\text{CONDUCTOR}(x, y)$ y por tanto G3 cierta. Cómo estos valores no la hacen falsa, la fórmula puede ser segura.

variable y: la discusión es la misma que para la variable x .

variable z: ligada por un cuantificador existencial. La fórmula a evaluar es toda $\exists z \text{ CONDUCE}(x, z)$.

Cualquier valor de z que no pertenezca al dominio de la fórmula hace falso $\exists z \text{ CONDUCE}(x, z)$. Como los valores de "fuera" del dominio no la hacen cierta, la fórmula puede ser segura.

Las tres variables de la fórmula cumplen las reglas respectivas; por lo tanto, la fórmula es segura.

G4	$\forall x \forall y (\text{CONDUCTOR}(x, y) \wedge \exists z \text{ CONDUCE}(x, z))$
----	---

variable x : ligada por un cuantificador universal. La fórmula a evaluar es toda G4.

Cualquier valor de x que no pertenezca al dominio de la fórmula hace falso $\text{CONDUCTOR}(x, y)$ y por tanto G3 falsa. La fórmula no es segura.

G5 $\exists y \text{ CONDUCTOR}(x, y) \wedge \neg \exists z \text{ CONDUCE}(x, z)$

Esta fórmula abierta nos devolvería aquellos conductores que no conducen ningún vehículo.

variable x: es la variable libre. La fórmula a evaluar es toda G5.

Cualquier valor de x que no pertenezca al dominio de la fórmula hace falso $\text{CONDUCTOR}(x, y)$ y por tanto G4 falsa. Cómo estos valores no la hacen cierta, la fórmula puede ser segura.

variable y: ligada por un cuantificador existencial. La fórmula a evaluar es $\exists y \text{ CONDUCTOR}(x, y)$.

Cualquier valor de y que no pertenezca al dominio de la fórmula hace falso $\text{CONDUCTOR}(x, y)$. Como los valores que no pertenecen al dominio no la hacen cierta, la fórmula puede ser segura.

variable z: ligada por un cuantificador existencial. La fórmula a evaluar es $\exists z \text{ CONDUCE}(x, z)$. Nótese que la negación no está dentro del alcance del cuantificador

Cualquier valor de z que no pertenezca al dominio de la fórmula hace falso $\text{CONDUCTOR}(x, y)$. Como los valores que no pertenecen al dominio no la hacen cierta, la fórmula puede ser segura.

La fórmula es segura puesto que ninguna variable incumple la definición.

VI5. cálculo relacional

Ahora sabemos que una base de datos relacional se puede describir utilizando el CPPO. Eso quiere decir, también, que la podemos consultar usando fórmulas lógicas. No obstante, el lenguaje y la interpretación antes descritos pueden “refinarse” y hacerlos un poco más útil para la aplicación deseada, para las bases de datos relacionales.

Se definieron dos lenguajes de cálculo relacional: **orientado a tuplas** y **orientado a dominios**. La diferencia fundamental entre ambos reside en el tipo de variables que manejan. En el primero las variables se denominan *variables-tupla*, y como su nombre indica van a designar tuplas de relaciones. En el segundo lenguaje las variables se denominan *variables-dominio*, y van a tomar valores de los dominios asociados a los atributos de las relaciones.

Como ya se ha dicho, los dos lenguajes están basados en el Cálculo de Predicados de Primer Orden, por lo que utilizaremos las definiciones del tema anterior y las adaptaremos a los nuevos tipos de variables, sin profundizar más en dichas definiciones.

VI5.1. cálculo relacional orientado a tuplas

Es necesario realizar unas ligeras modificaciones a las definiciones introducidas en el tema anterior, para tener en cuenta el hecho de que las variables de este lenguaje son **variables-tupla** y, como ya se ha dicho, designan tuplas de relación.

Lo primero es definir cual es el dominio de una variable de este lenguaje. Anteriormente esto no ha sido necesario porque habíamos supuesto una lógica homogénea y, por tanto, este dominio coincidía con el conjunto de constantes del lenguaje. La forma de realizar esta asignación es *declarar* la variable. Cada variable-tupla está definida sobre una intensión de una relación R . Por ejemplo:

$$R(a_1 : dom_1, a_2 : dom_2, \dots, a_n : dom_n)$$

La declaración de la variable-tupla sería de la forma:

$$x : R$$

Así, declarando las variables sobre relaciones se especifica donde van a tomar sus valores. En nuestro ejemplo, la variable x tomará valores en el producto cartesiano de todos los dominios de R , $dom_1 \times dom_2 \times \dots \times dom_n$.

Aclarado este punto, pasemos a ver cuales van a ser los diferentes objetos de este lenguaje. Para empezar, los símbolos de este lenguaje serán los mismos que los vistos en el tema anterior, con la única diferencia de que las variables ahora son variables-tupla.

La construcción de fórmulas bien formadas precisa, pues, de los conceptos de término y fórmula atómica.

término

Decimos que son **términos** cualquiera de las dos expresiones siguientes:

- símbolos de constante
- $s.A$, donde s es una variable-tupla y A es el nombre de un atributo de la relación sobre la que se declaró la variable

fórmula atómica

Definimos **fórmula atómica** o **átomo** como cualquier expresión del tipo:

- $R(s)$, donde R es el nombre de una relación y s es el nombre de una variable-tupla que se declaró sobre la relación R o sobre otra relación compatible con R .
- $t_1 \otimes t_2$, donde t_1 y t_2 son dos términos y \otimes es un operador de comparación ($<$, $>$, $=$, \dots).

fórmulas abiertas

En el caso de las fórmulas abiertas añadimos una pequeña modificación de tal forma que podemos indicar qué atributos de las variables libres son los que nos interesa obtener. Es decir, podemos no necesitar las tuplas completas, si no sólo algunos de sus atributos, y lo especificamos así:

$$\{ t_1, t_2, \dots, t_n \mid F \}$$

Donde cada t_i es un término, y F una FBF en la que las únicas variables libres son las que aparecen en los t_i .

ejemplos

“Matrícula y marca de los vehículos”

x: VEHÍCULO
 $\{ x.matricula, x.marca \mid VEHÍCULO(x) \}$

“Matrícula y marca de los vehículos de 2003”

x: VEHÍCULO
 $\{ x.matricula, x.marca \mid VEHÍCULO(x) \wedge x.añoFab=2003 \}$

“Matrícula y marca de los vehículos de 2003 y número de los conductores que los conducen”

x: VEHÍCULO y: CONDUCE
 $\{ x.matricula, x.marca, y.conductor \mid VEHÍCULO(x) \wedge x.añoFab=2003 \wedge CONDUCE(y) \wedge y.vehiculo=x.matricula \}$

“Número de los conductores que conducen vehículos de 2003”

x: VEHÍCULO y: CONDUCE
 $\{ y.conductor \mid CONDUCE(y) \wedge \exists x (VEHÍCULO(x) \wedge y.vehiculo=x.matricula \wedge x.añoFab=2003) \}$

“Año de nacimiento de los conductores que conducen vehículos de 2003”

x: VEHÍCULO y: CONDUCE z: CONDUCTOR
 $\{ z.añoNac \mid CONDUCTOR(z) \wedge \exists y (CONDUCE(y) \wedge z.número=y.conductor \wedge \exists x (VEHÍCULO(x) \wedge y.vehiculo=x.matricula \wedge x.añoFab=2003)) \}$

O de una forma más concisa:

x: VEHÍCULO y: CONDUCE z: CONDUCTOR
 $\{ z.añoNac \mid CONDUCTOR(z) \wedge \exists y \exists x (CONDUCE(y) \wedge VEHÍCULO(x) \wedge z.número=y.conductor \wedge y.vehiculo=x.matricula \wedge x.añoFab=2003) \}$

“Todos los conductores conducen al menos un vehículo”

y: CONDUCE z: CONDUCTOR
 $\forall z (\text{CONDUCTOR}(z) \rightarrow \exists y (\text{CONDUCE}(y) \wedge$
 $z.\text{número}=y.\text{conductor}))$

VI5.2. cálculo relacional orientado a dominios

Las expresiones del cálculo relacional orientado a dominios se construyen siguiendo unos principios análogos al cálculo relacional basado en variables-tupla. Existen, sin embargo, un pequeño número de diferencias, siendo la principal precisamente el concepto de variable. Las variables de este lenguaje se denominan **variables-dominio**, y toman valores de los dominios asociados a los atributos de las relaciones.

Análogamente, la variable-dominio va a tomar valores en el dominio asociado a alguno de los atributos de una relación. Por ejemplo:

$$x : dom_k$$

De esta forma la variable x tomará valores en el dominio dom_k .

Aclarado este punto, veamos las adaptaciones del CPPO necesarias para definir el lenguaje.

fórmula atómica

Son fórmulas atómicas o átomos las expresiones de la forma:

- $R(a_1 : t_1, a_2 : t_2, \dots, a_n : t_n)$, donde R es una relación de grado igual a n o superior, a_1, a_2, \dots, a_n son atributos de R y t_1, t_2, \dots, t_n son términos. Los términos han de pertenecer a los dominios asociados a los atributos de la relación
- $t_1 \otimes t_2$, donde t_1 y t_2 son dos términos y \otimes es un operador de comparación ($<$, $>$, $=$, ...)

Al igual que en el cálculo relacional orientado a tuplas, añadimos una pequeña modificación en las fórmulas abiertas indicando qué variables son libres:

$$\{x_1, x_2, \dots, x_k \mid F\}$$

Las únicas ocurrencias de variable libres que aparecerán en F son x_1, x_2, \dots, x_k ; el resto de ocurrencias serán ligadas. La evaluación de esta expresión dará como resultado una relación k -aria, de manera que al sustituir cada una de las variables libres por los valores de los atributos de una de las tuplas de esta relación, da como resultado una fórmula cerrada que se evalúa a cierto.

ejemplos

“Matrícula y marca de los vehículos”

x: cadena(9) y: cadena(30)
 $\{ x, y \mid \text{VEHÍCULO}(\text{matrícula: } x, \text{marca: } y) \}$

“Matrícula y marca de los vehículos de 2003”

x: cadena(9) y: cadena(30)
 $\{ x, y \mid \text{VEHÍCULO}(\text{matrícula: } x, \text{marca: } y, \text{añoFab:2003}) \}$

“Matrícula y marca de los vehículos de 2003 y número de los conductores que los conducen”

x: cadena(9) y: cadena(30) z: cadena(2)
 $\{ x, y, z \mid \text{VEHÍCULO}(\text{matrícula: } x, \text{marca: } y, \text{añoFab:2003}) \wedge \text{CONDUCE}(\text{conductor: } z, \text{vehículo: } x) \}$

“Número de los conductores que conducen vehículos de 2003”

x: cadena(9) y: cadena(30) z: cadena(2)
 $\{ z \mid \exists x (\text{CONDUCE}(\text{conductor: } z, \text{vehículo: } x) \wedge \text{VEHÍCULO}(\text{matrícula: } x, \text{añoFab:2003})) \}$

“Año de nacimiento de los conductores que conducen vehículos de 2003”

x: cadena(9) y: cadena(30) z: cadena(2) t: entero
 $\{ t \mid \exists z (\text{CONDUCTOR}(\text{número: } z, \text{añoNac: } t) \wedge \exists x (\text{CONDUCE}(\text{conductor: } z, \text{vehículo: } x) \wedge \text{VEHÍCULO}(\text{matrícula: } x, \text{añoFab:2003}))) \}$

“Todos los conductores conducen al menos un vehículo”

x: cadena(9) z: cadena(2)
 $\forall z (\text{CONDUCTOR}(\text{número: } z) \rightarrow \exists x \text{ CONDUCE}(\text{conductor: } z, \text{vehículo: } x))$

BIBLIOGRAFÍA

[DATE2000]

J. D. Ullman

Principles of Database Systems

Computer Science Press, 1982

M. Jesús Castel, Faraón Llorens

Lógica de Primer Orden

Delobel, C.; Adiba, M.

Relational Database Systems.

Elsevier Science Publishers B.V., 1985.

Hamilton, A.G.

Lógica para Matemáticos

Ed. Paraninfo, 1981.

M.L. Brodie, J.L. Mylopoulos y J.W. Schmidt (eds),

Towards a Logical Reconstruction of Relational Database Theory.

“On Conceptual Modelling” Ed. Springer-Verlag, 1984.

VII ORGANIZACIÓN FÍSICA DE LAS BASES DE DATOS

1. INTRODUCCIÓN
2. CONCEPTOS BÁSICOS
3. FICHEROS
4. IMPLEMENTACION DE BASES DE DATOS RELACIONALES

Las bases de datos, independientemente del modelo de datos que se haya usado para su diseño, se almacenan físicamente como ficheros de registros, los cuales se almacenan normalmente en discos magnéticos.

Durante este tema presentaremos cómo se organiza la base de datos en su espacio de almacenamiento, y cuáles son las técnicas que se usan para acceder a estos datos de la forma más eficaz y rápida posible.

VII1. introducción

La colección de datos que constituye una base de datos debe estar almacenada en un **medio de almacenamiento** del ordenador. De esta manera, el SGBD será capaz de recuperar, actualizar y procesar dichos datos siempre que sea necesario. Los medios de almacenamiento en un ordenador forman una jerarquía llamada **jerarquía de almacenamiento** o **jerarquía de memoria** que contiene dos categorías principales:

- el almacenamiento primario: que incluye la memoria principal y la memoria caché, medios de almacenamiento muy rápidos aunque con capacidad limitada.
- el almacenamiento secundario: que incluye discos magnéticos, discos ópticos, cintas, etc., todos ellos medios de almacenamiento de grandes capacidades, aunque con un acceso mucho más lento.

Si bien el nivel de almacenamiento primario se caracteriza por ser más caro y rápido que el secundario, podemos decir que dentro de cada uno de estos niveles hay subniveles atendiendo también a estos criterios de coste y velocidad.

Así, en el nivel de almacenamiento primario destacamos los subniveles:

- memoria caché: es la capa más cara de la jerarquía y se trata de una memoria RAM estática, usada por la CPU para aumentar la velocidad de ejecución de las máquinas. Volátil.
- memoria DRAM (RAM dinámica) o memoria principal: usada por la CPU para mantener los programas y los datos. Más lenta que la RAM estática aunque más barata. Volátil.
- memoria flash: memoria no volátil situada en el último subnivel, dentro del nivel primario.

En el nivel de almacenamiento secundario se destaca:

- los discos magnéticos: siendo los más rápidos de este nivel.
- los discos CD-ROM (Compact Disk Read Only Memory) y su versión reescribible CD-RW: basados en tecnología óptica con lectura láser.
- las cintas: en el nivel más barato.

En cualquier caso, tanto el nivel de almacenamiento primario como el secundario se usan para almacenar bases de datos. La memoria principal, siempre que tenga capacidad suficiente, puede ser usada para albergar datos (o parte de ellos), manteniendo siempre una copia de respaldo en memoria secundaria para evitar los problemas que podría ocasionar la volatilidad de la primera.

Sin embargo, en la gran mayoría de las ocasiones, el tamaño de la base de datos impide su almacenamiento en una memoria primaria, y se mantiene en almacenamiento secundario utilizando la memoria primaria para cargar índices que aumentan la velocidad de acceso a los datos.

