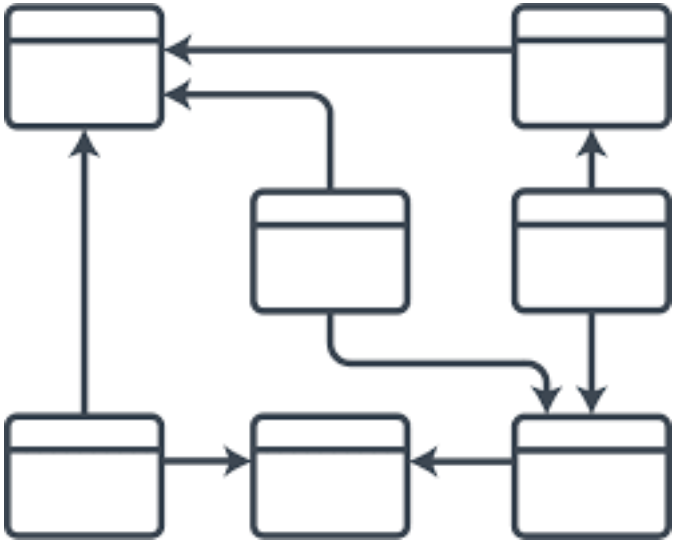
Introducción al Modelado de datos



JORGE DOMÍNGUEZ CHÁVEZ

IEASS, Editores

ORCID: 0000-0002-5018-3242

ISBN 978-980660200-7

Publicado por IEASS, Editores
ieass.editores@gmail.com
Venezuela, 2018

La portada y contenido de este libro fue editado y maquetado en TeXstudio 2.12.10

Índice general

Índice general

Abreviatura	Definición
ER	Entidad-Relación
DER	Diagrama Entidad-Relación
UML	Lenguaje unificado de modelado
DSD	Diagramas de estructura de datos
DFD	Diagramas de flujo de datos
MD	Modelos de Datos
SGBD	Sistema gestor de base de datos
MBD	Modelo de base de datos
LDE	Lenguaje de Definición de Esquemas
LMD	Lenguaje de Manipulación de Datos
MER	Modelo entidad-relación
MRO	Modelo relacional de objetos
SQL	Lenguaje Estructurado de consultas
CMS	Sistema Gestor de Contenidos

Capítulo 1

Introducción

En este libro, conoceremos y comprenderemos las entidades y las relaciones en las diferentes base de datos, comprensión que conduce al modelado que es la parte más importante del proceso de diseño de la base de datos jerárquica, de red, relacional o bien orientada a objetos.

El diseño de una base de datos no es un proceso sencillo. La complejidad de la información y la cantidad de requisitos de los sistemas de información hacen que sea complicado. Por este motivo, cuando se diseñan bases de datos es conveniente aplicar la conocida estrategia de dividir para vencer.

Por lo tanto, conviene descomponer el proceso del diseño en varias etapas; en cada una se obtiene un resultado intermedio que sirve de partida para la etapa siguiente, y en la última etapa se obtiene el resultado requerido. De este modo no hace falta resolver toda la problemática que plantea el diseño, sino que en cada etapa se afronta un sólo tipo de subproblema. Así se divide el problema y, al mismo tiempo, se simplifica el proceso.

La mayoría de los métodos de modelado de datos proporcionan algún método de mostrar gráficamente las entidades y relaciones.

Aprenderemos que existen varios estilos y diferentes de diagramas Entidad-Relación (**ER**). Al final de este estudio, determinaremos nuestro estilo preferido, así que utilícelo.

Ch. Bachman y A.P.G. Brown, en los años 60 y 70, trabajaron con los primeros artículos de Chen. Bachman desarrolló un tipo de diagrama de estructura de datos que lleva su nombre **diagrama de Bachman**. Brown publicó

sobre el modelado de los sistemas del mundo real. James Martin mejoró el Diagrama Entidad-Relación (**DER**). Chen, Bachman, Brown, Martin y otros contribuyeron al desarrollo del lenguaje unificado de modelado (**UML**), ampliamente utilizado en el diseño de software.

Los diagramas E-R que estudiamos sirven para los siguientes objetivos:

- Modelar las necesidades de información de una organización.
- Identificar entidades y sus relaciones.
- Proporcionar un punto de partida para la definición de datos (diagramas de flujo de datos).
- Proporcionar una excelente fuente de información para los programadores de aplicaciones, así como para los administradores de bases de datos y de sistemas.
- Crear un diseño lógico de la base de datos que se puede convertir en un esquema físico.

Los diagramas ER se relacionan con los diagramas de estructura de datos (**DSD**), que se centran en las relaciones de los elementos dentro de las entidades, en lugar de las relaciones entre las entidades mismas. Los diagramas ER a menudo se combinan con los diagramas de flujo de datos (**DFD**), que trazan el flujo de la información para procesos o sistemas.

El diseño de bases de datos es el proceso que determina su organización y estructura. Este diseño se realiza, generalmente, en tres pasos:

- Diseño conceptual: obtenemos una estructura de la información de la futura BD independiente de la tecnología a emplear. Aún no sabemos qué tipo de base de datos utilizaremos –relacional, orientada a objetos, jerárquica, etc.–; tampoco con qué SGBD ni con qué lenguaje concreto implementaremos la base de datos. Así pues, el diseño conceptual nos permite concentrarnos únicamente en la problemática de la estructuración de la información, sin tener que preocuparnos de resolver, en paralelo, aspectos tecnológicos.

  El resultado del diseño conceptual se expresa mediante algún modelo de datos de alto nivel. Uno de los más empleados es el modelo entidad-relación, que abreviaremos con **ER**.

- Diseño Lógico: partimos del diseño conceptual que se adaptará a la tecnología a emplear. Más concretamente, es preciso que se ajuste al modelo del SGBD con el que implementaremos la base de datos. Si se trata de un SGBD relacional, obtendremos un conjunto de relaciones con sus atributos, claves primarias y claves foráneas.

  Partimos del hecho de que hemos resuelto la problemática de la estructuración de la información en un ámbito conceptual, y nos concentraremos en las cuestiones tecnológicas relacionadas con el modelo de base de datos.

- Diseño Físico: transformamos la estructura obtenida en el diseño lógico con el objetivo de conseguir una mayor eficiencia; además, lo completamos con aspectos de implementación física que dependerán del SGBD.

  Si se trata de una base de datos relacional, la transformación de la estructura consiste en: tener almacenada alguna relación que sea la combinación de varias relaciones obtenidas en el diseño lógico, dividir una relación en varias, añadir algún atributo calculable a una relación, etc. Los aspectos de implementación física a completar consisten normalmente en la elección de estructuras físicas de implementación de las relaciones, la selección del tamaño de las memorias intermedias (buffers) o de las páginas, etc.

La fase inicial del diseño de la base de datos física es la traducción del modelo de datos lógico global a una forma que puede ser implementada.

La siguiente fase, diseña el archivo de organización y métodos de acceso que puedan usarse para almacenar, analizar las transacciones que se ejecutarán en la base de datos, la elección conveniente de archivos de organización basados en estos análisis, añadir índices primarios y secundarios, introducir y controlar la redundancia para mejorar el funcionamiento y, finalmente, estimar el espacio del disco necesario para su implementación.

El objetivo del diseño físico es encontrar un esquema interno que soporte la estructura conceptual y los objetivos del diseño lógico con la máxima eficiencia de los recursos suministrados por la máquina.

La figura 1.1 resume el material a cubrir por el diseño de una base de datos.

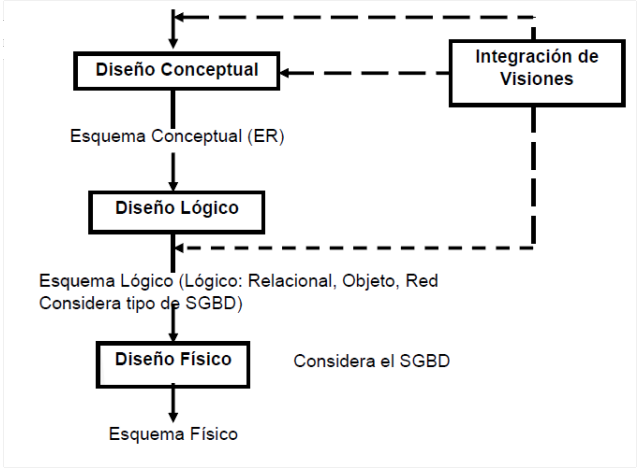


Figura 1.1: Diseño de una base de datos

1.1. Modelos de Datos y Esquemas

- Los modelos de datos **MD** son herramientas para describir la realidad.
- Los programadores los utilizan para construir esquemas, que son representaciones de la realidad.
- La calidad de un esquema se relaciona con las propiedades del MD y la experiencia de los programadores.

1.2. Representación de los Modelos de Datos

- Los MD tienen asociados una serie de conceptos, que describen un conjunto de datos y operaciones para manipular los datos.
- Dichos conceptos tienen asociados una construcción lingüística y otra gráfica.

1.3. Clases de Modelos de Datos

Existen dos tipos de MD:

- Modelo Conceptual, que representa la realidad en un alto nivel de abstracción.
- Modelo Lógico o Modelo de Base de Datos, que describe las relaciones lógicas entre los datos y la base de datos.

El primero genera el esquema conceptual y el segundo, el esquema lógico.

1.4. Características

- Los elementos de un modelo representan **Entidades** genéricas.
- Los valores concretos se denominan **Instancias u Ocurrencias** de una entidad.
- Cada sistema gestor de base de datos **SGBD** se asocia a un modelo de base de datos **MBD** específico, aunque existen excepciones.

1.5. Contenidos

Por lo que respecta a los datos:

- Datos o Entidades.
- Propiedades de los Datos.
- Relaciones de los Datos.
- Restricciones de los Datos.

Se representan mediante el Lenguaje de Definición de Esquemas **LDE** del SGBD.

Por lo que se refiere a las operaciones.

- Operaciones de los Datos.
- Operaciones sobre Relaciones de los Datos.
- Relaciones entre Operaciones.

Se representan mediante el Lenguaje de Manipulación de Datos **LMD** del SGBD.

1.6. Sistemas Prerrelacionales

1.6.1. Modelo Jerárquico

Los datos se relacionan de modo jerárquico y se representan mediante una estructura en árbol.

1.6.2. Modelo de Red

Entendemos como una generalización del modelo jerárquico cuando los nodos hijo pueden tener varios nodos padre.

1.6.3. Modelo Relacional

Utilizamos conceptos matemáticos, como las relaciones, para representar los datos y las operaciones sobre ellos.

1.7. Sistemas Postrrelacionales

1.7.1. Modelo Orientado a Objetos

Los datos se representan mediante objetos, que contienen variables y métodos, y su manipulación se realiza mediante mensajes.

Capítulo 2

Datos, información y conocimiento

Una base de datos o banco de datos es un conjunto de datos, que pertenecen al mismo contexto y que están almacenados sistemáticamente para su uso posterior. El insumo principal son los datos y el resultado de su procesamiento es la información.

En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.

En la actualidad, y debido al desarrollo tecnológico de campos como la informática y electrónica, la mayoría de las bases de datos tienen formato electrónico digital, lo que ofrece un amplio rango de soluciones al problema del almacenamiento y consulta de los datos.

Se habla de formato digital porque la información, no importa su forma: texto, sonido, imagen fija o en movimiento, se registra en un medio electrónico. Un diskette, Cd, DVD, BlueRay, pen-drive, en un CD Rom, en un disco duro (interno o externo) o en una cinta o bien están almacenados en la 'NUBE' todo lo cual constituye un 'Documento Digital'; recibe su nombre por la manera como que se registra.

Son documentos electrónicos, los archivos de 'procesadores de palabras', 'hojas de cálculo', 'administradores de bases de datos', o programas para elaborar gráficos.

2.1. Dato

Vivimos en un mundo formado por objetos, una persona es un objeto, al igual que los animales, plantas, rocas, entre otros, con objetos.

Un objeto tiene propiedades físicas, las cuales son medibles y definen el estado de un sistema físico. Los cambios en las propiedades físicas de un sistema describen sus transformaciones y su evolución entre estados temporales.

Las propiedades físicas son matemáticamente representadas por funciones de magnitudes físicas. Su interacción con objetos –y sus propiedades-, bajo leyes físicas establecidas, predice comportamientos y descubre propiedades no observadas. Como lo son: la masa, la velocidad, el tiempo, la energía.

Mencionamos que un objeto es una persona, por lo que sus propiedades físicas son un conjunto básico de hechos referentes a ella como: tamaño, estatura, peso, sexo, descripción, volumen, tasa, nombre, monto, edad o lugar. Según Murdick.

Las propiedades físicas son sinónimos de atributos o magnitudes y, éstos lo son de datos.

Un dato es un conjunto discreto, de factores objetivos sobre un hecho real. Un dato no dice nada sobre el porqué de las cosas, y por sí mismo tiene poca o ninguna relevancia o propósito.

En informática, los datos son representaciones simbólicas (numéricas, alfabéticas, algorítmicas, etc.) de un determinado atributo o variable cualitativa o cuantitativa, es: la descripción codificada de un hecho empírico, un suceso, una entidad.

Según O'Brien, los datos son útiles después de un proceso de valor añadido, como:

1. Su forma es agregada, manipulada y organizada,
2. Su contenido es analizado y evaluado,
3. Es puesto en un contexto para el usuario humano.

Todos los datos son generados por una de las cuatro escalas de medición: nominal, ordinal, de intervalo o de razón:

1. Nominal.

2. Ordinal,
3. de Intervalo.
4. de Razón.

Una escala de medición es nominal si los datos son etiquetas o categorías que se usan para definir un atributo de un elemento. Los datos nominales son o no numéricos.

El sexo de una persona es un dato nominal no numérico. El número de seguro social de una persona es un dato nominal numérico.

Una escala de medición es ordinal si los datos se usan para jerarquizar u ordenar las observaciones. Los datos ordinales son o no numéricos.

Las medidas pequeño, mediano y grande para el tamaño de un objeto son datos ordinales no numéricos. Una escala de medición es de intervalo si los datos tienen las propiedades de los datos ordinales y los intervalos entre observaciones se expresan en términos de una unidad de medición fija. Los datos de intervalo tienen que ser numéricos.

Las mediciones de temperatura son datos de intervalo. Si la temperatura en un lugar es 21 ° C y en otro es 4 ° C. Estos lugares se jerarquizan de acuerdo con lo caluroso que son: el primero es más caliente que el segundo. La unidad fija de medición, 1 ° C , dice cuán más caliente es el primer lugar: 17 ° C . Una escala de medición es de razón si los datos tienen las propiedades de los datos de intervalo y el cociente (o razón) entre dos medidas tiene sentido. Los datos de razón tienen que ser numéricos.

Variables como distancia, altura, peso y tiempo se miden con una escala de razón.

Según Burch, los datos son hechos aislados y en bruto son el elemento principal de la información.

Los datos suelen ser magnitudes numéricas directamente medidas o captadas, pero también pueden ser nombres o conjunto de símbolos; o valores cualitativos; o frases enteras, principio filosóficos; o imágenes, sonidos, colores, olores.

Los datos no son información más que en un sentido amplio de **información de partida** o **inicial**, por si solos no permiten la adopción de la decisión más conveniente porque no aportan los conocimientos necesarios. Sólo una

elaboración adecuada de los datos (un proceso de los datos) proporciona el conocimiento adecuado.

2.2. Tipos de datos

En la informática, cuando hablamos de tipo de dato nos referimos a un atributo que indica al computador la naturaleza de los datos a procesar. Esto delimita o restringe los datos, define los valores que pueden tomar, qué operaciones se realiza con ellos, etc.

Algunos tipos de datos son:

- Caracteres. Dígitos individuales que se representan mediante datos numéricos (0-9), letras (a-z) u otros símbolos.
- Caracteres unicode. Estándar de codificación que representa cualquier dato, hasta 65535 caracteres diferentes.
- Numéricos. Números reales o enteros, dependiendo de lo necesario.
- Booleanos. Valores lógicos (verdadero o falso).
- Fechas.