Dentro de la jerarquía del almacenamiento secundario, el principal soporte usado hasta el momento para el almacenamiento de las bases de datos es el disco magnético, y aunque puede que en el futuro los CD-ROM, DVD y otras tecnologías usurpen el espacio que actualmente ocupa el disco magnético, por el momento nos centraremos en este último como la principal fuente de

almacenamiento para bases de datos. Así en este tema nos centraremos en el estudio y comprensión de las propiedades y características de los discos magnéticos y la forma en que se pueden organizar los ficheros de datos en el disco con el fin de obtener bases de datos con un rendimiento adecuado.

Las técnicas usadas para el almacenamiento de grandes cantidades de información en disco deben ser conocidas tanto por los diseñadores como por los administradores de bases de datos, así como, evidentemente por los diseñadores e implementadores del SGBD.

VII2. conceptos básicos

Para comenzar con el estudio de las técnicas usadas en el almacenamiento de datos en los discos magnéticos, es necesario recordar algunas de las características principales que tienen estos medios. Comenzaremos por repasar el hardware de los discos.

EL DISCO MAGNÉTICO

El disco magnético sirve para almacenar grandes cantidades de datos. Aunque originalmente se construían de una sola cara, con el tiempo se construyeron discos de dos caras, capaces de albergar información en las dos superficies, e incluso formando *paquetes de discos* donde los datos se organizan en varias superficies. La información almacenada en un disco se encuentra en círculos concéntricos de pequeña anchura que se denomina **pista** y que varían de diámetro conforme se alejan o se acercan al eje del disco. En los paquetes de discos, todas las pistas de cada disco que tienen el mismo diámetro se organizan formando un **cilindro**. El concepto de *cilindro* es importante para el almacenamiento de una base de datos, ya que los datos que se encuentran en mismo cilindro se pueden leer con mayor rapidez que si se encuentran en varios cilindros, debido a que las cabezas lectoras del disco son capaces de leer simultáneamente los datos de todas las pistas de un mismo cilindro.

Puesto que un pista puede contener gran cantidad de información, ésta se puede organizar a su vez en unidades menores llamadas **sectores**. Finalmente, en la fase de formateo del disco, el sistema operativo realiza la división de la pista en unidades de igual tamaño llamadas **bloques de disco** o **páginas**.

El mecanismo hardware que se encarga de escribir o leer los bloques es la **cabeza de lectura/escritura** del disco que se maneja gracias a un brazo mecánico. Esta estructura junto con el disco, y el **motor** del disco, forman la **unidad de disco**.

Además, el **controlador de disco** (situado generalmente en la unidad de disco) controla y hace de interfaz entre ésta y el ordenador. El controlador acepta instrucciones de E/S de alto nivel y realiza las acciones oportunas para colocar el brazo y dar la orden de lectura o escritura. Una vez conocida la dirección del bloque a la que se desea acceder, el controlador de disco necesita un tiempo (entre 8 y 14 mseg) para colocar la cabeza sobre la pista correcta. Este tiempo se conoce como **tiempo de búsqueda**. Una vez situada la cabeza sobre la pista, es necesario que el disco gire hasta situar el inicio del bloque bajo la cabeza generándose otro retardo llamado **retardo rotacional** o **latencia**. Por último deberá transcurrir un tiempo adicional para la transferencia de los datos, conocido como **tiempo de transferencia de**

bloque. Para mejorar la eficiencia en la transferencia de bloques, se suelen transferir varios bloques consecutivos de la misma pista, e incluso del mismo cilindro.

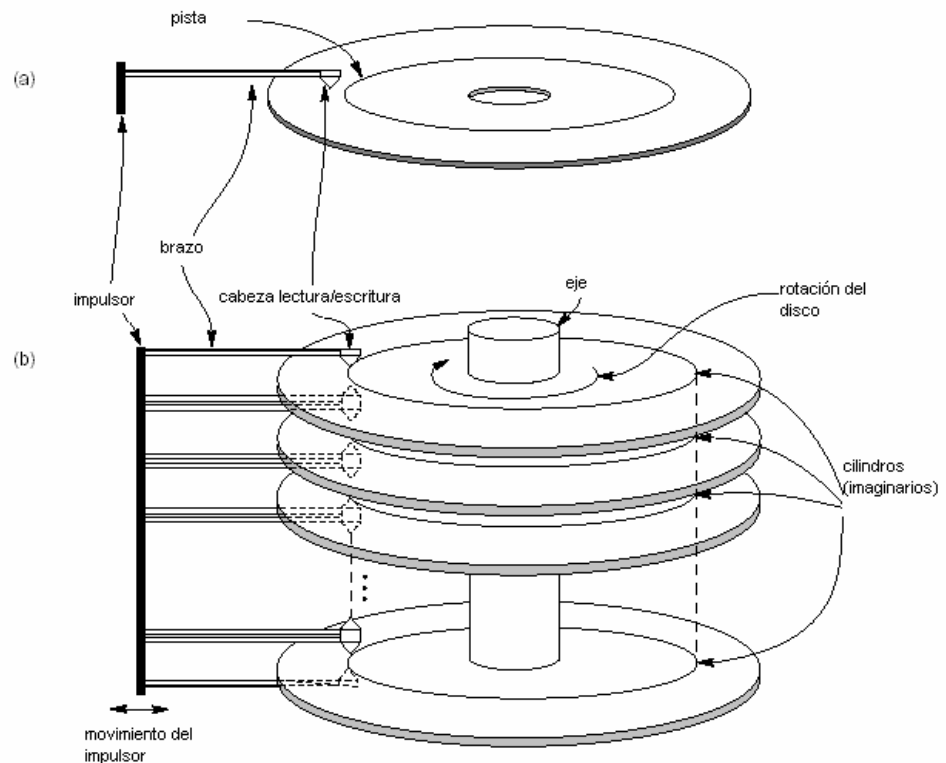


Figura 1. Partes principales del disco magnético

Resumiendo, el tiempo necesario para localizar y transferir un bloque es la suma de:

tiempo de búsqueda + latencia + tiempo de transferencia de bloque

Este tiempo suele variar entre 12 y 60 mseg dependiendo del fabricante. Es por tanto crucial la elección de un buen disco duro como uno de los factores de éxito en la implantación de una base de datos.

TECNOLOGÍA RAID

Pese a que la tecnología en el desarrollo de dispositivos de almacenamiento secundario (especialmente discos duros) ha evolucionado espectacularmente, las prestaciones que se pueden alcanzar con una única unidad de disco resultan insuficientes para el almacenamiento eficiente de grandes cantidades de datos. Es por este motivo por el que se ha desarrollado la tecnología conocida con el nombre de **RAID** (*Redundant Array of Inexpensive Disk*).

La idea original de RAID fue la de igualar los rendimientos de los discos a los de los procesadores y memorias principales. Mientras la capacidad de la RAM se cuadruplica cada dos o tres años, los *tiempos de acceso* a disco apenas mejoran un 10% al año, y los *tiempos de transferencia* menos de un 20%.

Como solución RAID plantea crear un array de pequeños discos independientes que actúan como un único disco lógico de alto rendimiento. Además, se utiliza una estrategia de almacenamiento conocida como **data striping** (*franqueo de datos*) que se basa en el paralelismo. Según esta técnica un mismo fichero se distribuye entre varios discos para que su lectura completa se realice simultáneamente desde todos los discos. El acceso en paralelo a todos los discos es mucho más rápido que un acceso secuencial a

uno de ellos. De ahí que las lecturas y escrituras de los datos resulten mucho más rápidas.

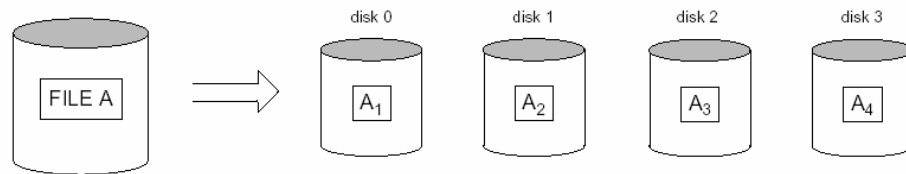


Figura 2. **Data striping** en RAID

RAID tiene como objetivo principal mejorar el *rendimiento* de los discos magnéticos, aunque sin olvidar un aspecto fundamental que es la *fiabilidad*. Para un array de n discos, la posibilidad de fallo es n veces mayor que para un único disco. El mantenimiento de una única copia de los datos en este tipo de estructura conllevaría una importantísima pérdida de fiabilidad. La solución es introducir *redundancia* de datos.

Una técnica que permite la redundancia es lo que se conoce como **mirroring** o **shadowing**, también conocido en español como **espejo**. Según esta técnica, los datos se escriben físicamente en dos discos diferentes de forma redundante. Al leer los datos, éstos se recuperan del disco que tenga menos carga de peticiones, o que sea más rápido. Si un disco falla, se usa el otro hasta que se repare el primero.

ORGANIZACIÓN Y NIVELES RAID

Hay diferentes organizaciones RAID atendiendo a la forma de hacer el franqueo de los datos y a la forma de acceder a los datos redundantes. Estas organizaciones se jerarquizan en 7 niveles: desde RAID nivel 0 hasta RAID nivel 6.

- **RAID 0** no tiene redundancia de datos. De todas las organizaciones RAID, ésta tiene el mejor rendimiento en escritura ya que no tiene que duplicar modificaciones. Sin embargo, el rendimiento en lectura es menor.
- **RAID 1** ya plantea la redundancia de datos mediante discos espejo. El rendimiento en lectura es mayor que en RAID 0 debido a que la petición de lectura se aplica al disco que sea más rápido en cada momento.
- **RAID 2** optimiza el almacenamiento de información redundante al almacenar los códigos de detección de error (bits de paridad) una única vez (no una para cada copia como ocurre en RAID 1) comunes. De esta manera, un ejemplo concreto de 4 discos, sólo necesitaría 3 discos espejo (en lugar de los 4 que necesita RAID 1).
- **RAID 3 a RAID 6** optimizan RAID 2 con diferentes estrategias para detección de errores que minimizan el número de discos necesarios aumentando el *rendimiento* del sistema sin perder *fiabilidad*.
-
-

Como conclusión para este apartado podríamos concluir que el disco magnético es una de las fuentes de almacenamiento secundario más extendidas para el almacenamiento de bases de datos. Sin embargo, la

evolución tecnológica del disco magnético no ha sido tan espectacular como lo ha sido la evolución tecnológica de la memoria RAM o de los procesadores. Por este motivo ha sido necesario la introducción de técnicas de almacenamiento redundante que han conseguido mejorar los rendimientos de estos dispositivos. En concreto, la organización RAID es una de las más extendidas en los servidores que tienen como principal misión el almacenamiento de grandes bases de datos.

Una vez sentados los fundamentos del “hardware” que se usa para la implementación de las bases de datos, introduciremos algunos conceptos sobre cómo se almacena físicamente un fichero.

VII3. ficheros

El fichero es la organización básica de almacenamiento en disco, y por tanto es la organización empleada para la implementación de una base de datos. Para introducir el concepto de fichero es necesario introducir previamente el concepto de registro.

VII3.1. registro

Un **registro** es una colección de valores o elementos de datos relacionados donde cada valor está formado por uno o más bytes y corresponde a un determinado **campo** del registro.

Por regla general, cada registro representa a una entidad, y cada valor de campo en ese registro representa a un atributo de esa entidad. La lista de nombres asociados a cada campo y los tipos de datos que se almacenan en ellos constituyen lo que se llama el **formato del registro**.

Actualmente las bases de datos también incluyen grandes objetos que no pueden ser estructurados en campos como imágenes, secuencias de video, secuencias de audio, o incluso texto libre. Estos objetos se conocen como **BLOB** (*Binary Large Objects*) y por lo general se almacenan aparte de su registro, en un área de bloques de disco, y se incluye un puntero desde el registro al BLOB.

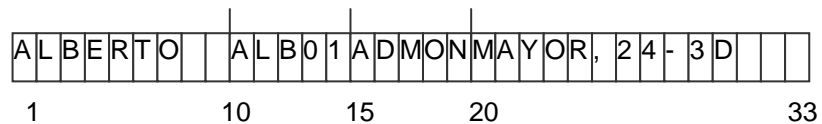
VII3.2. fichero

Un **fichero** se define como una secuencia de registros.

Si todos los registros de un fichero tienen la misma longitud se les conoce como **registros de longitud fija**. Sin embargo, si cada registro tiene una longitud diferente hablaremos de **registros de longitud variable**. Esto nos proporciona diferentes organizaciones para el almacenamiento de los registros:

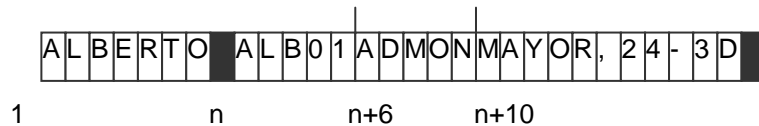
- a) *almacenamiento de registros de longitud fija*: el registro tiene una longitud conocida y cada campo ocupa una posición conocida dentro del registro. Para acceder a alguno de ellos únicamente es



necesario conocer su posición de inicio y de fin del campo en el registro.



<i>Campo</i>	<i>Posición inicio</i>	<i>Posición fin</i>
NOMBRE	1	9
CÓDIGO	10	14
DEPARTAMENTO	15	19
DIRECCIÓN	20	33

- b) *almacenamiento de registros de longitud variable* formados por *campos de longitud fija* y *campos de longitud variable*: se utilizan caracteres especiales para delimitar el final de un campo de longitud variable, y se usan las longitudes conocidas de los campos de longitud fija para conocer sus límites.



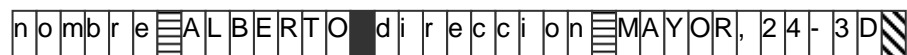
<i>Campo</i>	<i>Posición inicio</i>	<i>Posición fin</i>
NOMBRE	1	n-1
Carácter fin 	N	n
CÓDIGO	n+1	n+5
DEPARTAMENTO	n+6	n+10
DIRECCIÓN	n+11	m-1
Carácter fin 	M	m

- c) *almacenamiento de registros de longitud variable* formados por *campos de longitud variable* y algunos de ellos opcionales: La existencia de campos optativos obliga a almacenar junto con cada campo el nombre del campo. Así se introducen tres tipos de caracteres especiales separadores: separador del nombre del campo, indicador de fin de campo e indicador de fin de registro.

Carácter separador de nombre de campo

Carácter indicador fin de campo

Carácter indicador fin de registro



Los ficheros constituyen la unidad básica en el almacenamiento físico de las bases de datos. A nivel práctico, podemos considerar que cada tabla de un esquema lógico definida sobre el modelo relacional se convertirá en un fichero (o conjunto de ficheros) que será manejado por el sistema operativo desde el SGBD.

VII3.3. operaciones con ficheros

Las operaciones que se realizan con los ficheros se pueden clasificar en **operaciones de recuperación** y **operaciones de actualización**, encargadas de leer y de escribir datos respectivamente. En cualquiera de los dos casos es necesario realizar una selección previa del registro con el que se quiere trabajar mediante una **condición de selección** o **filtro** que especifica los criterios que debe satisfacer el registro o registros con los que se va a operar.

Quien se encarga de acceder físicamente a los ficheros, y por tanto, quien ejecuta las operaciones de recuperación y actualización de ficheros es el *sistema operativo*. Sin embargo, la condición de búsqueda debe establecerse desde fuera del sistema operativo. Cuando se trabaja con una base de datos, el encargado de establecer esa condición de selección es el propio *SGBD*. Una condición de selección simple puede ser una comparación de igualdad de un valor para un campo, por ejemplo, `dni='22233334'`.

Sin embargo, en bases de datos es frecuente realizar selecciones con múltiples condiciones (ya conocemos la complejidad selectiva que puede llegar a tener una sentencia `SELECT` de `SQL`). En este caso, el *SGBD* debe descomponer la selección en sus múltiples condiciones para acceder al registro adecuado. Por ejemplo, si un usuario solicita “empresas de Murcia que tienen más de tres empleados”, su pregunta la traducirá a una consulta `SQL` como:

```
SELECT * FROM empresas WHERE ciudad='murcia' AND empleados>3
```

Esta consulta se introduce en el *SGBD*. Sin embargo, el sistema operativo no podrá interpretar `SQL`. Por este motivo, el *SGBD* tendrá que traducir esta a consulta una *operación de recuperación* que deberá obtener el registro (o registros) que cumpla las dos siguientes *condiciones de selección*:

- a) *campo ciudad del fichero empresas = 'murcia'*
- b) *campo empleados del fichero empresas > 3*

Cuando hay varios registros que satisfacen la condición de selección, inicialmente el sistema operativo únicamente le devuelve al *SGBD* el *primer* registro que cumple con esa selección, aunque le proporciona un puntero al *siguiente* para que el *SGBD* pueda ir recorriendo la cadena de registros seleccionada.

Para realizar esta búsqueda, la mayoría de los *SGBD* disponen de las siguientes instrucciones implementadas a través del sistema operativo:

Abrir: prepara un fichero para su lectura o escritura. Si se puede, carga en memoria principal los bloques del disco que contienen al fichero, si no puede (por falta de memoria) cargará una parte del fichero (al menos la cabecera). Además sitúa el puntero al principio del fichero para iniciar la operación.

Volver al principio: pone el puntero de un fichero abierto en el principio.

Buscar: localiza el primer registro que satisface una condición de búsqueda, y transfiere ese bloque de disco a memoria principal (si no está ya).

Leer: copia el contenido del registro en una variable del programa.

Buscar siguiente: busca el siguiente registro que satisface la condición de búsqueda especificada en la última operación *buscar* efectuada. Carga el bloque en memoria (si no lo tiene ya).

Eliminar: elimina el registro actual del fichero y actualiza el fichero en el disco.

Modificar: modifica valores de algunos campos del registro y actualiza el fichero en el disco.

Insertar: inserta un nuevo registro en el fichero localizando el bloque donde se debe insertar, transfiriéndolo a memoria principal, insertándolo, y volviendo a escribir el bloque en el disco.

Cerrar: finaliza el acceso al fichero liberando la memoria principal ocupada por él.

VII3.4. organizaciones de ficheros usadas para bases de datos

En el almacenamiento de los datos, la organización interna del fichero tiene también su importancia. La organización de los ficheros debe facilitar la localización de los registros, especialmente en las búsquedas que se realizan sobre determinados campos. Por ejemplo, si deseo buscar un cliente por su dni, el fichero que contiene a los clientes de la base de datos debería estar ordenado físicamente por este valor o definir un índice sobre este campo. Sin embargo, si sobre el mismo fichero deseo buscar los clientes que residen en una misma ciudad, se necesitaría agrupar físicamente los datos por el campo ciudad para obtener una respuesta eficiente. A priori, esta organización no es compatible con la anterior, sin embargo, vamos a estudiar a continuación diversos tipos de organización que pueden ser útiles para hacer compatibles dichas organizaciones.

Ficheros de montículo (heap)

Los **ficheros de montículo** o **heap** constituyen el tipo más simple de organización. Los registros se colocan en el mismo orden en el que se insertan. Esta organización simplifica algunas de las operaciones aunque otras se hacen más complejas.

VII3.4.1.1. Inserción

Cada registro nuevo se inserta al final del fichero de la siguiente forma:

- a) se consulta en la cabecera del fichero la dirección física del último bloque de disco para ese fichero,
- b) se copia el bloque a memoria principal,
- c) se añade el nuevo registro,
- d) se reescribe el bloque en el disco,
- e) se actualiza la cabecera del fichero con la dirección del último bloque de disco (si hace falta).

La operación se realiza de forma sencilla y eficiente.

VII3.4.1.2. Búsqueda

Por el contrario, en este tipo de organización la búsqueda de un determinado registro de acuerdo a una *condición de búsqueda* se realiza comprobando registro a registro, lo que supone físicamente una lectura bloque a bloque del disco. En un fichero de n bloques, se requiere realizar $n/2$ comprobaciones por término medio.

VII3.4.1.3. Borrado

Para eliminar un registro hay que:

- a) buscar el bloque de disco donde se encuentra el registro,
- b) copiar el bloque a memoria principal,
- c) eliminar el registro,
- d) rescribir el bloque en disco.

Solamente el primero de los pasos (la búsqueda) ya tiene complejidad suficiente para hacer la operación poco eficiente. Además, será necesario pasar un proceso periódico de reorganización del fichero ya que los huecos que quedan libres por las eliminaciones nunca se rellenan por lo que se desperdicia espacio.

VII3.4.1.4. Actualización

La actualización de un registro se realiza según los siguientes pasos:

- a) buscar el bloque de disco donde se encuentra el registro,
- b) copiar el bloque a memoria principal,
- c) modificar el registro,
- d) rescribir el bloque en disco.

El proceso de nuevo se complica en el primer paso, pero también tiene una complicación añadida cuando se usan *registros de longitud variable*. Si un registro se modifica ampliando su longitud puede que ya no tenga espacio suficiente en su bloque para poder albergarlo. En ese caso, la operación de actualización se convertiría en dos operaciones: *borrado* del registro antiguo + *inserción* de uno nuevo al final del fichero.

VII3.4.1.5. Lectura

La lectura del fichero según un determinado criterio (por ejemplo, por orden alfabético del nombre, o por orden de dni) requiere hacer una copia del fichero según este orden, o bien, si el tamaño es excesivo utilizar ficheros de *índices* externos.

Ficheros ordenados (o secuenciales)

Los **ficheros ordenados** se caracterizan por tener un orden físico según los valores de uno de sus campos, llamado **campo de ordenación**. Si el campo de ordenación tiene además propiedades de identificador (es decir, siempre tiene valor y no se repite) se conoce como **clave de ordenación** del fichero.

	NOMBRE	DNI	CUMPLEAÑOS	EDAD	SALARIO	SEXO
bloque 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
bloque 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
bloque 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
bloque 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
bloque 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
bloque 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
⋮						
bloque n-1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
bloque n	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

Figura 3. Fichero secuencial

Los *ficheros ordenados* plantean ciertas ventajas sobre *los ficheros de montículo*:

- la lectura de registros en el orden físico almacenado es extraordinariamente eficiente,
- los accesos a registros en orden consecutivo no requiere cambiar de bloque en la mayoría de los casos, por lo que no se efectúan muchos accesos a disco,
- las condiciones de búsqueda que afectan a valores de un campo *clave de ordenación* son muy eficientes cuando se emplean

estrategias de búsquedas binarias. Una búsqueda binaria accede normalmente a $\log_2(n)$ bloques, encuentre o no el registro que busca entre n bloques.

Sin embargo, sigue planteando algunas desventajas. Analicemos una a una las diferentes operaciones.

VII3.4.1.6. Inserción

La inserción es una operación costosa ya que se debe conservar el orden físico de los registros. Se debe:

- a) encontrar la posición para el nuevo registro en el fichero según el campo de ordenación,
- b) abrir espacio en el bloque correspondiente, desplazando el resto de los registros.

Esta última fase de la operación es especialmente dura, ya que significa que se tiene que leer y volver a escribir una media de $n/2$ bloques para un fichero de n bloques.

VII3.4.1.7. Búsqueda

La búsqueda por un campo de ordenación ya sabemos que es extremadamente eficiente, sin embargo la búsqueda por cualquier otro tipo de campo se convierte en una tarea de búsqueda lineal como lo era en el *fichero de montículo*.

VII3.4.1.8. Borrado

El borrado se realiza exactamente igual que en el *fichero de montículo*. Sigue siendo una operación costosa. La única diferencia estriba en que la búsqueda del registro a borrar será más eficiente si se busca por el *campo de ordenación*.

VII3.4.1.9. Actualización

La actualización se realiza de la misma manera que en los ficheros de montículo. Sólo un par de diferencias:

- a) el proceso de búsqueda es más eficiente,
- b) si lo que se ha modificado es el valor del campo de ordenación, entonces se convierte en una operación de *borrado + inserción* del registro en su nueva ubicación, con la complejidad que ello conlleva.

VII3.4.1.10. Lectura

La lectura secuencial del fichero ordenado es muy eficiente siempre que se siga el campo de ordenación. En cualquier otro caso es igual que la lectura de un *fichero de montículo*.

Ficheros de direccionamiento calculado (hashing)

Los **ficheros de direccionamiento calculado**, también conocidos como *dispersos* o **hashing**, proporcionan un acceso muy rápido a los registros bajo ciertas condiciones de búsqueda. La condición de búsqueda se realiza sobre un único campo del registro conocido como **campo de direccionamiento calculado**. Este campo también se llama **clave de direccionamiento calculado**. El funcionamiento de esta organización se basa en establecer una función h llamada **función de direccionamiento calculado**, que una vez aplicada sobre el valor del campo de direccionamiento calculado de un registro produciría la dirección del bloque de disco en el que se almacena el registro. Una vez obtenido el bloque se copia a memoria principal y se realiza la búsqueda del registro dentro del bloque.

Una función de direccionamiento calculado muy común para campos de direccionamiento calculado con valor numérico es:

$$h(K) = K \bmod M$$

donde K es el valor del campo, y M es el número total de bloques de disco donde se almacenan los registros.

Así por ejemplo, si se van a usar 20 bloques, y el campo de direccionamiento calculado DNI tiene valor = 21987239, la función h devuelve

$$h(21987239) = 21987239 \bmod 20 = 19$$

Esto indicaría que el registro correspondiente al valor de dni 21987239 se almacena en el bloque número 19. A partir de ahí ya sólo resta conocer la dirección física de ese bloque y copiarlo a memoria principal.

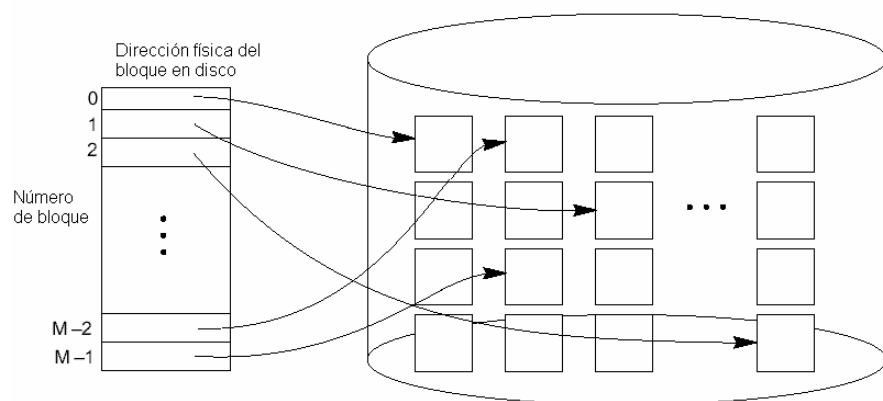


Figura 4. Localización física de los bloques calculados en disco

Los ficheros basados en el direccionamiento calculado plantean muchas ventajas respecto a los de montículo o a los ordenados.

- a) Las búsquedas se realizan de manera rápida en cualquiera de los campos de búsqueda.

- b) Los datos se insertan en el disco de forma dispersa, lo cual permite fácilmente las operaciones de borrado, inserción y actualización, sin necesidad de reorganizar el fichero.

VII3.5. estructuras de índices para ficheros

En el apartado anterior hemos podido comprobar que hay diferentes organizaciones primarias para almacenar la información: organización de montículo (no ordenada), organización ordenada, y organización de direccionamiento calculado (dispersa o *hashing*).

Sin embargo, estas estructuras pueden no ser suficientes para las necesidades de una base de datos. En muchos casos es necesario hacer uso de unas estructuras de acceso auxiliares llamadas **índices** que se emplean para aumentar la velocidad de recuperación de registros bajo ciertas condiciones de búsqueda.

Los *índices* proporcionan caminos alternativos para acceder a los registros sin que se vea afectada la posición física que los registros ocupan dentro del disco.

Los campos de un registro que se utilizan para construir un índice se denominan **campos de indexación**.

Cualquier campo del registro puede ser un *campo de indexación*, y un mismo fichero puede incluir *múltiples índices* (es decir más de un campo de indexación por registro).

El funcionamiento del *índice* es el siguiente: cuando se produce una condición de búsqueda que afecta a un campo de indexación, en lugar de acceder directamente al *fichero de datos* para iniciar la búsqueda (según la organización propia del *fichero de datos*: *montículo*, *ordenada* o *direccionamiento calculado*), se accede previamente a un fichero (llamado **índice** o **fichero de índice**) donde se encuentran los valores del *campo de indexación* correspondiente ordenados según la organización propia del fichero de *índice*, junto con un puntero que señala físicamente al bloque de disco que alberga a ese registro en el *fichero de datos*. El acceso al registro ya es una tarea trivial según hemos visto.

El uso de índices permite:

- a) realizar búsquedas eficientes en ficheros de datos organizados en *montículo* (sin índices recordemos que se trataba de una lenta búsqueda bloque a bloque)
- b) acceder de forma eficiente mediante condiciones de búsqueda que afectan a cualquier campo en un fichero ordenado (recordemos que esta organización sólo se accedía de forma eficiente cuando la búsqueda se realizaba sobre el *campo de ordenación*, y éste era único para un determinado fichero).
- c) acceder de forma eficiente mediante condiciones de búsqueda que afectan a cualquier campo en un fichero con direccionamiento calculado (éste accede de forma eficiente cuando la búsqueda se refiere al *campo de direccionamiento calculado*, pero las búsquedas por otros campos se tienen que realizar secuencialmente).

La implementación de los ficheros de *índice* se realiza mediante organización ordenada, o mediante el empleo de estructuras de datos en árbol (árboles binarios) que optimizan las búsquedas.