2.3. Ejemplos de datos

Los datos informáticos tiene su jerarquía, la cual es una pirámide ascendente de lo básico a lo complejo:

- Bits. Cada entrada del lenguaje de código binario, 1 o 0.
- Caracteres. Números, letras o caracteres especiales, formados cada uno según una combinación de bits. El número decimal 99 corresponde a 1100011 en binario.
- Campos. Conjunto ordenado de caracteres, como una palabra, como el nombre y/o el apellido del usuario que llena un formulario en línea.
- Registros. Conjuntos de campos ordenados, como para iniciar sesión en nuestro correo electrónico.

- Archivos. Conjuntos ordenados de registros, como las cookies[1] que las páginas web guardan en nuestro sistema y contienen la información de las sesiones iniciadas.
- Base de datos.

2.4. Datos e información

Los datos son información codificada, lista para ser introducida y procesada por un computador. Es decir, los datos no son más que una forma de representar información.

Como resultado del procesamiento de datos tenemos información. Una vez que han sido procesados dichos datos, se muestran de modo inteligible.

Los datos son la información (valores o referentes) que recibe el computador a través de distintos medios, y que es manipulada mediante el procesamiento de programas. Los datos pueden ser: estadísticas, números, descriptores, que por separado no tienen relevancia para los usuarios del sistema, pero que en conjunto pueden ser interpretados para obtener una información completa y específica.

En los lenguajes de programación, crear y organizar los algoritmos que todo sistema informático o computacional requiere, los datos son las características puntuales de las entidades sobre las cuales operan dichos algoritmos. Es decir, son la entrada inicial, para procesar y componer la información.

Los datos son importantes para la rama de la computación denominada éstructura de datos'que estudia la forma particular de almacenamiento de la información en porciones mínimas para lograr una posterior recuperación eficiente.

En informática se entiende por información al conjunto de datos ordenados, secuenciados o procesados por un algoritmo de programación, que recompone un referente, como un hecho concreto o algún sentido real.

La recuperación de la información a partir de los paquetes o conjuntos de datos es el objetivo final de la computación, dado que los sistemas informáticos

[1] Archivo creado por un sitio web que contiene pequeñas cantidades de datos y que se envían entre un emisor y un receptor.

codifican y representan la información a través de distintos mecanismos y lenguajes para comunicarse entre sí de manera eficaz y eficiente.

En informática, información es un termino muy usado y valorado, debido a que ella se introduce en las computadoras en forma de datos y los mismos son manipulados para que generan distintas soluciones a diferentes problemas.

Un dato es, en general, una expresión que indica las cualidades de los diferentes comandos sobre los que un algoritmo trabaja.

2.5. Cualidades de la información

Para que una información sea útil es necesario que facilite, al responsable de la toma de decisiones, una idea clara y completa de la situación, en forma tal que sus decisiones tengan el fundamento objetivo óptimo posible.

Las cualidades de una buena información son:

- Precisión: la información debe ser precisa, se mide en nivel de detalle y discernimiento. **Se han vendido 39 artículos** es más preciso que decir vendimos **varias docenas de artículos**. La precisión a exigir depende la aplicación. La falta de precisión es un defecto como un exceso de ella. Decir a un carpintero que la altura de una mesa es de 81.473245 cm., es un exceso de precisión.

- Exactitud: La información debe ser exacta. Para ello se mide en términos de porcentaje de error. Es una medida de alejamiento de la realidad. No obtenemos exactitud suficiente partiendo de datos incorrectos o erróneos, podremos corregir algunos pequeños errores o aplicar filtros a datos no válidos.

- Oportunidad: La información tiene que ser oportuna. Debe estar disponible para el usuario en el momento, en el lugar indicado y en la calidad justa para resolver un problema. El usuario debe actuar antes de que la realidad haya sufrido un cambio que anule la acción.

- Integridad: La información debe ser completa. Aún cuando la integridad al 100 % es inalcanzable en la mayoría de las aplicaciones, conviene en todo caso que la información sea tan completa como sea posible.

- Significatividad: La información debe ser clara y relevante. Es importante no forzar la comprensión del destinatario. Cualquier ayuda gráfica, visual, auditiva, o algún tipo que añada facilidad y rapidez a la recepción de la información debe ser considerada.

Recordemos que información es el resultado del procesamiento de datos.

En sentido general, la información es un conjunto organizado de datos procesados, que constituyen un mensaje sobre un determinado ente o fenómeno. De esta manera, si organizamos datos sobre un país como: número de habitantes, densidad de población, nombre del presidente, etc. y escribimos el capítulo de un libro, decimos que él contiene información sobre ese país.

Cuando tenemos que resolver un determinado problema o tomar una decisión, empleamos diversas fuentes de información (como el capítulo del libro imaginario) y construimos lo que, en general, se denomina conocimiento o información organizada para la resolución de problemas o la toma de decisiones.

2.6. Datos en la organización

Las organizaciones actuales almacenan datos mediante el uso de la tecnología de información y comunicación (**TIC**). Desde un punto de vista cuantitativo, evaluamos la gestión de los datos en términos de calidad, exactitud, costo, velocidad y capacidad.

Las organizaciones necesitan datos y algunas son totalmente dependientes de ellos.

Bancos, compañías de seguros, agencias gubernamentales, comercios y la Seguridad Social son casos obvios. En este tipo de organizaciones, la buena gestión de los datos es esencial para su funcionamiento, ya que operan con millones de transacciones diarias. Pero en general, para la mayoría de las empresas tener muchos datos no siempre es bueno. Las organizaciones almacenan datos sin sentido.

Esta actitud no tiene razón:

- Demasiados datos hacen complicado identificar aquellos que son relevantes.

- Los datos no tienen significado en sí mismos.

Los datos describen una parte de lo que pasa en la realidad y no proporcionan juicios de valor o interpretación y, por lo tanto, no son conducentes para la acción.

La toma de decisiones se basa en datos, pero nunca dirán que hacer. Los datos no dicen nada de lo que es o no importante. A pesar de todo, los datos son importantes para las organizaciones, ya que son la base para la creación de información.

2.7. Datos, información, conocimiento

Muchos investigadores que han estudiado el concepto de información la definen como un mensaje, en forma de documento o algún tipo de comunicación audible o visible. Como cualquier mensaje, tiene un emisor y un receptor. La información es capaz de cambiar la forma en que el receptor percibe algo e impacta sobre sus juicios de valor y comportamiento. El mensaje tiene que informar y sus datos marcan la diferencia.

La palabra **informar** significa **dar forma a** y la información es capaz de formar a una persona cambiando su comportamiento tanto interno como externo. Por tanto, estrictamente hablando, es el receptor, y no el emisor, quién decide si el mensaje recibido es información útil, es decir, si realmente le informa. Un informe lleno de datos inconexos, puede ser considerado información por el que lo escribe, pero a su vez es juzgado como **ruido** por el que lo recibe.

A diferencia de los datos, la información tiene significado (relevancia y propósito). No sólo forma al que la recibe, sino que está organizada para algún propósito. Los datos se convierten en información cuando su creador les añade de significado. Transformamos datos en información añadiéndoles valor en varios sentidos.

¿Cómo?

Contexto sabemos para qué propósito se generaron los datos.

Categoría conocemos las unidades de análisis de los componentes principales de los datos.

Cálculo analizamos matemática o estadísticamente los datos.

Corrección eliminamos errores de los datos.

Condensado resumimos, en forma concisa, los datos.

Las computadoras añaden valor y transforman datos en información, pero es muy difícil que nos ayuden a analizar el contexto de dicha información. Un problema común es confundir información (o conocimiento) con la tecnología que la soporta. Desde la televisión a Internet, es importante tener en cuenta que el medio no es el mensaje. Lo que se intercambia es más importante que el medio usado para hacerlo. El tener un teléfono no garantiza mantener conversaciones brillantes. En definitiva, que tener acceso a las tecnologías de la información no implica que hayamos mejorado nuestro nivel de información.

2.8. Conocimiento

La mayoría de las personas tienen la sensación intuitiva que el conocimiento es amplio, profundo y rico, más que los datos y la información.

2.8.1. Definición de conocimiento

Para Davenport y Prusak el conocimiento es una mezcla de experiencia, valores, información y **saber hacer** que sirve como marco para la incorporación de nuevas experiencias e información y es útil para la acción. Se origina y aplica en la mente de los conocedores. En las organizaciones, con frecuencia, esto no sólo se encuentra dentro de documentos o almacenes de datos, sino que también está en rutinas organizativas, procesos, prácticas, y normas.

Lo que deja claro la definición anterior es que ese conocimiento no es simple. Es una mezcla de varios elementos; es un flujo al mismo tiempo que tiene una estructura formalizada; es intuitivo y difícil de captar en palabras o de entender plenamente de forma lógica. El conocimiento existe dentro de las personas, como parte de la complejidad humana y de nuestra impredecibilidad. Aunque pensamos en activos definibles y concretos, los activos de conocimiento son difíciles de manejar. El conocimiento es visto como un proceso (flujo) o como algo acumulado.

El conocimiento se deriva de la información, así como la información se deriva de los datos. Para que la información se convierta en conocimiento las personas deben hacer prácticamente todo el trabajo.

Esta transformación se produce gracias a:

- Comparación.
- Consecuencias.
- Conexiones.
- Conversación.

Estas actividades de creación de conocimiento tienen lugar dentro y entre personas. Al igual que encontramos datos en registros, e información en mensajes, obtenemos conocimiento de individuos, grupos de conocimiento, o incluso en rutinas organizativas.

Los datos indican condiciones o situaciones que por sí solos no aportan ninguna información importante. En base a la observación y a la experiencia que un dato puede tomar cierto valor instruccional. También se dice que los datos son atributos pertenecientes a cualquier ente, pues una utilidad muy significativa de los datos es que se emplean en estudios comparativos.

La información modifica nuestros conocimientos.

2.9. Procesamiento de datos

2.9.1. Informática

A lo largo de la historia, el hombre necesita transmitir y tratar información de forma continua. La humanidad no ha parado de crear máquinas y métodos para procesar información. Con este fin surge la Informática, como una ciencia encargada del estudio y desarrollo de estas máquinas y métodos.

La Informática nace de la idea de ayudar al hombre en aquellos trabajos rutinarios y repetitivos, generalmente de cálculo y de gestión, donde es frecuente la repetición de tareas. La idea es que una máquina los realiza mejor, aunque siempre bajo la supervisión del hombre.

El vocablo informática proviene del francés informatique, acuñado por el ingeniero Philippe Dreyfus para su empresa **Société d'Informatique Appliquée** en 1962. Procede de la contracción de las palabras **INFORMA**tion auto**MATIQUE** (información automática). En los países anglosajones se conoce como **Computer Science**.

Informática es el conjunto de conocimientos científicos y técnicas que hacen posible el tratamiento automático de la información por medio de computadores. También, es la ciencia que estudia el tratamiento automático y racional de la información.

Sistema. Conjunto de partes o elementos organizadas y relacionadas que interactúan entre sí para lograr un objetivo.

Entonces, un sistema informático como todo sistema, es el conjunto de partes interrelacionadas, hardware, software y elemento humano (personal y usuarios). Un sistema informático típico está formado por una computadora que usa dispositivos programables para capturar, almacenar y procesar datos.

Los sistemas informáticos realizan las siguientes tres tareas básicas:

- Entrada: captación de la información.
- Proceso: tratamiento de la información.
- Salida: transmisión de resultados.

Capítulo 3

Modelos y tipos de base de datos

El modelado de datos es uno de los elementos más importantes al iniciar el desarrollo de cualquier proyecto. Esta es la estructura, sobre la que realmente reside la verdadera esencia de la aplicación. Incluso determina si el proyecto va a cumplir con su verdadero objetivo.

3.1. Modelo de datos

El modelado de datos es una técnica independiente de la implementación a la base de datos. Esto es importante, porque la metodología L5[1], siempre busca obtener el máximo provecho de diversas herramientas. En particular, el esquema final y su implementación sufren cambios sin afectar de manera drástica la lógica de programación.

Uno de los puntos a resaltar es que el modelado de los datos debe ser llevado como una guía general. Para los profesionales expertos, esto implica el desarrollo de los diagramas de entidades y del modelo entidad-relación. Independientemente de la metodología a utilizar, esta herramienta siempre será importante, para entender las relaciones entre las diversas entidades en la base de datos. Ésta, es un conjunto integrado de datos que modelizan un universo dado, el cual está compuesto por objetos relacionados, los objetos

[1] Ver Anexo A

de un mismo tipo que constituyen una entidad y el vínculo entre entidades se denomina asociación.

La parte esencial de una estructura de base de datos es el modelo de datos: una colección de herramientas conceptuales para describir los datos, las relaciones entre ellos, la semántica y condiciones de consistencia.

Un modelo de base de datos muestra la estructura lógica de la base, incluidas las relaciones y limitaciones que determinan cómo se almacenan los datos y cómo se accede a ellos. Generalmente todo modelo tiene una representación gráfica, para el caso de datos el modelo más popular es el modelo entidad-relación o diagrama. Los modelos de bases de datos individuales se diseñan en base a las reglas y los conceptos de cualquier modelo de datos más amplio que los diseñadores adopten. La mayoría de los modelos de datos se representan por un diagrama de base de datos acompañante. Vea la figura 3.1.

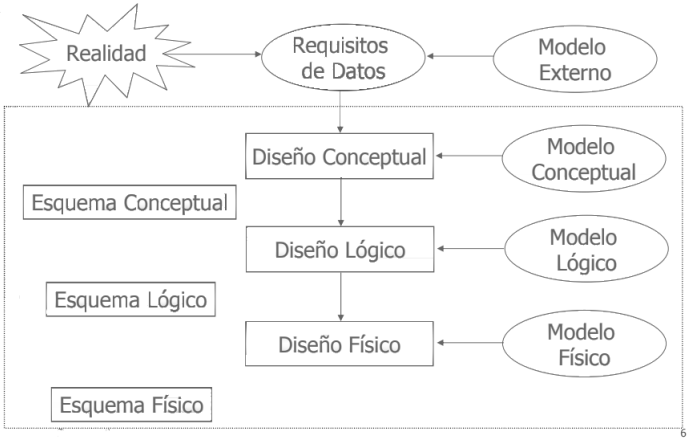


Figura 3.1: Definición de una base de datos

3.2. Importancia del modelo de base de datos

Está radica en que si tenemos buenos esquemas en los modelos a desarrollar, los datos tendrán una estructura que refleje de buena manera las entidades del mundo real. Y es el punto de partida para diseñar una base de datos haciendo

uso de técnicas que ayuden a realizarlo de una buena manera.

Otro aspecto importante en el modelado de datos durante el diseño de una base de datos es que tenemos una buena organización en la información, realizamos una búsqueda de los datos de manera ordenada, evitamos la duplicidad de información, verificamos que el dato tenga un significado, que toda la información sea oportuna y ayuda a que la base de datos no tenga errores o fallas.

Los modelos de datos definen con claridad cómo se modela la estructura lógica de una base de datos.

Los modelos, son las entidades necesarias para introducir la abstracción en un **SGBD**, entendiéndola como el proceso de aislar un elemento de su contexto o del resto de elementos que lo acompañan. Un modelo de base de datos incluye las relaciones y limitaciones que determinan cómo se almacenan los datos y accede a ellos.

El modelado de datos es el proceso de documentar un diseño de sistema de software complejo como un diagrama de fácil comprensión, usando texto y símbolos para representar la forma en que los datos necesitan fluir. El diagrama se utiliza como un mapa para la construcción de un nuevo software o para la reingeniería de una aplicación antigua.

Un modelo de datos es una **descripción** de algo conocido como contenedor de datos (donde se guarda la información), así como los métodos para almacenar y recuperar información de esos contenedores.

Los modelos de datos no son cosas físicas: son abstracciones para la implementación de un sistema eficiente de base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos.

3.3. Tipos de modelo de base de datos

El proceso de descripción de asociaciones y entidades se llama modelización y se hace con un modelo de datos.

A continuación una lista de algunos modelos de base de datos:

- Modelo jerárquico.
- Modelo en red.

- Modelo de base de datos orientado a objetos.
- Modelo relacional.
- Modelo transaccional.