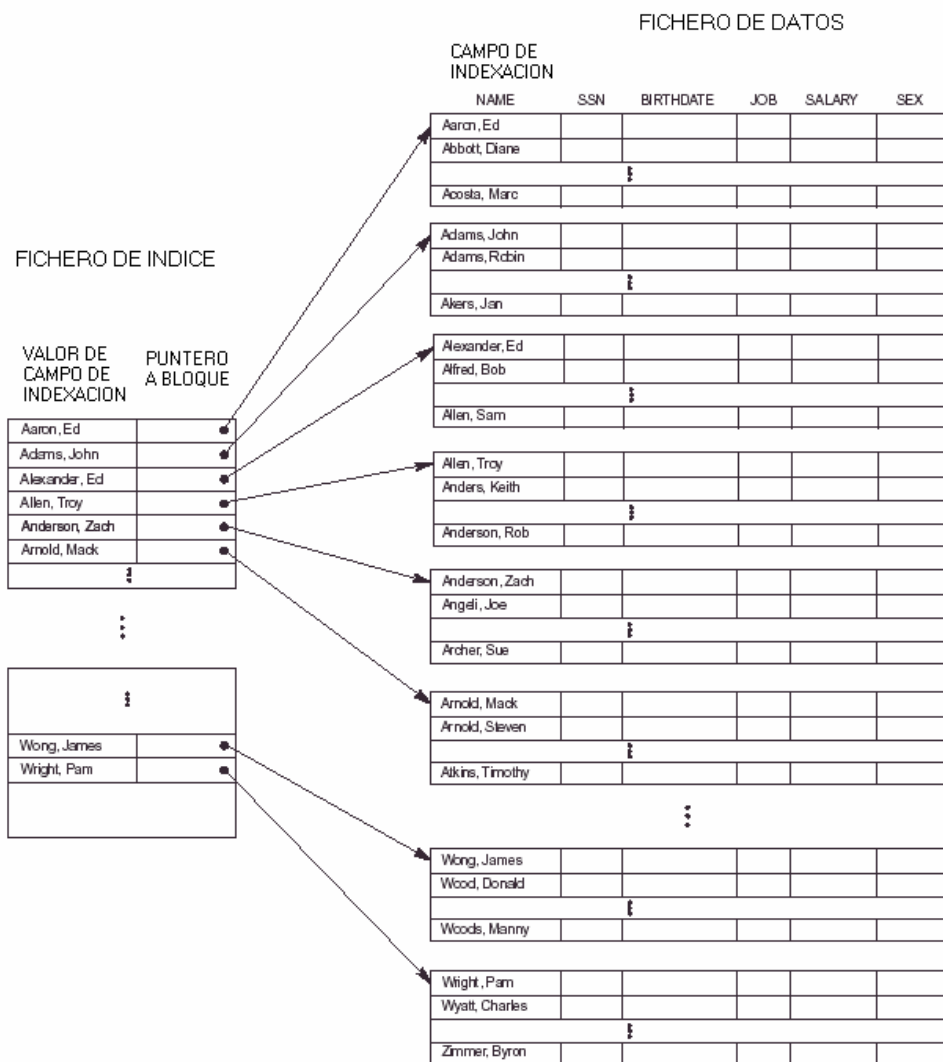


Figura 5. Funcionamiento del índice sobre un fichero con organización ordenada

VII4. implementación de bases de datos relacionales

Las bases de datos relacionales, como cualquier otra base de datos, necesita una estructura física de almacenamiento basada en el uso de ficheros. Aunque los *ficheros basados en direccionamiento calculado* tienen un acceso muy potente, su implementación es compleja y no siempre es rentable su coste.

Para la implementación de tablas que tienen mucha volatilidad, es decir, que crecen y decrecen rápidamente, puede ser especialmente aconsejable el uso de *ficheros de direccionamiento calculado*. Sin embargo, para las tablas que tienen poca movilidad puede ser suficiente con *ficheros de montículo* o *ficheros ordenados*.

El apoyo de ficheros de *índice* que permiten localizar los registros rápidamente siguiendo cualquier condición de búsqueda, se hace

especialmente necesario cuando se implementan las bases de datos relacionales. Pensemos que la instrucción *SELECT* de *SQL* permite realizar búsquedas que afectan a cualquier columna de la tabla, por lo que se traduce físicamente en condiciones de búsqueda que afectan a cualquier campo del registro.

En general, el uso de uno u otro tipo de organización de ficheros depende de que el SGBD permita su uso (no todos los SGBD implementan *ficheros de direccionamiento calculado*), y del criterio que tome el administrador de la base de datos acerca de la implementación concreta de cada tabla. El uso o no de ficheros de índice que mejoren la velocidad de búsqueda de registros queda también a juicio del propio administrador.

Para facilitar la tarea del administrador, el SGBD suele mostrar estadísticas de uso y tiempos de acceso a cada uno de los ficheros que implementan la base de datos. Con estos datos, el administrador se encargará de hacer los ajustes correspondientes (operación conocida como **tunning**) tomando las oportunas decisiones respecto a la organización de los ficheros.

BIBLIOGRAFÍA

[ELMASRI2002]

VIII SISTEMAS DE GESTIÓN DE BASES DE DATOS

Por los problemas que planteaban los antiguos sistemas de información mecanizados por ordenador basados en archivo convencional, se intentó desarrollar un nuevo enfoque a la gestión de los datos que garantizara lo máximo posible la independencia de las aplicaciones frente a los datos, la integridad y la seguridad de los mismos.

Apareció el concepto de base de datos y el software que lo manejaba: los sistemas de gestión de bases de datos. Se verá en este tema cual es la estructura de un SGBD y la funcionalidad que ofrece, y qué los hace una opción mucho más apetecible a la hora de manejar grandes volúmenes de datos. En general, se ofrecerá una visión introductoria de qué es un SGBD, cómo funciona y para qué sirve.

La parte final aborda, superficialmente, los problemas de independencia, integridad y seguridad de los datos.

Dada la creciente importancia del concepto, se introduce la arquitectura cliente-servidor como una visión distinta del acceso a los SGBD.

VIII1. técnicas de base de datos

En un principio, los problemas de los sistemas basados en fichero convencional se solventaban fácilmente, o ni siquiera existían, dada la poca envergadura de los sistemas de información que se mecanizaban. Pero, a medida que los avances en un campo tan inestable como éste fueron aumentando las prestaciones y capacidades de las herramientas de proceso de datos, los programas se hacían más extensos y complejos, manejaban una mucho mayor cantidad de datos, y necesitaban que cada vez más usuarios compartieran los mismos datos.

Lo que antes era un mínimo contratiempo se convirtió en un interminable quebradero de cabeza: mala o nula planificación y la pobreza de los modelos de datos utilizados provocaron que cada vez fuera más difícil el mantenimiento, a unos mínimos niveles de rendimiento y fiabilidad, de las aplicaciones existentes.

Las **técnicas de Bases de Datos** surgen al intentar superar la situación descrita. El objetivo prioritario es *unificar toda la información del sistema para evitar redundancias, sin perder las distintas perspectivas que de la misma tienen los usuarios*. Los datos se organizan y se mantienen en un conjunto estructurado que no está diseñado para una aplicación concreta, sino que tiende a satisfacer las necesidades de información de toda la organización. Es decir, se separa lo que es el sistema de información en sí, de lo que son las aplicaciones que pretenden explotar los datos almacenados en él. No obstante, la consecución de este objetivo crea nuevos problemas y agrava otros ya existentes: independencia, concurrencia, privacidad, etc.

En un *sistema de bases de datos* podemos identificar someramente los siguientes componentes:

- Datos
- Hardware
- Software: programas de gestión de bases de datos y de aplicación, utilidades, etc.
- Usuarios: programadores de aplicación, usuarios finales, administrador de la Base de Datos (DBA).

Entre las herramientas de software desarrolladas para crear y gestionar una base de datos, sin duda, la más importante es el **Sistema de Gestión de Bases de Datos** (SGBD). Este “programa” es el encargado de poner a disposición de los distintos usuarios las técnicas de bases de datos.

- *Descripción centralizada de los datos*: si todos se encuentran en un mismo lugar al que acceden las aplicaciones, ya no es necesaria la replicación.
- *Posibilidad de definir vistas parciales de dichos datos para los diferentes usuarios*: estas vistas parciales estarán compuestas por los datos y relaciones de interés para una aplicación o

conjunto de aplicaciones en concreto. Pensemos que un empleado de Contabilidad no tiene porqué conocer el estado civil de otro empleado pero este dato si le puede interesar a otro de Personal.

Por otra parte, los objetivos de un SGBD son:

- **Independencia** de datos: los programas de aplicación deben verse afectados lo menos posible por cambios efectuados en datos que no usan.
- **Integridad** de los datos: la información almacenada en la BD debe cumplir ciertos requisitos de calidad; para ello hace falta, en el momento de introducirse los valores de los datos, que éstos se almacenen debidamente, y que posteriormente no se deterioren (en sí mismos, en sus interrelaciones y en su accesibilidad).
- **Seguridad** de los datos: a la información almacenada en la BD sólo pueden acceder las personas autorizadas y de la forma autorizada.

Podemos dar una primera definición de Base de Datos:

“Colección de datos estructurados según un modelo que refleje las relaciones y restricciones existentes en el mundo real. Los datos han de ser compartidos por diferentes usuarios y aplicaciones, y deben mantenerse independientes de éstas; su definición y descripción han de ser únicas, estando almacenadas junto con los mismos. Los tratamientos habrán de conservar la integridad y seguridad de los datos.”

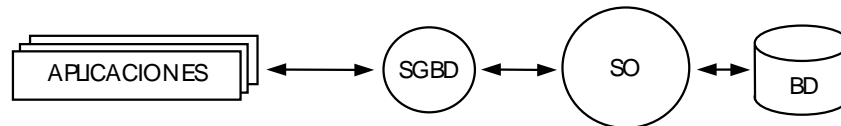
Podemos decir, de una manera muy general, que la *base de datos* es el conjunto de información, y el *sistema de gestión de bases de datos* el encargado de mantener las virtudes que se proponen en la definición.

VIII2. arquitectura de un sistema de gestión de bases de datos

Un SGBD no es, ni más ni menos, que un programa de ordenador que maneja Bases de Datos. Como tal, tiene unas características comunes que se encuentran, implementadas de una forma o de otra, en todos los productos comerciales disponibles en el mercado. Nuestra pretensión es dar una visión general de esas características sin centrarnos en producto alguno.

El hecho fundamental que justifica la utilización de un SGBD es que, si antes los programas llamaban directamente al sistema operativo para manejar sus ficheros, ahora es el SGBD es el encargado de facilitar los servicios de acceso de las aplicaciones a los datos.

Los programas piden al SGBD ciertos datos de la BD y éste, con la información y herramientas de que dispone, las convierte en operaciones de acceso a los distintos ficheros de la BD, operaciones que son ejecutadas por los *métodos de acceso*; entendemos por tales el conjunto de rutinas del Sistema Operativo que gestionan al nivel más próximo al hardware el acceso a los ficheros.

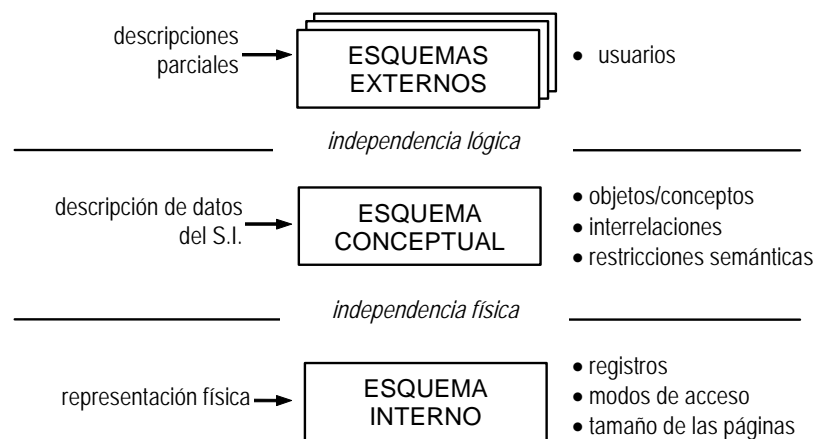


Como se observa, el SGBD introduce un nuevo nivel de independencia entre los usuarios y el hardware que no existe en un sistema clásico en el que los programas se comunican directamente con los métodos de acceso.

Uno de los objetivos de las BD es la independencia de datos. Para alcanzar dicho objetivo se han hecho distintas propuestas para definir la estructura ideal de un SGBD. El grupo ANSI/SPARC³⁴ (1977) propuso una arquitectura a tres niveles, donde se distinguen tres esquemas de BD:

- Esquema Conceptual (EC)
- Esquema Interno (EI)
- Esquemas Externos (EE)

El EC es la descripción del sistema de información (objetos e interrelaciones) con independencia del SGBD que se vaya a utilizar; el EI es la descripción del EC en términos de representación física (la forma de almacenarlo en el ordenador), y los EE son las vistas parciales (subconjuntos del conjunto global de información) que tienen los distintos usuarios.



Sin embargo, se encontraron con que no disponían de un Modelo Conceptual general y accesible desde cualquier SGBD que nos permita definir el esquema conceptual. Generalmente, los modelos de datos soportados por los SGBD comerciales suelen ser demasiado pobres

³⁴American National Standard Institute / System Planing And Requeriment Committee.

semánticamente³⁵, por lo que se prefiere realizar la descripción del sistema de información en algún modelo más expresivo para luego “traducirlo” a un SGBD concreto.

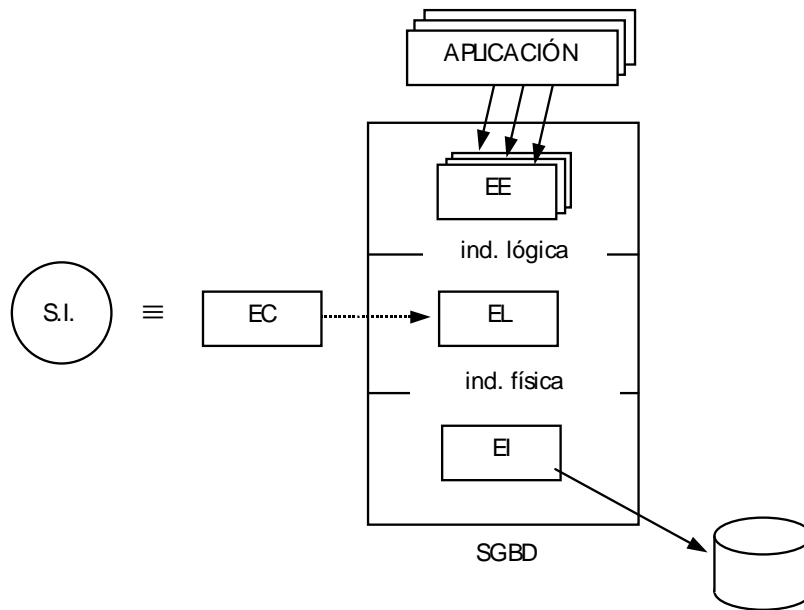
La intención es disponer de una primera descripción lo más completa posible e independiente de las herramientas que se vayan a utilizar para gestionarlo, incluso con la posibilidad de que no se vaya a mecanizar mediante un ordenador.