Además, existen el modelo entidad-relación **MER**, el entidad-atributo-valor, el esquema en estrella, el modelo de documento y el modelo relacional de objetos **MRO**.

3.4. Componentes básicos

3.4.1. Entidades

En bases de datos, una entidad es la representación de un objeto con existencia independiente o concepto del mundo real que se describe en una base de datos.

Tipo de entidad Persona, organización, tipo de objeto o concepto sobre los que se almacena información. Describe el tipo de la información que se está controlando. Normalmente un tipo de entidad corresponde a una o varias tablas relacionadas en la base de datos.

3.4.2. Atributos

Los atributos son las características que definen o identifican a una entidad. Estas pueden ser muchas, y el diseñador sólo utiliza o implementa las que considere más relevantes.

3.4.3. Relaciones

Es un vínculo que define una dependencia entre varias entidades, es decir, exige que compartan ciertos atributos de forma indispensable.

3.4.4. Cardinalidad

La cardinalidad se representa en un diagrama ER como una etiqueta que se ubica en ambos extremos de la línea de relación de las entidades y que contiene diversos valores entre los que destacan el 1 (uno) y el * (asterico).

3.4.5. Claves

Es el campo o atributo de una entidad o tabla que distingue cada registro del conjunto, sirviendo sus valores como datos vinculantes de una relación entre registros de varias tablas.

3.5. Diferencias entre el modelo de datos y el modelado de datos

3.5.1. Modelo de datos

El resultado de un modelado de datos es una representación que tiene dos componentes: las propiedades estáticas que se definen en un esquema y las propiedades dinámicas que se definen como especificaciones de transacciones, consultas e informes.

Un esquema consiste en una definición de todos los tipos de objetos de la aplicación, incluyendo sus atributos, relaciones y restricciones estáticas.

Correspondientemente, existe un repositorio de información, la base de datos, que es una instancia del esquema. Un determinado tipo de procesos sólo necesita acceder a un subconjunto predeterminado de entidades definidas en un esquema, por lo que este tipo de procesos requiere sólo un subconjunto de las propiedades estáticas del esquema general.

3.5.2. Modelado de datos

Entendemos al modelado de datos como el proceso de documentar un diseño de sistema de software complejo como un diagrama de fácil comprensión, usando texto y símbolos para representar la forma en que los datos necesitan fluir.

El diagrama se utiliza como un mapa o plano para la construcción de un nuevo software o para la reingeniería de una aplicación antigua.

Los modeladores de datos utilizan varios modelos para ver los mismos datos y garantizar que todos los procesos, entidades, relaciones y flujos de datos han sido identificados. Hay varios enfoques diferentes para el modelado de datos, incluyendo:

Modelado conceptual de datos

Identifica las relaciones de más alto nivel entre diferentes entidades.

Modelado de datos empresariales

Similar al modelado de datos conceptuales, pero se dirige a los requisitos únicos de un negocio específico.

Modelado lógico de datos

Ilustra las entidades, atributos y relaciones específicas que participan en una función de negocios. Sirve como soporte a la creación del modelo de datos físico.

Modelado de datos físicos

Representa una aplicación e implementación específica de base de datos de un modelo de datos lógicos.

3.6. Hitos en la evolución de los modelos de datos

A partir de los años 60 en muchas empresas no existían mecanismos para gestionar la información de manera organizada; dentro de este período surgen ciertos lenguajes y técnicas para solucionar este problema, éste fue el caso de los SGBD que registran la información mediante campos, tipos de datos, basándose en computadoras.

El término base de datos fue escuchado por primera vez en un simposio celebrado en California en 1963.

Los orígenes de las bases de datos se remontan a la Antigüedad donde existían bibliotecas y toda clase de registros. Además, también se utilizaban para recoger información sobre las cosechas y censos.

En los años 70 aparecen las primeras bases de datos relacionales: – Ingres – System R.

En la época de los 80 también se desarrolla el Lenguaje Estructurado de consultas **SQL** o lo que es lo mismo un lenguaje de consultas o lenguaje decla-

rativo de acceso a bases de datos relacionales que efectúa consultas con el objetivo de recuperar información de interés de una base de datos.

A principios de los años 80 comenzó el auge de la comercialización de los sistemas relacionales, y SQL comenzó a ser el estándar de la industria, porque las bases de datos relacionales están formadas por un sistema de tablas (compuesta por filas y columnas).

En la década de 1990, la investigación en bases de datos giró en torno a las bases de datos orientadas a objetos. Así se desarrollaron herramientas como Excel y Access de Microsoft Office que marcan el inicio de las bases de datos orientadas a objetos.

En la actualidad, las tres grandes compañías que dominan el mercado de las bases de datos son IBM, Microsoft y Oracle. Como base de datos libres destacan: MySQL, MariaDB, PostgreSQL y SQLite3.

Capítulo 4

Proceso de Diseño

El diseño de bases de datos es el proceso que determina la organización de una base de datos, incluidos su estructura, contenido y las aplicaciones que se han de desarrollar. Se ha progresado mucho en el diseño de bases de datos y éste se considera ahora una disciplina estable, con métodos y técnicas propios.

Debido a la creciente aceptación por parte de la industria y el gobierno en el plano comercial, y a una variedad de aplicaciones científicas y técnicas, el diseño de bases de datos desempeña un papel central en el empleo de los recursos de información en la mayoría de las organizaciones. Ver la figura 4.1.

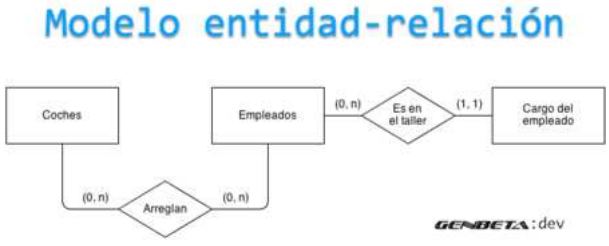


Figura 4.1: Un modelo entidad-relación o diagrama entidad-relación

Es importante tener en cuenta que los términos **base de datos** y **tabla** no son sinónimos. El término base de datos se refiere a una base de datos relacional

que almacena información sobre una o más tablas.

La clave para obtener un diseño de base de datos eficaz radica en comprender exactamente qué información se desea almacenar y la forma en que un sistema de administración de bases de datos relacionales, almacena los datos. Para ofrecer información de forma eficiente y precisa, debe tener almacenados los datos sobre distintos temas en tablas separadas. Al organizar los datos de forma apropiada, proporciona flexibilidad a la base de datos y tiene la posibilidad de combinar y presentar información de muchas formas diferentes.

El proceso de diseño consta de:

- Determinar la finalidad de la base de datos: nos prepara para los demás pasos.
- Buscar y organizar la información necesaria: Reunir todos los tipos de información a registrar.
- Dividir la información en tablas: Divide los elementos en entidades o temas principales.
- Convertir los elementos de información en columnas: Decidir qué información requiere almacenar en cada tabla. Convertir cada elemento en un campo y se mostrará como una columna en la tabla.
- No incluir datos calculados.
- Almacenar la información en sus partes lógicas más pequeñas
- Especificar claves principales: La clave principal es una columna que se utiliza para identificar inequívocamente cada fila, como Id. de producto o Id. de pedido o código.
- Definir relaciones entre las tablas: Examinar cada tabla y decidir cómo se relacionan los datos de una tabla con las demás tablas.
- Ajustar el diseño: Analizar para detectar errores.
- Aplicar las reglas de normalización: Comprobar si las tablas están correctamente estructuradas.

4.1. Modelo Entidad – Relación

Un diagrama entidad-relación, también conocido como modelo entidad relación o ERD, es un tipo de diagrama de flujo que ilustra cómo las **entidades**, como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema.

Los diagramas de ER se relacionan con los diagramas de estructura de datos (DSD), que se centran en las relaciones de los elementos dentro de las entidades, en lugar de las relaciones entre las entidades mismas. Los diagramas ER a menudo se combinan con los diagramas de flujo de datos (DFD), que trazan el flujo de la información para procesos o sistemas.