Por esta razón muchos autores prefieren distinguir dos esquemas en lugar del EC propuesto por ANSI/SPARC, lo que la convierte en una arquitectura a cuatro niveles:

- **Esquema Conceptual:** visión desde un punto de vista organizativo, independiente del SGBD que se utilice, e incluso de la utilización o no de sistemas de bases de datos. En este esquema se describe la información de la organización (objetos y relaciones) desde un punto de vista no informático.
- **Esquema Lógico:** visión expresada en términos de un SGBD concreto, o mejor dicho, de un modelo de datos soportado por un SGBD. En este esquema lógico se representan las entidades y relaciones de acuerdo a las características de dicho modelo, sin entrar todavía en detalles de representación física.
- **Esquema Interno** (o Esquema Físico): descripción de la representación en la memoria externa del ordenador de los datos del esquema lógico, sus interrelaciones y los instrumentos para acceder a ellos.
- **Esquemas Externos:** cada uno de ellos describe los datos y relaciones entre ellos de interés para una aplicación dada. Estos esquemas pueden verse como subconjuntos del Modelo Lógico de la BD³⁶.

³⁵Por ejemplo, en la asignatura Bases de Datos 2 se utiliza el modelo Entidad-Relación Extendido para el esquema conceptual y el modelo relacional para el esquema lógico.

³⁶ Por evitar una posible asociación de ideas con algo ya visto, las vistas tienen cierto parecido conceptual con los esquemas externos, en el sentido de que muestran una porción de la totalidad del sistema de información, pero un usuario de un determinado esquema externo, dentro de su visión limitada, puede utilizar o no vistas sobre sus datos particulares. De hecho, un esquema externo, en su equivalente relacional, es un conjunto de varias vistas.



Un buen SGBD debe permitir:

- Que se puedan describir los distintos esquemas de la BD, y en cada uno los aspectos que interesen de los datos y sus interrelaciones.
- Lo que finalmente hemos llamado Esquema Conceptual se define exteriormente al SGBD en términos de un cierto modelo de datos, realizándose manualmente la transformación entre el Esquema Conceptual y el Lógico.
- Existen distintos modelos para el diseño del EC como el Entidad-Relación, RM/T, Modelo Semántico General, etc...
- Que se puedan establecer las *correspondencias* entre los esquemas, es decir, indicar que tal dato o relación en un esquema se puede traducir a su correspondiente en otro.
- En la siguiente tabla tenemos un ejemplo ilustrativo de dos esquemas externos, uno escrito en PL/I y el otro en COBOL, que el SGBD ha de saber transformar, primero al correspondiente esquema lógico y, después, al interno.

esquemas EXTERNOS

(PL/I)	(COBOL)
DCL 1 EMPP,	01 EMPC.
2 #EMP CHAR(6),	02 EMPNO PIC X(6).
2 SAL FIXED BIN(31);	02 DEPTNO PIC X(4).

esquema LÓGICO

EMPLEADO	
NÚMERO_EMPLEADO	CARACTER(6)
NÚMERO_DEPARTAMENTO	CARACTER(4)
SALARIO	NUMÉRICO(5)

esquema INTERNO

STORED_EMP	LENGTH=20
PREFIX	TYPE=BYTE(6), OFFSET=0
EMP#	TYPE=BYTE(6), OFFSET=6, INDEX=EMPX
DEPT#	TYPE=BYTE(4), OFFSET=12
PAY	TYPE=FULLWORD, OFFSET=16

- Aislar los esquemas de manera que no se vean afectados por cambios efectuados en otros. A este requisito se le conoce

como *independencia de datos*, y podemos establecer dos tipos distintos según afecten a unos esquemas u otros:

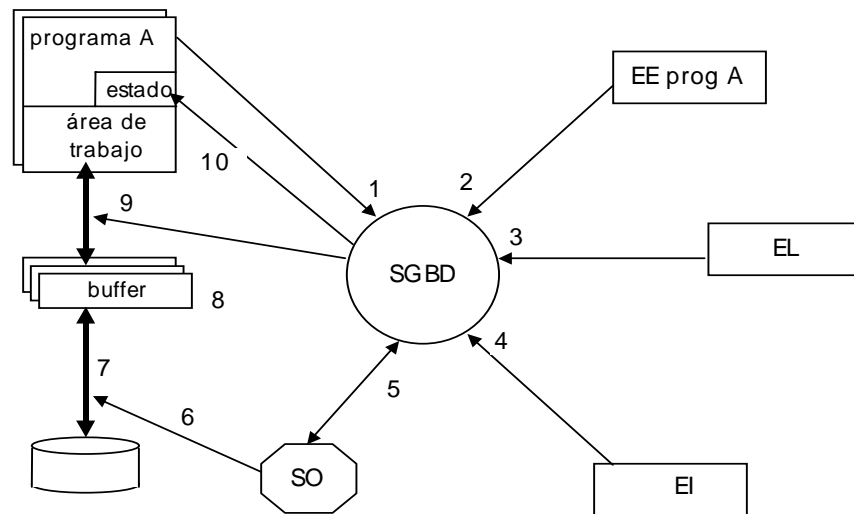
- **Independencia Lógica** (entre esquema lógico y esquemas externos): los esquemas externos y los programas de aplicación que se basan en ellos no deben verse afectados por modificaciones del EL referente a datos o relaciones que no usan.
- Si es suprimido un dato del EL no deberán verse afectados los EE que no lo incluyan en su definición; ni los programas de aplicación que no lo usen, independientemente de que dicho dato aparezca en su esquema externo asociado. En cambio deberán modificarse aquellos EE en los que aparezca el dato suprimido, y compilados de nuevo los programas que lo utilicen directamente.
-
- **Independencia Física** (entre esquema lógico y esquema interno): el EL no debe verse afectado por cambios en el EI referente a modos de acceso, tamaño de páginas, etc...

El SGBD dispone de la definición de cada uno de estos esquemas para satisfacer las peticiones de los programas. Se muestra en este momento una posible interacción entre el SGBD, el Sistema Operativo y los programas de aplicación a la hora de atender una petición de acceso a los datos.

EJEMPLO

- El programa de aplicación A hace una llamada al SGBD.
- El SGBD obtiene el EE utilizado por A y examina la descripción de los datos solicitados.
- El SGBD obtiene el EL y realiza la correspondiente transformación entre EE y EL.
- El SGBD examina la descripción física (EI) de la BD.
- El SGBD emite una orden al SO indicando los que debe leer.
- El SO interacciona con el almacén físico en el que se encuentran los datos.
- Los datos se transfieren desde la memoria externa a las memorias intermedias del sistema.
- Comparando el EL y el EE del programa A el SGBD deduce los datos pedidos por el programa de aplicación. El SGBD ejecuta todas las transformaciones necesarias.
- El SGBD transfiere los datos desde las memorias intermedias al área de trabajo del programa de aplicación.
- El SGBD suministra al programa información de estado sobre el resultado de su petición, incluyendo cualquier posible indicación de error.

- El programa de aplicación puede ahora operar con los datos pedidos que ya se encuentran en su área de trabajo.



Además de estas operaciones, el SGBD se ocupa de los controles de *privacidad* e *integridad* que hayan sido previstos tanto en el esquema lógico como en el esquema externo. Asimismo recoge información para la posible reconstrucción de la base de datos en caso de fallo.

Conviene hacer una aclaración en este punto. La mayoría de los SGBD comerciales trabajan como si únicamente tuvieran un esquema: el lógico. La percepción del usuario es que sólo tiene que definir la estructura de la base de datos en función del modelo de datos que soporta el programa, siendo totalmente transparente para él la organización que utiliza a nivel físico de los datos. Únicamente en sistemas muy grandes, el administrador de la base de datos ha de tomar decisiones sobre cómo se deben guardar los datos en los sistemas de almacenamiento de la máquina, generalmente para aumentar el rendimiento del sistema.

VIII3. el administrador de la bd

Dentro de la Organización, el encargado de controlar los aspectos técnicos del sistema de información mecanizado es el administrador de la base de datos (DBA). En resumen, es el encargado de definir la base de datos dentro del SGBD y de optimizar su rendimiento, al mismo tiempo que da soporte a las necesidades específicas de cada usuario.

Las funciones de un administrador de la BD son:

- *Definir el esquema lógico* (suponiendo la arquitectura a cuatro niveles) Partiendo el esquema conceptual que describe el sistema de información, el DBA transformará dicho esquema en el esquema lógico que manejará el SGBD a utilizar. Al realizar esta transformación, tomará las decisiones oportunas bajo criterios de optimización de la eficiencia del sistema final. Debe,

así mismo, especificar las correspondencias a establecer entre los esquemas conceptual y lógico.

- *Definir el esquema interno.* El DBA debe decidir cuál será la forma en que se almacenarán físicamente los datos, y establecer, al igual que antes, las debidas correspondencias entre esquemas.
- *Definir los esquemas externos.* Atendiendo a las necesidades de los usuarios, el DBA debe asegurar que toda la información necesaria está disponible y accesible por aquellos que la precisan, y debe crear (o asesorar en su creación) los esquemas externos de cada aplicación que definirán las partes del sistema de información a las que accederán. Además, el DBA será el encargado de suministrar asesoramiento técnico para el diseño de aplicaciones, formación de usuarios y programadores, etc.
- *Definir las reglas de integridad y seguridad.* Se encargará de definir aquellas reglas de integridad que actúan sobre los datos y las operaciones en términos que entienda el SGBD. Los permisos otorgados a los usuarios sobre los datos, y la política de transferencia de permisos entre ellos es, también, responsabilidad del DBA.
- *Especificar los mecanismos de recuperación y copias de seguridad.* El sistema de base de datos es un elemento crítico en el funcionamiento de la Organización, y el DBA debe especificar los procedimientos adecuados para la realización de copias de seguridad periódicas y recuperación de datos ante eventuales daños en parte o en todo el sistema de información almacenado.
- Verificar el rendimiento del sistema y atender a los cambios en los requerimientos.
- Como parte muy importante de sus funciones, el DBA debe vigilar que el sistema de información no se degrade, por la continua manipulación y actualización de sus datos, hasta el punto de afectar a los tiempos de respuesta o la sobreocupación de memoria secundaria. Se responsabilizará de las reorganizaciones necesarias, a cualquier nivel, que permitan mantener o mejorar el rendimiento de todo el sistema, incluido la renovación del hardware y el software si fuera necesario.

VIII4. componentes y funciones de un SGBD

Ya se han mencionado, de una forma u otra, cuales son las **funciones** de un SGBD:

- *Descripción* de los distintos esquemas de la base de datos.
- *Manipulación* de la base de datos: recuperación y actualización de la información.

- *Gestión*: lenguajes huésped, lenguajes conversacionales, utilidades para el administrador de la BD, etc.

Dicho de otra manera, el SGBD debe dar soporte a los siguientes conceptos:

- Definición de datos
- Manipulación de datos
- Seguridad e Integridad de los datos
- Recuperación de datos y concurrencia
- Diccionario de datos
- Rendimiento

A fin de cumplir con dichas funciones el SGBD debe proporcionar las herramientas adecuadas:

- Lenguajes de definición de esquemas (DDL)

Con ellos podremos definir los distintos esquemas con los que va a trabajar el SGBD (excepto, seguramente, el EC).

- Lenguajes de manipulación de datos (DML)

Con ellos podremos insertar, eliminar, modificar y consultar la información almacenada en la base de datos.

Decimos que estos lenguajes son autosuficientes si disponen de un traductor propio, y de tipo huésped si no disponen de estructuras de control como if-then-else, bucles, etc. En estos últimos, las instrucciones de acceso a la BD del SGBD deben ser intercaladas entre las de un programa escrito en algún lenguaje de programación. Estas instrucciones son traducidas por un precompilador que genera un programa en el lenguaje anfitrión puro (por ejemplo: COBOL, C, etc.); posteriormente, en el montaje, se añadirán las rutinas de librería que permitirán al programa, finalmente, manejar la BD.

- Controles de seguridad e integridad de los datos.

El SGBD debe supervisar todas las peticiones de acceso y rechazar aquellas que violen las reglas de integridad y seguridad definidas para el sistema de información representado en la BD.

Se deben establecer los métodos de control de acceso concurrente a la BD (varios usuarios que intentan acceder al mismo dato en el mismo período de tiempo) y la forma de recuperar la BD ante catástrofes (cortes de luz, fallos de hardware, ...), transacciones no completadas, etc.

- Diccionario de datos

Información adicional sobre la definición de datos (datos sobre los datos), también conocida como metadatos, que puede incluir los propios esquemas de BD, referencias cruzadas sobre qué datos utiliza qué programa, reports de cada usuario, terminales conectadas, etc., y que se utiliza fundamentalmente como documentación del análisis y diseño. Esta información no la utilizaría el usuario final y es mantenida bajo la responsabilidad del administrador de la BD.

- Utilidades.
- Módulos de reestructuración de la BD, a nivel físico. Son programas que obtienen copias secuenciales de la BD y la reorganizan cambiando detalles de caminos de acceso, tamaño de las páginas, etc.

- Módulos de impresión (generadores de informes).
- Generadores de Prototipos, con los que poder simular el funcionamiento de la BD conocido su diseño, generalmente con la intención de consultar a los usuarios finales sobre la calidad del diseño; con prototipos sucesivos se puede ir afinando los requerimientos del sistema hasta obtener una versión definitiva que se pueda optimizar en eficiencia. Actualmente, los sistemas de gestión se acompañan de lenguajes de cuarta generación (4GL) y generadores automáticos de aplicaciones.
- Módulos para la obtención de estadísticas.

Haciendo uso de todo ello correctamente podremos alcanzar los objetivos perseguidos por las técnicas de bases de datos: *independencia, integridad y seguridad de datos*. Profundizaremos un poco más en estos conceptos a continuación.

VIII5. independencia, integridad y seguridad

Independencia de datos.

Entendemos por independencia de datos el hecho de que los programas y esquemas no se vean afectados por cambios en datos que no usan.

En este punto es conveniente aclarar dos conceptos que influyen decisivamente en el grado de independencia: granularidad de los esquemas externos y ligadura.

Granularidad de los esquemas externos

Es el grado de detalle de definición de los esquemas externos en función del esquema lógico. Los más usuales son:

- de registro completo.
- de campo de registro.

Se dice que la granularidad es más *fin*a en el segundo caso que en el primero, y cuanto más fina sea mayor grado de independencia de datos conseguiremos.

Es evidente que si la granularidad es fina, un determinado esquema externo puede precisar de la definición de un único campo y, por tanto, modificaciones en el resto de campos del mismo registro no le afectarán; si la granularidad es a nivel de registro completo, aún sin

utilizar todos los campos, debería incluir la definición del registro y sería modificado en cualquier alteración del mismo en el esquema lógico.

Ligadura

Se entiende por ligadura el momento en que se transforman los esquemas externos usados por las aplicaciones en términos de esquema interno.

Una definición de dato de un EE, en el momento de la ligadura, se transforma en una longitud y una posición dentro de un registro, términos en los que se expresa el EI. No obstante, puesto que se hace necesaria una primera transformación entre EE y esquema lógico, y de éste al EI, se diferencia entre ligadura lógica y ligadura física.

El momento en que se produce la ligadura desaparece la independencia de datos puesto que el esquema externo ya ha sido traducido al más bajo nivel. Por eso, si el momento de la ligadura se produce cuando se compilan los programas, cualquier alteración en el esquema interno provocará la recompilación, aunque tal modificación se haya producido en un dato que no usen.

Podemos decir, por entendernos, que al realizar la ligadura en el momento de la compilación, los esquemas externos “desaparecen”, puesto que en cada ejecución del programa se parte ya de los datos físicos de almacenamiento.

Si, por el contrario, es en cada acceso a la base de datos cuando se realiza la ligadura, cada vez se realizará una nueva traducción del esquema externo en esquema interno. Eso implica que, si el cambio en el EL o EI no le afecta directamente, no habrá necesidad de modificar su correspondiente EE.

No obstante, cuanto más tardía sea la ligadura (por ejemplo, en cada acceso) la eficiencia será menor puesto que continuamente estaremos haciendo uso de las correspondencias entre esquemas.

Se consideran cuatro momentos en los que puede tener lugar la ligadura, siendo el último el más tardío y el que garantiza una mayor independencia de datos:

- en la compilación
- en el montaje (link)
- al iniciarse la ejecución del programa
- en cada acceso a la BD.

La independencia de datos, resumiendo, es mayor cuanto más fina sea la granularidad de los esquemas externos y cuanto más tardío el momento de la ligadura. La tendencia de los SGBD actuales es apoyarse en la granularidad a nivel de campo y la ligadura al comienzo de la ejecución del programa.

integridad de datos

La integridad de datos atiende a la calidad de la información almacenada en los siguientes aspectos:

- los valores de los datos han de ser correctos.
- las ocurrencias de los datos (los valores en un instante determinado) han de estar debidamente interrelacionados.
- no se deben producir interferencias en las lecturas y escrituras concurrentes, del tipo de actualizaciones incorrectas, bloqueos activos o mortales, etc.

Se distinguen dos aproximaciones a este problema: *optimista* y *pesimista*. La primera da por supuesto que tales problemas se presentan muy de cuando en cuando, así que la solución más eficiente es solucionarlos cuando han ocurrido. Dichos problemas pueden venir de deterioros de la BD por catástrofes, interrupciones en procesos de actualización, actualizaciones indebidas por fallo humano, etc. De hecho, la táctica es llevar un registro, más o menos actualizado, de los estados sucesivos de la base de datos o de las últimas transacciones efectuadas para devolver a la base de datos a un estado anterior que se sabía correcto, y volver a comenzar desde ese punto.

La segunda identifica las situaciones que pueden provocar un conflicto y retrasa su ejecución hasta que tal posibilidad desaparece.

Así, pues, tenemos:

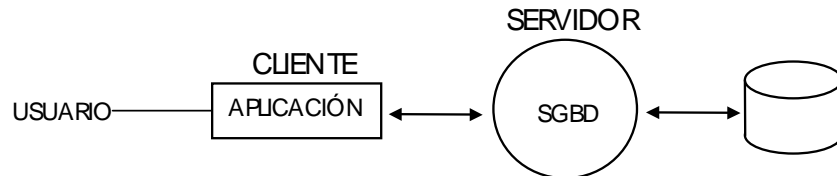
- *técnicas preventivas*: verificación de las restricciones semánticas, controlar la consistencia de las copias redundantes, y control de los accesos concurrentes, generalmente por el método de reservas (evitar que un programa acceda al dato mientras lo está modificando y/o consultando otro).
- *técnicas curativas*: fichero dietario con el que deshacer o rehacer todas las actualizaciones realizadas por transacciones durante un tiempo determinado, copias de seguridad o volcados de la BD completa en un momento en que todas las transacciones efectuadas han sido confirmadas, toma de puntos de control, ...

Seguridad de los datos

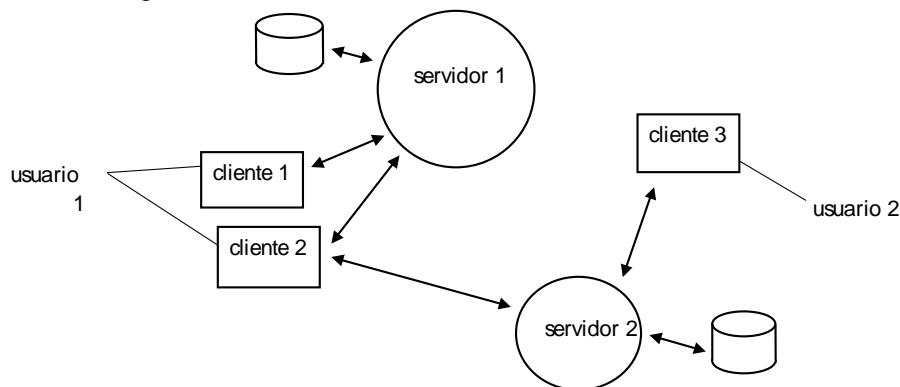
Se refiere a la posibilidad, por parte de los usuarios, de realizar determinado tipo de operaciones sobre los datos de la BD, y sólo a aquellos usuarios que estén autorizados. Se utilizan técnicas tales como la identificación del usuario (*login* y *password*), determinación de permisos de lectura, escritura, etc., gestión de autorizaciones transferibles de un segundo usuario a un tercero, etc.

VIII6. arquitectura cliente-servidor

Aparte de la arquitectura a cuatro niveles (o a tres) mencionada anteriormente, se puede dar otra visión de un SGBD frente a las aplicaciones que deben atacarle para acceder a una determinada base de datos. La arquitectura cliente-servidor ofrece una visión simplificada de un SGBD, de tal forma que considera únicamente dos partes:



- El *servidor* (o frontend), que es en realidad el propio SGBD y mantiene todas las características antes apuntadas (esquemas para cada nivel de arquitectura, lenguajes de manipulación y definición de datos, etc.), es el encargado de satisfacer todas las peticiones de acceso y recuperación de datos de sus bases de datos.
- Los *clientes* (o backend) son las aplicaciones que realizan dichas peticiones. Estas aplicaciones pueden ser las desarrolladas para un usuario en particular o las propias herramientas suministradas por la empresa vendedora del SGBD o por terceras compañías que fabrican software para ese SGBD: lenguajes de consulta, generadores de prototipos, generadores de informes, interfaces de usuario, etc.



La utilidad de esta visión de alto nivel de un SGBD está en el procesamiento distribuido de la BD. Dentro del procesamiento distribuido de BD, la arquitectura cliente-servidor permite que el servidor atienda varias peticiones simultáneamente, y el cliente podrá acceder a varios servidores al mismo tiempo; estos servidores no tienen porque mantener el mismo SGBD, ni tan siquiera el mismo modelo de datos subyacente. Tan sólo es necesario establecer los correspondientes protocolos para que clientes y servidores se puedan entender.

El cliente, además, es capaz de procesar en su máquina la información que le ha proporcionado el servidor, de tal manera que ese trabajo ya no lo tiene que realizar la máquina remota, favoreciendo los tiempos de respuesta a través de la red. De hecho, uno de los objetivos de esta arquitectura es descargar al servidor de trabajo.

Pensemos en aplicaciones desarrolladas para la red internet: es el cliente el que se encarga del procesamiento de la información para su visualización dentro del explorador, el interfaz del usuario frente a la red. El servidor lo único que hace es proporcionar respuesta a las peticiones del cliente, suministrar los datos tan sólo, y éste último se responsabiliza de las tareas gráficas de presentación de los datos en la pantalla de su ordenador. El único requisito es que existan unos protocolos definidos para la comunicación entre el cliente y el servidor. Así se asegura que los datos que van de uno a otro sean entendidos por quien los recibe. De esta forma, el cliente puede trabajar con un sistema operativo distinto del servidor, no tener software alguno de gestión de bases de datos, y todo ello sin que el usuario necesite más conocimiento que el manejo de su programa de acceso a la red. En definitiva, se puede ver como un nivel más de independencia de las aplicaciones frente al hardware y la representación física de los datos.

BIBLIOGRAFÍA

[CELMA97]

[DATE2001]

[KORT87]

IX INTRODUCCIÓN AL MODELO ENTIDAD-RELACIÓN EXTENDIDO

Ante la falta de ciertos mecanismos para representar la realidad de la que adolecían los modelos de datos clásicos, surgieron los modelos de datos semánticos.

Básicamente, se puede decir que los modelos de datos clásicos atendían más a cuestiones de acceso de los datos, es decir, modelaban la realidad en función de las estructuras a utilizar y la forma de rentabilizar estos accesos en tiempo y espacio.

Por contra, los modelos de datos semánticos dejan a un lado ese aspecto y se centran más en el comportamiento de los objetos dentro de su entorno, buscando una representación más natural e intuitiva de las relaciones y el funcionamiento del Sistema de Información. Se inscribe, por tanto, totalmente dentro del paradigma Orientado a Objeto.

Los modelos de datos semánticos incorporan conceptos y mecanismos de abstracción que no están permitidos, o al menos no se pueden representar de una forma natural, en los modelos previos. Sin embargo, no tienen un reflejo en algún SGBD comercial y únicamente se utilizan como herramienta de análisis para la confección del Esquema Conceptual de la BD, que posteriormente se ha de traducir en términos de modelos de datos si implantados comercialmente.

Como ejemplo de modelo semántico se propone el Entidad-Relación (E-R), que es, posiblemente, uno de los de mayor aceptación a la hora de confeccionar los esquemas conceptuales de base de datos.

IX1. modelo entidad-relación

El modelo E-R (entidad-relación) fue propuesto por E. Chen en 1976 para la definición del esquema conceptual de una BD. Posteriormente se ha ido enriqueciendo con nuevos mecanismos de abstracción y representación de la realidad, lo que se conoce como el modelo EER (entidad-relación extendido). Es uno de los más ampliamente utilizados de los llamados semánticos.

Se basa en conceptos tales como entidad (objeto), atributo y relación entre objetos para representar los diferentes métodos de abstracción, a saber: clasificación, agregación y generalización. Se dispone de un formalismo gráfico para realizar estas representaciones, pero no de un lenguaje de manipulación de datos.

La principal ventaja, que seguramente ha forzado su difusión, es que es traducible casi automáticamente a un esquema de BD bajo Modelo Relacional, con cierta pérdida de expresividad en el proceso, pero garantizando que las tablas que resultan están directamente en Tercera Forma Normal (3FN).

Pasaremos ahora a describir cual es el lenguaje de representación de entidades, atributos, y relaciones entre entidades. El objetivo es proporcionar la base suficiente para interpretar diagramas EER.

- ***Representación de entidades.***

Una entidad se representará mediante un rectángulo nominado.

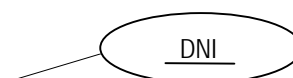


- ***Representación de atributos.***

Un atributo se verá en un E-R como una elipse unida a una entidad mediante un arco.

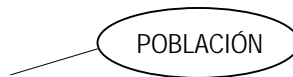
En función de los distintos tipos de atributos que nos podemos encontrar, variará el tipo de representación:

- ***atributo identificador:*** son aquellos que identifican las ocurrencias de la entidad. Se representan mediante el subrayado del nombre del



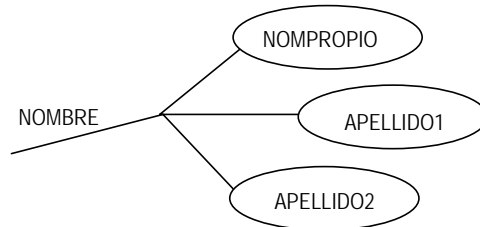
atributo.

- ***atributo descriptor:*** atributo no identificador.

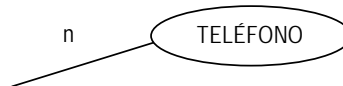


Si atendemos a su posible estructura:

- atributo simple o escalar.
- **atributo compuesto o estructurado**: el nombre del atributo compuesto es la etiqueta de un arco que se subdividirá en tantos atributos simples como forme la estructura.



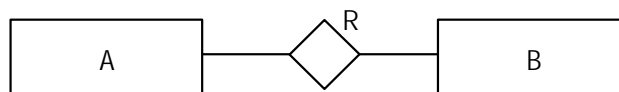
- **atributo multivaluado**: se indica mediante la etiqueta n sobre el arco.



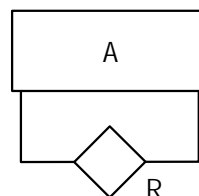
• **Representación de Relaciones.**

Las relaciones entre entidades se representan mediante un polígono de tantos lados como entidades se asocian, salvo en el caso de las binarias (relaciones que asocian dos entidades o una consigo misma) que utilizan un rombo, unido a las entidades mediante arcos. Este polígono irá etiquetado con el nombre de la relación. Asimismo, se pueden etiquetar los arcos para realzar el papel que juega dicho objeto dentro de la relación.

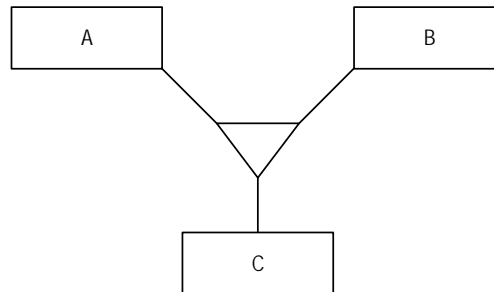
- relaciones binarias.



Un caso especial serían aquellas relaciones de un objeto consigo mismo: las *relaciones reflexivas*.



- *relaciones ternarias.*



- **Representación de Restricciones.**

- Sobre atributos.

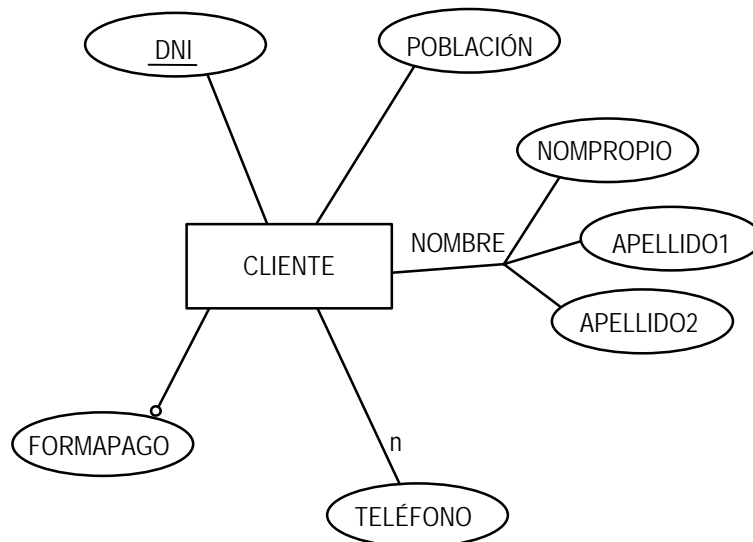
Las restricciones de valor se pueden indicar colocando al lado del atributo el dominio sobre el que se define el mismo.

Si un atributo **no puede tomar valores nulos** lo representaremos como:



- Sobre entidades.

Toda entidad debe tener su conjunto de atributos identificador.



- Sobre relaciones.

Las restricciones de cardinalidad se expresan mediante el rallado de la parte del polígono que indica la asociación de una de las entidades. Dependiendo del número de entidades asociadas la cardinalidad máxima implica una lectura u otra. Sólo nos vamos a fijar en las relaciones binarias.