Un modelo de datos es una colección de herramientas conceptuales para la descripción de datos, relaciones entre datos, semántica de los datos y restricciones de consistencia. Podemos citar dos modelos de datos —el modelo entidad-relación y el modelo relacional.

El modelo entidad-relación es un modelo de datos de alto nivel. Está basado en una percepción de un mundo real que consiste en una colección de objetos básicos, denominados entidades, y de relaciones entre estos objetos.

El modelo relacional es un modelo de menor nivel. Usa una colección de tablas para representar tanto los datos como las relaciones entre los datos.

Hay tres nociones básicas que emplea el modelo de datos E-R: conjuntos de entidades, de relaciones y de atributos.

Una entidad es una **cosa u objeto** en el mundo real que es distinguible de todos los demás objetos. Un conjunto de entidades es un conjunto de entidades del mismo tipo que comparten las mismas propiedades, o atributos.

Una entidad se representa mediante un conjunto de atributos. Éstos, describen propiedades que posee cada miembro de un conjunto de entidades.

Para cada atributo hay un conjunto de valores permitidos, llamado dominio, de ese atributo. El dominio del atributo nombre cliente sería el conjunto de todas las cadenas de texto de cierta longitud.

Habrá ocasiones en las que un atributo tiene un conjunto de valores para una entidad específica. Como una entidad cliente con el atributo número-teléfono. Cualquier empleado particular puede tener uno o más números de teléfono, o ninguno. Este tipo de atributo se llama multivalorado.

También están los atributos derivados que son otra cosa más que valores a ser calculados.

Un atributo toma un valor nulo cuando una entidad no tiene un valor para un atributo o también puede indicar **no aplicable**, es decir, que el valor no existe para la entidad.

4.2. Restricciones

Una restricción es una condición que obliga el cumplimiento de ciertas condiciones en la base de datos. A veces, algunas restricciones no son creadas por los usuarios sino por que la base de datos es relacional y las restricciones proveen un método de implementar reglas en la base de datos además juegan el rol de organizar mejor los datos.

La correspondencia de cardinalidades, o razón de cardinalidad, expresa el número de entidades a las que otra entidad puede estar asociada vía un conjunto de relaciones. Y estas son:

- Uno a uno. Una entidad en A se asocia con sólo una entidad en B, y una entidad en B se asocia sólo una entidad en A.
- Uno a varios. Una entidad en A se asocia con cualquier número de entidades en B. Una entidad en B, sin embargo, se asocia con sólo una entidad en A. Varios a uno. Una entidad en A se asocia solo una entidad en B. Una entidad en B, sin embargo, se asocia con cualquier número de entidades en A.
- Varios a varios. Una entidad en A se asocia con cualquier número de entidades en B, y una entidad en B se asocia con cualquier número de entidades en A.
- Restricciones Parciales: La participación de un conjunto de entidades E en un conjunto de relaciones R se dice que es total si cada entidad en E participa al menos en una relación en R. Si sólo algunas entidades en E participan en relaciones en R, la participación del conjunto de entidades E en la relación R se llama parcial.

4.3. Diseño con Diagramas E-R

La estructura lógica de una Base de Datos se puede representar gráficamente a través de un diagrama, el cual llamaremos Diagrama E-R. Estos diagramas se apoyan de diferentes símbolos los cuales tienen un significado particular. Los diagramas se usan para que la información se presente de forma clara y sencilla. Los componentes principales son entidad o conjuntos de entidades, relación o conjunto de relaciones, atributos y estos pueden clasificarse en:

- Simples o atómicos: Son aquellos que no contienen otros atributos
- Compuestos: Son los que incluyen otros atributos simples.. Ejemplo: dirección (Se puede dividir en calle, número, ciudad). Monovalorados o Univalorados: Atributo que toma un sólo valor para una entidad en particular.
- Multivalorado: Atributo que para una misma entidad puede tomar muchos valores.
- Derivados o calculados: Atributos cuyos valores se pueden conseguir con operaciones sobre valores de otros atributos.
- Nulos: Atributos para los cuales, en algún momento, no existe o no se conoce su valor.

El Diagrama Entidad – Relación es la representación gráfica del Modelo Entidad-Relación y permite ilustrar la estructura de la base de datos del negocio modelado.
Está compuesto por los siguientes elementos.

- Rectángulos: representan conjuntos de entidades.
- Elipses: representan atributos.
- Rombos: representan relaciones.
- Líneas: unen atributos a conjuntos de entidades y conjuntos de entidades a conjuntos de relaciones.
- Elipses dobles: representan atributos multivalorados.

- Elipses discontinuas: que denotan atributos derivados.
- Líneas dobles: indican participación total de una entidad en un conjunto de relaciones.
- Rectángulos dobles: representan conjuntos de entidades débiles.

4.4. Conjunto de entidades débiles

Las entidades que tienen un conjunto de atributos que forman sus claves primarias y que permiten identificarlas completamente se denominan, entidades fuertes.

Aunque también existen las entidades débiles cuyos atributos no la identifican completamente, sino que sólo la identifican de forma parcial. Esta entidad debe participar en una interrelación que ayuda a identificarla. Una entidad débil se representa con un rectángulo doble, y la interrelación que ayuda a identificarla se representa con una doble línea. Para que un conjunto de entidades débiles tenga sentido, debe estar asociada con otro conjunto de entidades, denominado el conjunto de entidades identificadoras o propietarias.

4.5. Modelo E-R extendido

La generalización refleja el hecho de que hay una entidad general, denominada **superclase** la cual se puede especializar en entidades subclase:

- a) La superclase modeliza las características comunes de la entidad vista de una forma genérica.
- b) La subclase modeliza las características propias de sus especializaciones.

4.6. Otros aspectos del diseño de bases de datos

Un diagrama E-R, puede ser representado también a través de una colección de tablas. Para cada una de las entidades y relaciones existe una tabla única

a la que se le asigna como nombre el del conjunto de entidades y de las relaciones respectivamente, cada tabla tiene un número de columnas que son definidas por la cantidad de atributos y las cuales tienen el nombre del atributo. Ver la figura 4.2

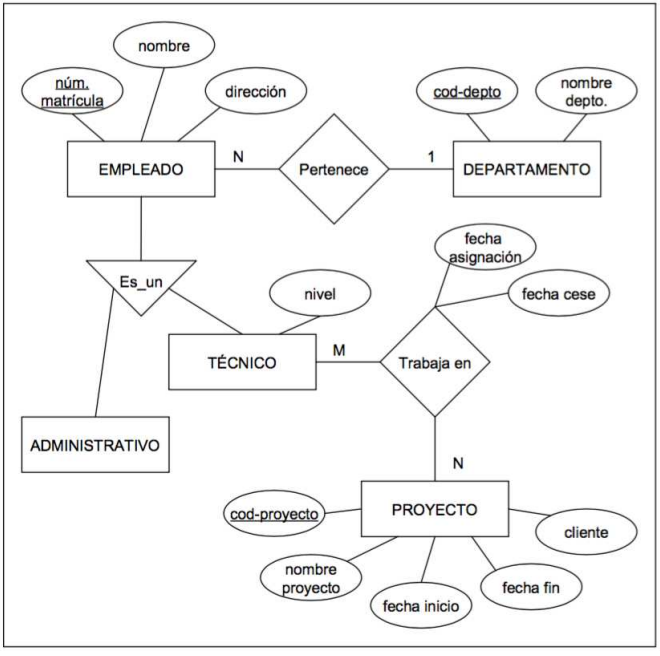


Figura 4.2: El modelo entidad/relación extendido describe con un alto nivel de abstracción la distribución de datos almacenados en un sistema.

4.7. La Notación E-R con UML

UML es un estándar propuesto para la creación de especificaciones de varios componentes de un sistema software. Se representa mediante diagramas, algunos de los cuales son:

- Diagrama de clase. Es similar a un diagrama E-R.
- Diagrama de caso de uso. Muestran la interacción entre los usuarios y el sistema, en particular los pasos de las tareas que realiza el usuario

- Diagrama de actividad. Describen el flujo de tareas entre varios componentes de un sistema.
- Diagrama de implementación. Muestran los componentes del sistema y sus interconexiones tanto en el nivel del componente software como el hardware.