Suponiendo dos entidades A y B asociadas por una relación R, definimos las restricciones de cardinalidad mínima y máxima como:

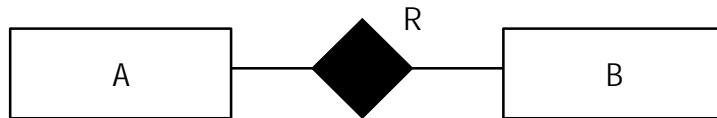
$$\text{Card}(A, R) = (m_A, M_A),$$

$$\text{Card}(B, R) = (m_B, M_B)$$

Las restricciones de **cardinalidad máxima** se expresa por el rallado de la parte del polígono (en este caso, relaciones binarias, la mitad del rombo) que indica la asociación de una de las entidades. Nos encontraremos con los siguientes casos:

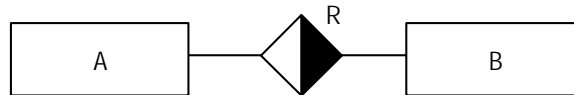
$$\text{Card}(A, R) = (0, n),$$

$$\text{Card}(B, R) = (0, n)$$



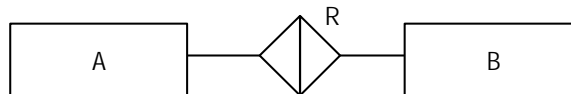
$$\text{Card}(A, R) = (0, n),$$

$$\text{Card}(B, R) = (0, 1)$$



$$\text{Card}(A, R) = (0, 1),$$

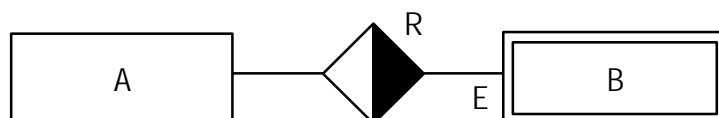
$$\text{Card}(B, R) = (0, 1)$$



Para las restricciones de cardinalidad mínima, esto es, las de tipo existencial, utilizaremos un doble rectángulo para la(s) entidad(es) que sufren la restricción, y etiquetaremos el arco de la relación con una "E":

$$\text{Card}(A, R) = (0, n),$$

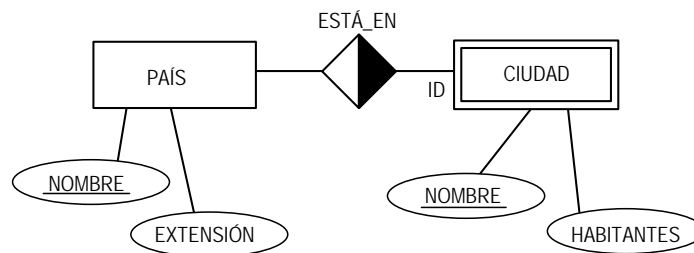
$$\text{Card}(B, R) = (1, 1)$$



- Restricción de dependencia de identificador.

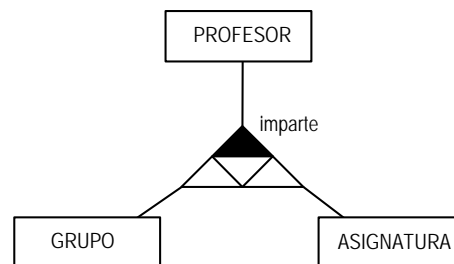
Aquellas entidades que no se puedan identificar por sí misma, es decir, que necesitan de los atributos identificadores de otra entidad para reconocer cada ocurrencia de su entidad, se dice que sufren *dependencia de identificador*. Se las conoce como *entidades débiles*, en contraposición con la entidad que “presta” sus identificadores, que se considera *fuerte*.

Se representa igual que la restricción existencial, pero etiquetando el arco con “ID”.



- Ternarias

La ternaria es una relación que asocia tres objetos, similar a la binaria que lo hace con dos. La peculiaridad de esta conectividad estriba en que se ha de leer como “todas las parejas contra el tercero”.



Sea, por ejemplo, la ternaria del esquema adyacente. Decimos que:

- “un profesor imparte una asignatura sólo en un grupo”
- “un profesor en un grupo imparte una única asignatura”
- “una asignatura en un grupo puede ser impartida por muchos profesores”

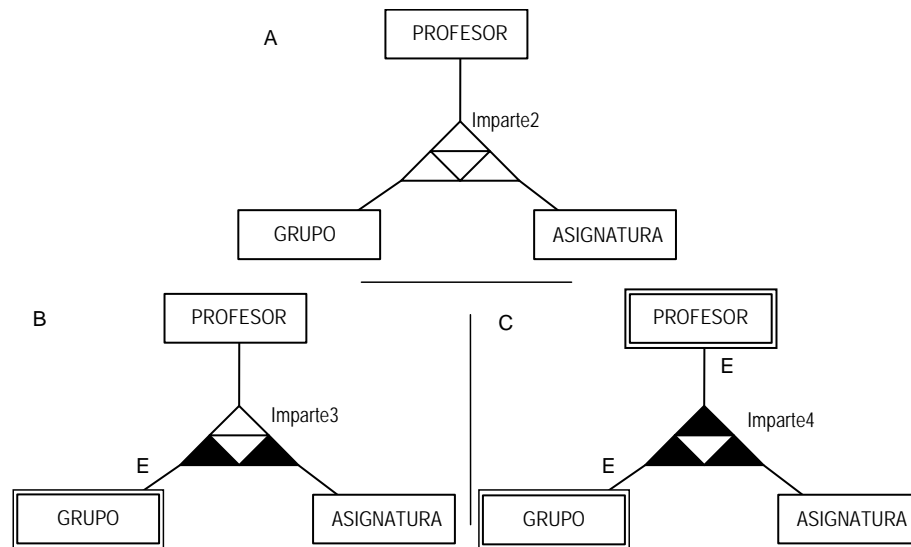
De una forma menos ambigua:

Card ((PROFESOR, ASIGNATURA), IMPARTE) = (0, 1)
 Card ((PROFESOR, GRUPO), IMPARTE) = (0, 1)
 Card ((ASIGNATURA, GRUPO), IMPARTE) = (0, n)
 Card (PROFESOR, IMPARTE) = (0, n)
 Card (ASIGNATURA, IMPARTE) = (0, n)
 Card (GRUPO, IMPARTE) = (0, n)

En realidad, a la hora de “leer” una ternaria, las únicas cardinalidades que tienen sentido son las máximas de las parejas y las mínimas de las entidades por si solas, ya que las otras no cambian nunca, sea cual sea el sentido de la ternaria, siempre hablando de la capacidad de representación del EER.

En general, las restricciones de cardinalidad representadas por los triángulos rayados o no acepta cualquier tipo de combinación, 1:1:1, 1:1:M,

1:M:M y M:M:M, así como restricciones de existencia en cualquiera de las entidades asociadas.



Caso A:

En este caso, la conectividad decimos que es 1:1:1, es decir, que para cada posible pareja de ocurrencias de entidad sólo una de la entidad restante se asocia con ella. O dicho de otro modo, una ocurrencia de grupo-asignatura (p. ej. <2A, FBD>) sólo puede aparecer una vez, como mucho, dentro de la relación *imparte2*.

Card ((PROFESOR, ASIGNATURA), IMPARTE) = (0, 1)
 Card ((PROFESOR, GRUPO), IMPARTE) = (0, 1)
 Card ((ASIGNATURA, GRUPO), IMPARTE) = (0, 1)
 Card (PROFESOR, IMPARTE) = (0, n)
 Card (ASIGNATURA, IMPARTE) = (0, n)
 Card (GRUPO, IMPARTE) = (0, n)

Caso B:

La conectividad 1:M:M nos dice mientras para un profesor que da clase a un determinado grupo, este puede impartir varias asignaturas y que para cada profesor que imparte una asignatura concreta lo hace en varios grupos, un único profesor imparte una asignatura concreta en un grupo determinado.

Por ejemplo, sea la siguiente ocurrencia de *imparte3* utilizando tan sólo los identificadores de cada entidad:

GRUPO	PROFESOR	ASIGNATURA
2A	ARMANDO	FBD
2A	EVA	DGBD
2B	ARMANDO	FBD
2B	EVA	DGBD
2C	EVA	FBD
2C	EVA	DGBD
2A	JAIME	FBD

La última ocurrencia no sería consistente con las restricciones de cardinalidad que representa la ternaria *imparte3*.

Por otro lado, la entidad *grupo* sufre una restricción de existencia respecto de *imparte3*, lo que obliga a que toda ocurrencia de *grupo* aparezca al menos una vez en la ternaria.

Nótese, sin embargo, que no se ofrece ningún mecanismo de representación para poner la cardinalidad mínima de una pareja frente a la ternaria a 1; no se puede obligar, con el EER, a que toda posible pareja de ocurrencias de dos entidades aparezca en la ternaria.

Card ((PROFESOR, ASIGNATURA), IMPARTE) = (0, n)
 Card ((PROFESOR, GRUPO), IMPARTE) = (0, n)
 Card ((ASIGNATURA, GRUPO), IMPARTE) = (0, 1)
 Card (PROFESOR, IMPARTE) = (0, n)
 Card (ASIGNATURA, IMPARTE) = (0, n)
 Card (GRUPO, IMPARTE) = (1, n)

Caso C:

La relación es M:M:M, es decir, se puede dar cualquier combinación de ocurrencias de entidad, teniendo en cuenta, además, que tanto la entidad *grupo* como la de *profesor* sufren una restricción de existencia cada una respecto de *imparte*⁴.

Card ((PROFESOR, ASIGNATURA), IMPARTE) = (0, n)
 Card ((PROFESOR, GRUPO), IMPARTE) = (0, n)
 Card ((ASIGNATURA, GRUPO), IMPARTE) = (0, n)
 Card (PROFESOR, IMPARTE) = (1, n)
 Card (ASIGNATURA, IMPARTE) = (0, n)
 Card (GRUPO, IMPARTE) = (1, n)

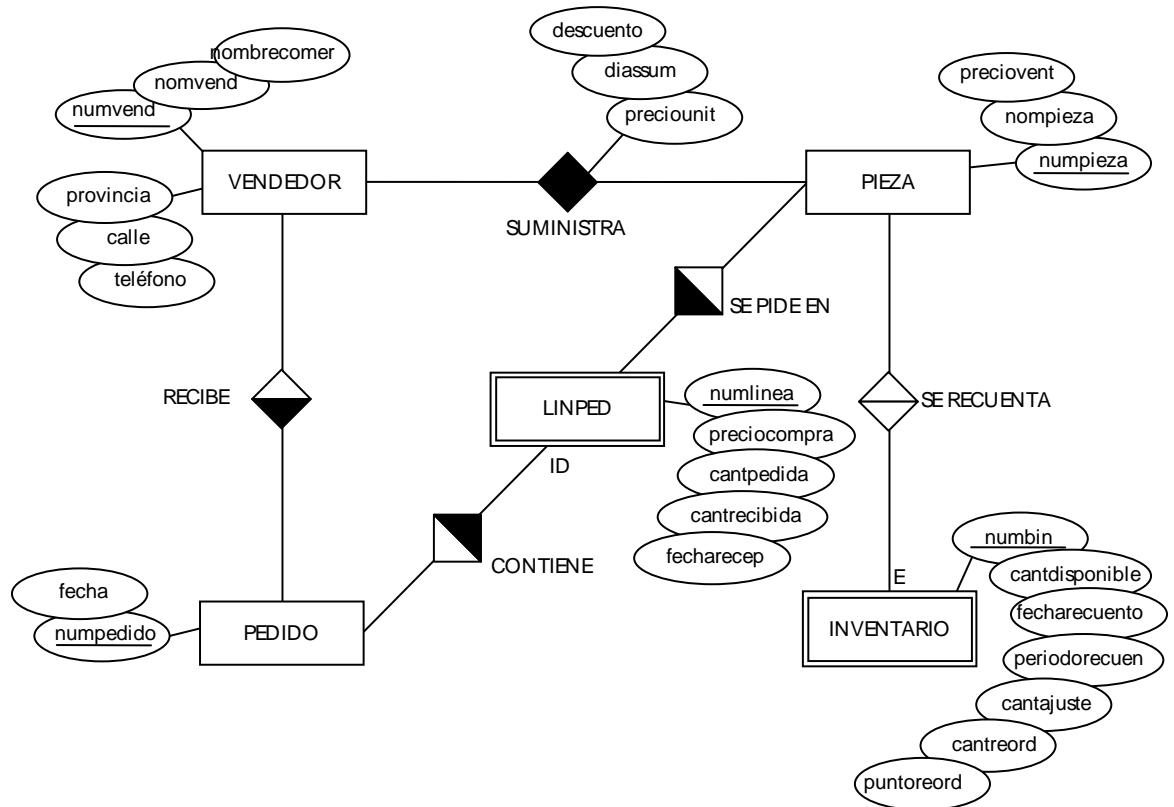
IX2. ejemplo 1

Mostramos a continuación el diagrama E-R para un Sistema de Información sobre la gestión de pedidos a proveedores de una empresa genérica.

El ejemplo pretende reflejar la política de compras de una empresa de distribución. Se compran ciertas mercancías a los distintos proveedores y son vendidas posteriormente al público o a otros distribuidores.

Básicamente, las tareas que se pretenden mecanizar son las siguientes:

- Lista de precios de compra.
Conocer en todo momento los precios a los que venden en el momento actual los proveedores.
- Control de pedidos.
De aquellas mercancías que se solicitan a los proveedores, controlar si se han servido en el tiempo estimado y en la cantidad pedida.
- Control de existencias.
Mediante la confección de un inventario, donde cada entrada, que corresponde a un único artículo, es el recuento real de existencias.



Lista de precios de compra a proveedores.

Tenemos almacenada la información de aquellos proveedores habituales de la empresa y de las piezas que entran y salen del almacén: son las entidades VENDEDOR y PIEZA, respectivamente.

Puesto que un mismo vendedor puede suministrar varias piezas, y una pieza puede ser ofertada por varios proveedores, nos interesa saber, para cada pieza, las condiciones que ofrecen unos y otros. La relación M:M entre las dos entidades mencionadas y que hemos denominado SUMINISTRA, contiene, además, los atributos que nos indican el precio actual al que se puede comprar, los días que tarda en ser servida la pieza, y el posible descuento ofrecido por el proveedor.

El siguiente ejemplo muestra una posible ocurrencia de la relación SUMINISTRA que asocia una ocurrencia de VENDEDOR, otra de PIEZA, y contiene los atributos particulares de la asociación.

ocurrencia de VENDEDOR

1	AGAPITO LAFUENTE DEL CORRAL	MECEMSA	96-5782401	Avda. Valencia 3205	ALICANTE	ALICANTE
---	-----------------------------	---------	------------	---------------------	----------	----------

ocurrencia de pieza

DD-0001-210	DISCO DURO WESTERN DIG 210M 28	25000
-------------	--------------------------------	-------

ocurrencia de SUMINISTRA

1	AGAPITO LAFUENTE DEL CORRAL		MECEMSA	96-5782401	Avda. Valencia 3205	ALICANTE	ALICANTE
DD-0001-210			DISCO DURO WESTERN DIG 210M 28		25000		
15000	3	15					

Se supone que almacenamos la información de las piezas que maneja la empresa y la de los vendedores que podrían suministrarlas. En la relación M:M SUMINISTRA se asocian efectivamente las piezas con los proveedores que las sirven, y el precio de oferta de cada uno. Nótese que, por la conectividad de la relación, un mismo vendedor puede proveer de varias piezas a la empresa, y una misma pieza puede ser ofertada por varios vendedores a distintos precios.