UML muestra los conjuntos de entidades como cuadros y, a diferencia del E-R, muestra los atributos dentro del cuadro en lugar de elipses separadas; además, que en UML las entidades son manejadas como objetos. Los conjuntos de relaciones binarias se representan en UML dibujando una línea que conecte los conjuntos de entidades. Ver la figura 4.3 El Lenguaje de Mode-

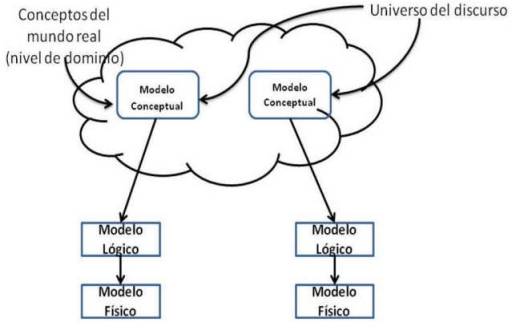


Figura 4.3: Modelos de data en diferentes niveles de abstracción.

lado Unificado (UML) es considerado hoy en día un método estándar para el desarrollo de sistemas de información. Sin embargo, el grado de fiabilidad y flexibilidad de los portales de gobierno electrónico, requieren el uso de principios básicos y del concurso y la experiencia de viejos estilos y disciplinas del desarrollo de sistemas, que sean independientes de los softwares de aplicación disponibles en el mercado. En ello, el modelo Entidad Relación (ER) es una herramienta fundamental para el modelado de trámites de e-gobierno porque son fácilmente transferidos a los diagramas de clases de UML. De manera que al modelar un sitio Web para gobierno electrónico con UML y ER, el sistema es dotado con diagramas gráficos semi-formales, que permiten el diseño de múltiples vistas basado en un modelado conceptual a partir de las instancias y requerimientos legales de la administración pública. Este tra-

bajo analiza el uso del Modelo Entidad Relación y los diagramas de clase de UML y propone, en forma práctica, las notaciones básicas que se requieren en el desarrollo de un portal de e-gobierno.

Capítulo 5

Modelo jerárquico

5.1. Introducción

Los **SGBD** jerárquicos fueron los primeros en aparecer. Una base de datos jerárquica se visualiza como una estructura en árbol.Una base de datos jerárquica consiste en una colección de segmentos (registro) que se conectan entre sí por medio de enlaces. Cada segmento es una colección de campos (atributos), que contienen un solo valor cada uno de ellos. Un enlace es una asociación o unión entre dos segmentos exclusivamente.

Las bases de datos jerárquicas son rígidas. Una vez diseñada, es complejo cambiarla y, además, es necesario un conocimiento amplio de la forma en la que se han almacenado los datos para recuperarlos de forma efectiva.

Por ello, a pesar de dominar el mercado en sus comienzos, estas bases de datos han ido decayendo y actualmente pocas de ellas se encuentran en el mercado.

- Una base de datos Jerárquica se compone de un conjunto ordenado de árboles, denominado bosque.
- También se define como el conjunto ordenado de instancias de un tipo de árbol.
- Un tipo de árbol se compone de un tipo de registro raíz junto con un conjunto ordenado de cero, uno o más subárboles dependientes.

- Un subárbol es el árbol de un nivel inferior.
- El orden global de la base de datos se obtiene mediante el recorrido en preorden del bosque.

Los SGBD jerárquicos son modelos lógicos basados en registros que describen datos en los niveles lógicos y de vista, la base de datos se estructura en registros de formato fijo de diferentes tipos, en cada tipo se define un numero fijo de campos o atributos, y cada campo tiene normalmente una longitud fija. Ésta, simplifica la implementación en el nivel físico de la base de datos. Esta simplicidad contrasta con muchos de los modelos basados en objetos.

Los árboles, como instrumentos para la representación de estructuras de datos, tienen problemas por su poca flexibilidad, lo que da origen a una falta de adaptación a muchas organizaciones reales. Además, no se ha llegado a una formalización matemática del modelo y de sus lenguajes, como ha ocurrido en el caso del modelo relacional; tampoco se ha intentado su estandarización, a pesar de ello los productos jerárquicos (IMS y DL/I de IBM como máximos exponentes de estos sistemas) consiguieron altas cuotas de mercado, pero la tecnología relacional los han llevado a convertirse en sistemas superados.

Sin embargo, todavía existen importantes aplicaciones soportadas de estos productos en operación, por su eficiente respuesta, a satisfacción de sus usuarios, seguridad por lo que las aplicaciones desarrolladas sobre ellos se mantienen sin apenas cambios.

Los productos basados en este tipo de modelos han perdido las altas cuotas de mercado de las que disfrutaban hace una década y se consideran sistemas superados por la tecnología relacional; aunque existen aplicaciones basadas en este modelo. Vea la figura 5.1.

5.2. Definiciones

- Los tipos de registros representan las entidades de la base de datos.
- Los arcos del árbol definen la relación existente entre las entidades.
- Los tipos de registros se descomponen en campos.

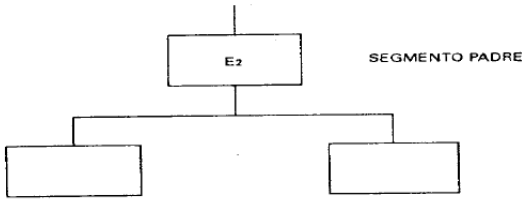


Figura 5.1: Aplicación típica base de datos jerárquica.

- Aparece una dependencia entre los niveles del árbol.
- Dicha dependencia lleva asociada la herencia de campos entre los niveles.
- Esa dependencia también tiene influencia en la integridad de los datos, ya que no existe un hijo sin su padre.

Veamos la figura **??**.

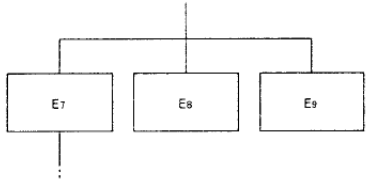


Figura 5.2: Segmentos hijos.

Definimos el modelo jerárquico como:

- Un conjunto de tipos de entidad E1, E2, ..., En.
- Un conjunto de interrelaciones o asociaciones no nominadas Rij que conectan los tipos de entidad Ei y Ej.
- Un conjunto de restricciones inherentes que provienen de la estructura jerárquica.

5.3. Conceptos básicos

La implementación del modelo Jerárquico se lleva a cabo en base a punteros; estructura física que varía según los productos, e incluso un mismo producto proporciona distintas organizaciones físicas a fin de que el usuario obtenga una mayor eficiencia en el diseño físico de la base de datos.

El modelo jerárquico fue desarrollado para representar aquellas situaciones de la vida real en las que predominan relaciones de tipo 1 : N.

Es un modelo rígido, donde las diferentes entidades de las que está compuesta una determinada situación, se organizan en niveles múltiples de acuerdo a una estricta relación PADRE/HIJO, de manera que un padre puede tener más de un hijo, todos ellos localizados en el mismo nivel, y un hijo únicamente puede tener un padre situado en el nivel inmediatamente superior al suyo.

Esta estricta relación PADRE/HIJO implica que no puedan establecerse relaciones entre segmentos dentro de un mismo nivel.

La representación gráfica de un modelo jerárquico se realiza mediante la estructura de ÁRBOL INVERTIDO, en la que el nivel superior está ocupado por una única entidad, bajo la cual se distribuyen el resto de las entidades en niveles que se van ramificando. Los diferentes niveles quedan unidos por medio de las relaciones, las entidades se denominan en el caso particular del modelo jerárquico SEGMENTOS, mientras que los atributos reciben el nombre de CAMPOS.

Los segmentos, se organizan en niveles de manera que en un mismo nivel estén todos aquellos segmentos que dependen de un segmento de nivel inmediatamente superior.

Los Segmentos se clasifican en tres tipos:

1. Padre: Es aquel que tiene descendientes(hijos) todos localizados al mismo nivel.

2. Hijo: Es aquel que depende de un segmento anterior, todos los hijos del mismo padre tendrán que estar localizados en el mismo nivel.

3. Segmento Raíz: Es el único segmento que no tiene padre ,es el antecesor de todos, y es el segmento de mayor nivel ,es decir esta en el nivel superior del Árbol.