La siguiente tabla, y prescindiendo de la mayor parte de atributos que, como el ejemplo anterior, formarían cada ocurrencia de suministra, ilustra cómo se relacionarían vendedores y piezas por suministro.

numvend	...	numpieza	...	precio	...
V1		P1		1500	
V1		P2		10000	
V2		P1		1750	
V2		P3		9000	

control de pedidos.

La empresa decide en un momento dado solicitar a un proveedor que le suministre una serie de piezas. Se confecciona un pedido que se envía al vendedor que, posteriormente servirá tales mercancías. Toda esta operativa se refleja en dos nuevas entidades, PEDIDO y LINPED, y en tres relaciones, RECIBE, CONTIENE y SE_PIDE_EN.

Además, para controlar la recepción de los pedidos, se almacena información sobre la fecha en que entran efectivamente las mercancías en el almacén, el precio al que se han comprado, y la diferencia entre la cantidad que se pidió y la que se recibió.

Debemos aclarar ciertos aspectos sobre estas relaciones. En primer lugar, los precios de compra no tienen porqué coincidir con los de la lista de precios SUMINISTRA. Se puede dar el caso de que este precio variara recientemente y que el pedido sea de fecha anterior, o simplemente que el proveedor ha hecho una oferta distinta de la que se conoce por SUMINISTRA.

En segundo lugar, tampoco es obligatorio que las piezas que suministra un vendedor sean las que aparecen en la lista. En definitiva, no hay relación directa entre SUMINISTRA y el CONTROL DE PEDIDOS.

Por último, los pedidos se estructuran mediante una cabecera, cuyos datos principales se almacenan en *PEDIDO*, y una serie de líneas de pedido que contienen, cada una, una referencia a una única pieza. Precisa de una

restricción de dependencia de identificador puesto que, para todos los pedidos, las líneas se numeran desde 1 en adelante, con lo que el identificador *numlinea* no basta para diferenciar las tuplas de *LINPED*.

Para ilustrar este punto, se muestra un ejemplo de cómo serían dos fichas de control de pedidos servidos por los proveedores, tal y como las veríamos escritas en papel (no son ocurrencias de las entidades y relaciones del EC).

numpedido	numvend	fecha				
1	1	05/05/1992				
numlinea	numpieza	preciocompra	cantpedida	fecharecep	cantrecibida	
1	M-0001-C	30000	10	10/05/1992	10	
2	P-0001-33	21000	20	10/05/1992	18	
3	FD-0001-144	13500	20	10/05/1992	20	
4	DD-0001-210	15000	20	10/05/1992	20	
5	T-0002-AT	3100	22	17/10/1992	22	

numpedido	numvend	fecha				
2	1	11/10/1992				
numlinea	numpieza	preciocompra	cantpedida	fecharecep	cantrecibida	
1	DK144-0002-P	545	100	15/10/1992	101	
2	T-0002-AT	3000	1	15/10/1992	1	

Al igual que el ejemplo de SUMINISTRA, una ocurrencia de PEDIDO, no necesariamente, se asociaría a una ocurrencia de VENDEDOR; y una de LINPED, a una de PIEZA y, obligatoriamente, a una de PEDIDO (con todos los atributos de cada una).

Lo que se pretende ilustrar con este ejemplo es que el concepto de relación entre entidades no presupone ningún mecanismo concreto (como las claves ajenas en el modelo relacional, punteros o similares). Simplemente sabemos que existe una asociación entre ocurrencias de entidades y no nos interesa cómo sea la implementación, la representación física de los datos.

control de existencias.

Para controlar el nivel de existencias en nuestro almacén se ha optado (no tiene porqué ser la mejor alternativa) por una entidad INVENTARIO con una relación 1:1 con PIEZA, con restricción de existencia.

De cada pieza se mantiene el último recuento físico en el almacén, por si ha habido roturas o pérdidas que alteren la diferencia entre bienes que entran y que salen. También se especifica cual es el nivel mínimo de existencias, que indica cual es el momento adecuado para realizar un nuevo pedido a proveedores.

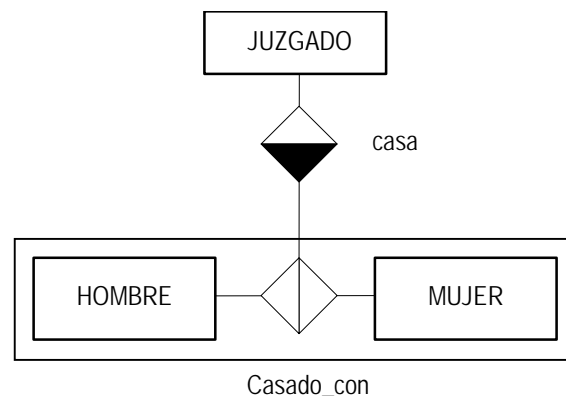
Cada ocurrencia de la entidad se identifica por un número de inventario (numbin) y obligatoriamente estará asociada a una (distinta para cada entrada de inventario) de PIEZA.

IX3. otros mecanismos de abstracción

IX3.1. agregación

Lo que en EER se entiende como agregación es un caso muy particular del concepto expuesto en el tema de modelos de datos. En EER, la agregación representa la creación de un objeto compuesto a partir de una relación entre entidades, de tal forma que éste se comporta como una entidad más, aunque de un nivel de abstracción mayor.

Sea, por ejemplo, el caso de hombre y mujeres que se unen en matrimonio (hemos obviado los atributos de las entidades para no complicar el diagrama). El hecho de que un hombre y una mujer se casen no significa que obligatoriamente lo hagan por lo civil (entendiendo como tal la acción de casarse ante una autoridad civil no eclesiástica). Sin embargo, para aquellas parejas que no hayan celebrado ceremonia religiosa, nos interesa saber el juzgado en que se han casado.



Evidentemente, si atendemos a la conectividad de la relación *casado_con*, un hombre sólo se puede casar con una única mujer y viceversa (en nuestro sistema de información). Si lo hicieran por lo civil, pasarían por un único juzgado, mientras que el mismo juzgado podría haber casado a muchas parejas.

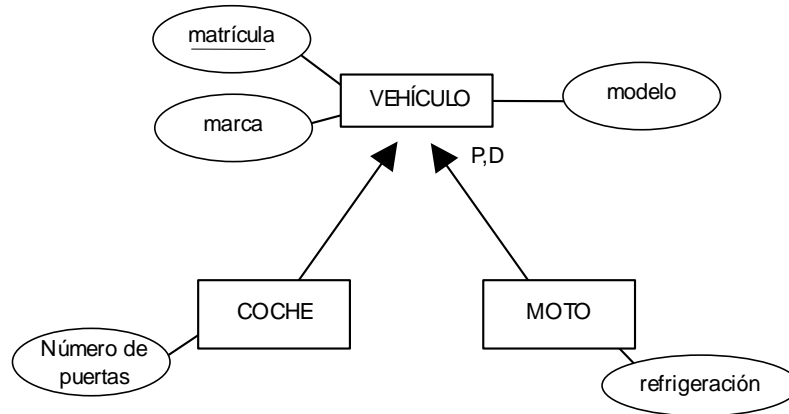
El mecanismo de representación lo que hace es abstraer las entidades y la relación que las asocia para obtener una entidad compleja, que a su vez puede relacionarse como una entidad normal con el resto de entidades de nuestro sistema.

Aunque tiene muchos puntos de contacto con una ternaria, la agregación remarca la relación entre una determinada pareja de entidades, al mismo tiempo que no implica una necesaria asociación con la tercera entidad, como sí ocurría en las ternarias.

Por último, recordar que las restricciones de existencia se pueden aplicar tanto a las entidades simples como a la nueva entidad agregada (podríamos obligar a que toda unión matrimonial fuera civil, pintando una "E" en el arco de la relación *casa* que la une con la entidad agregada *casado_con*).

IX3.2.generalización

La generalización representa fielmente el mecanismo de abstracción mostrado en el tema de modelos de datos. Recordamos que se trata de una jerarquización de entidades en tipos y subtipos, y que se aplican diferentes propiedades de cobertura.



El objeto especializado se une al objeto generalizado mediante una flecha, y las propiedades de cobertura se expresan mediante iniciales: P/T, D/S (parcial/total, disjunta/solapada).

IX4. ejemplo 2

Se trata de captar en un diagrama EER la información mínima que necesita recoger una localidad X sobre las elecciones municipales y autonómicas. Esa localidad tiene en su censo información sobre una serie de personas que pueden ejercer su derecho al voto en estas elecciones. Para las elecciones se han dispuesto una serie de colegios electorales en esta localidad, estos colegios vamos a suponer que se encuentran identificados por números correlativos y que conocemos el número total de electores, y dentro de ellos hay una serie de mesas que suponemos se nombran A, B, C...

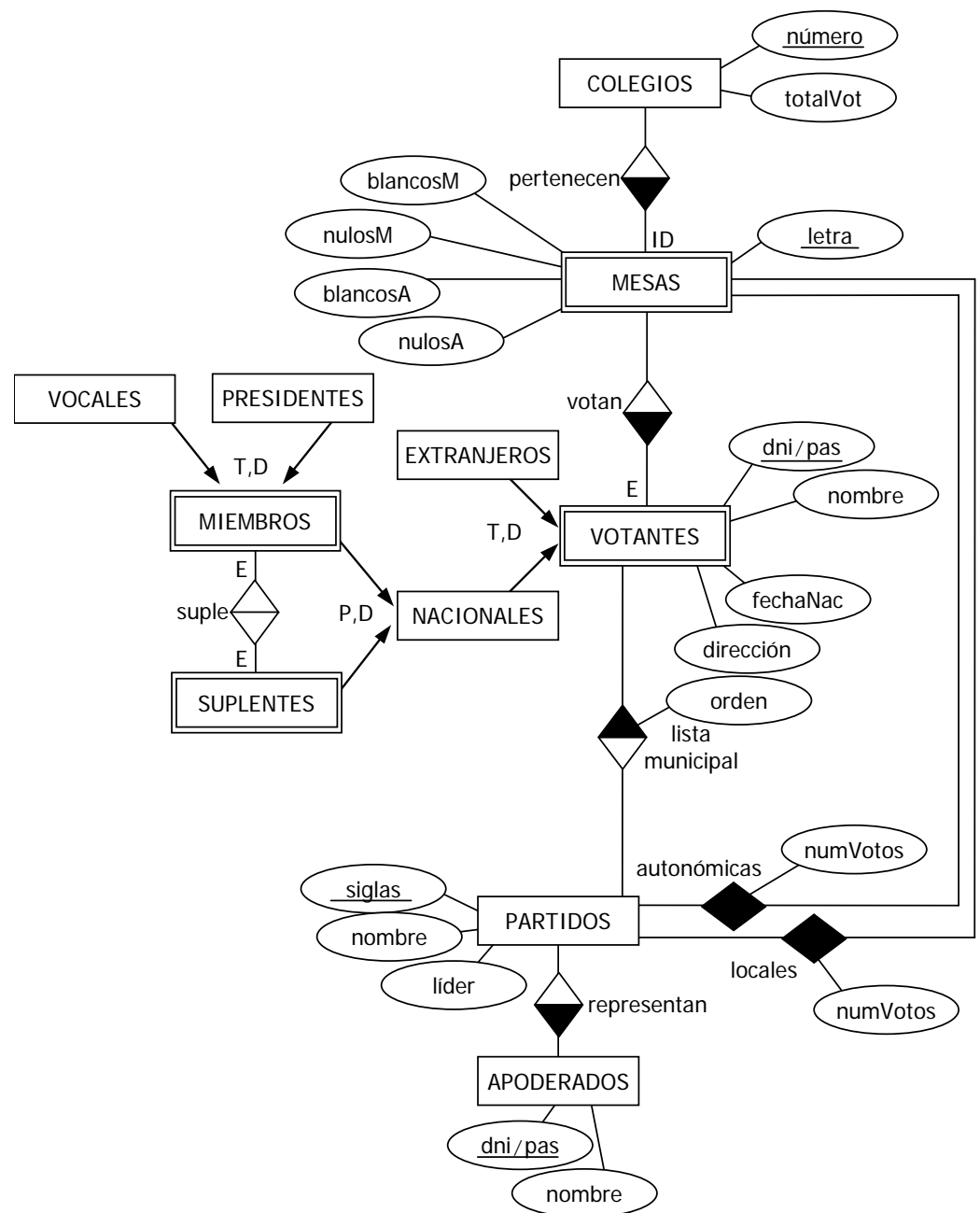
La información que mantiene sobre las personas del censo electoral de esta localidad es D.N.I./pasaporte, nombre, fecha de nacimiento, dirección y mesa en la que deben votar. De entre las personas que pueden votar hay algunas que no tienen la nacionalidad española pero que forman parte del censo para votar en las elecciones municipales. De las personas pertenecientes a cada una de las mesas y de nacionalidad española es necesario conocer a quién le ha tocado ser presidente de la mesa y a quién su suplente, a quiénes los vocales y también sus suplentes; estas figuras son imprescindibles en cada mesa.

Por otro lado se mantiene información sobre las formaciones políticas que se pueden votar en estas elecciones y de cada una de ellas además de sus siglas y su nombre completo guardamos el nombre del presidente del partido a nivel nacional, los nombres de las personas que integran su lista municipal (suponemos que son 4 personas por lista, deben pertenecer al censo de la localidad, y no pueden formar parte de ninguna mesa electoral) distinguiendo el orden que cada una ocupa en la lista; y los nombres y D.N.I. de los apoderados (que no tienen por qué pertenecer al censo y no están asignados a ninguna mesa) del partido en la localidad.

Una vez realizada la votación se debe de poder saber cuantos votos ha recibido cada uno de los partidos en cada una de las mesas tanto para las elecciones municipales como para las autonómicas, así como el número de votos en blanco y número de votos nulos que ha habido para cada una de ellas en cada mesa.

Cuestiones sobre este EER:

1. ¿El presidente de una mesa electoral es, obligatoriamente, votante en esa misma mesa?
2. ¿Todas las mesas tienen presidente y vocales?
3. ¿Todo miembro de una mesa tiene suplente?
4. ¿Cuántos presidentes hay como máximo en una mesa electoral?
5. ¿Conocemos a los candidatos a las elecciones autonómicas?



IX5. deficiencias del modelo

El modelo Entidad_Relación, tal y como lo definió Chen en su momento, describe de una manera muy comprensible la parte estática de un sistema de información aunque, eso sí, no es capaz de captar todas las peculiaridades de un sistema de información complejo. Sin embargo, sigue sin poder representar la dinámica, el conjunto de operaciones y transacciones que afectan a la evolución de los datos a almacenar.

Tampoco dispone de SGBD comerciales que permitan la explotación de un esquema lógico descrito bajo este modelo. No obstante, sí es ampliamente utilizado en la fase de diseño conceptual, que posteriormente es traducido a modelo relacional.

BIBLIOGRAFÍA

[CELMA97]