5.4. Características

Es un tipo de Sistema Gestor que organiza la información en forma de árbol genealógico, en el que un nodo puede tener a su vez varios nodos que deriven de él.

Una base de datos jerárquica es un tipo de sistema de gestión de bases de datos que almacenan la información en una estructura jerárquica que enlaza los registros en forma de estructura de árbol tal que un nodo padre de información puede tener varios nodos hijo. De la misma manera, se puede establecer relación entre los nodos hermanos En este caso, la forma de árbol se convierte en una estructura en forma de grafo dirigido.

El modelo jerárquico se clasifica en estructuras lineales y arborescentes. La primera clase de estructura, cada tipo de registro padre sólo puede tener un tipo de registro hijo. La segunda, un tipo de registro padre puede tener varios tipos de registros hijos (El producto comercial en nuestros días es el IMS).

El modelo jerárquico facilita relaciones padre-hijo, es decir, 1:N (de uno a varios) del modelo relacional. Pero a diferencia de éste último, las relaciones son unidireccionales. Técnicamente, dichas relaciones son hijo-padre, pero no padre-hijo.

Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos. De la misma manera, otra limitación es que no garantiza la inexistencia de registros duplicados. Esto también es cierto para los campos “clave”. Es decir, no se garantiza que dos registros tengan diferentes valores en un subconjunto concreto de campos.

Las características principales de implementar este modelo son:

- Globalización de la información: los diferentes usuarios la consideran un recurso corporativo **sin** dueños específicos.
- Eliminación de información inconsistente: si existen dos o más archivos con la misma información, los cambios que se hagan a éstos deberán hacerse a todas las copias del archivo de facturas.
- Compartir información.
- Mantener la integridad en la información: cualidad altamente deseable y que tiene por objetivo almacenar la información correcta.

- Independencia de datos: este concepto es, quizás, el que más ha ayudado a la rápida proliferación del desarrollo de sistemas de bases de datos.

Los elementos de base del modelo jerarquizado son registros lógicos que se unen entre sí, mediante punteros, para construir un árbol valorado. Éste, se entiende como aquel árbol cuyos nodos son tipos de registros lógicos y a la vez están valorados.

Una árbol valorado tiene una estructura puramente de arboles n-arios, y mantiene las estructuras y limitaciones de éste; un conjunto de registros y enlaces donde existe una única raíz (distinguido por ser el único que no recibe ninguna flecha), desde la que se desprenden los demás registros. Como enlaces se representan a los punteros (apuntadores a direcciones físicas) que puntan a registros que contienen información relacionada jerárquicamente con otra. Los enlaces son asociaciones entre exactamente dos registros. Cada registro padre, puede tener muchos hijos, y cada hijo puede tener un solo padre. No hay enlaces entre hermanos (como hermanos entendemos dos registros hijos de un mismo padre, aunque hay otros tipos de implementación que sí los permiten).

5.4.1. Altura (o Niveles) de una árbol

Se define como la altura de un árbol a la cantidad de "líneas"formadas por los registros de un árbol.

5.4.2. Momento de un árbol

Se entiende por momento de un árbol como los valores contenidos en éste en un instante dado. Un momento en un árbol determinado varía según la información contenida en él cambia o es actualizada (éstos cambios son producidos por que en la vida real, las entidades varían su información).

5.4.3. Peso de un árbol

Se define el peso de un árbol como la cantidad de hojas que contiene un árbol. Cada hoja se distingue por ser un nodo del que no se desprenden enlaces (flechas).

5.5. Ventajas

1. El acceso a los datos es rápido, es fácil de establecer su estructura, se pueden predefinir algunas estructuras de datos para establecer cambios a futuro.

2. Este modelo es el más utilizado en la actualidad para reducir problemas de administración de datos dinámicamente.

3. Soporta datos compartidos, se adapta a los cambios fácilmente, Asegura la integridad de los datos.

4. Un árbol con todo su entrerramado de relaciones, en el que la conexión es fija y sólo puede ser cambiada modificando una porción de código, suministra, sin embargo, la ventaja de que la navegación se realiza de una forma muy rápida.

5. Es fácil de ver la estructura de la base de datos.

6. Su implementación es sencilla y rápida de implantar.

7. Se predefinen relaciones, lo que simplifica las variaciones a futuro.

5.6. Desventajas

1. Es necesario conocer la estructura interna para recorrer el árbol en búsqueda de un dato solicitado, si el dato que se solicita se encuentra varios niveles abajo, es necesario pasar por todos los antecesores de éste.

2. Las operaciones de insertar y borrar se tornan complejas.

3. Puede dar lugar a la consistencia de los datos cuando se llevan a cabo actualizaciones.

4. Resulta inevitable el desaprovechamiento de espacio. Solución: introducir el concepto de registro virtual. Este tipo de puntero no contiene un valor de dato, sino un puntero lógico a un registro físico concreto. Cuando hay que replicar un registro en varios arboles de una base de datos, se guarda una copia de ése registro en uno de los árboles y se

sustituyen los demás por registros virtuales que contiene un puntero a ése registro físico.

5. La extracción de la información de una unidad que se encuentra varios niveles abajo requiere navegar por un camino a través de las unidades y sus relaciones hasta llegar a ella.
6. Presenta la desventaja de que es necesario un conocimiento en profundidad de las unidades de información y de sus relaciones entre sí. Adicionalmente, combinar la información de unidades que residen en ramas muy separadas de la estructura arbórea es una tarea que consume tiempo y esfuerzo.
7. Las operaciones de insertar y borrar son complejas.
8. Las relaciones Nodo a Nodo **NO** son implementadas de una forma eficiente, pues genera redundancia.

El esquema es una estructura arborescente compuesta de nodos, que representan las entidades, enlazados por arcos, que presentan las asociaciones o interrelaciones entre dichas entidades.

La estructura del modelo de datos jerárquico es un caso particular del modelo en red, con fuertes restricciones adicionales derivadas de que las asociaciones del modelo jerárquico deben formar un árbol ordenado, es decir, un árbol donde el orden de los nodos es importante.

Una estructura jerárquica, tiene las siguientes características:

- El árbol se organiza en un conjunto de niveles.
- El nodo raíz, el más alto de la jerarquía, se corresponde con el nivel 0.
- Los arcos representan las asociaciones jerárquicas entre dos entidades y no tienen nombre, ya que no es necesario porque entre dos conjuntos de datos sólo puede haber una interrelación.
- Mientras que un nodo de nivel superior (padre) puede tener un número ilimitado de nodos de nivel inferior (hijos), al nodo de nivel inferior sólo le corresponde un único nodo de nivel superior. en otras palabras, un padre puede tener varios descendientes o hijos, pero un hijo sólo tiene un padre.

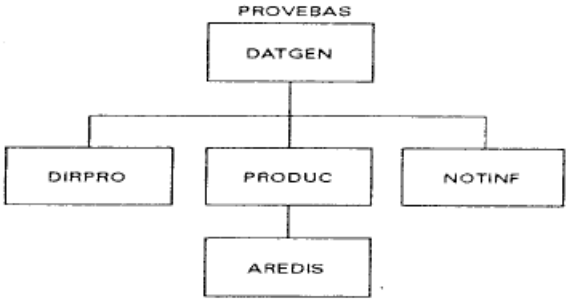


Figura 5.3: Case de una Empresa de ámbito nacional.

- Todo nodo, a excepción del nodo raíz, obligatoriamente tiene un padre.
- Se llaman hojas a los nodos que no tienen descendientes.
- Se llama altura al número de niveles de la estructura jerárquica.
- Se denomina momento al número de nodos.
- El número de hojas del árbol se llama peso.
- Sólo existen las interrelaciones 1:1 o 1:N
- Cada nodo no terminal y sus descendientes forman un subárbol, tal que un árbol es una estructura recursiva.

El árbol se recorre en preorden; es decir, raíz, subárbol izquierdo y subárbol derecho. Entre las restricciones propias de este modelo resaltan:

1. A) Cada árbol tiene un único segmento raíz.
2. B) No puede definirse más de una relación entre dos segmentos dentro de un árbol.
3. C) No se permiten las relaciones reflexivas de un segmento consigo mismo.
4. D) No se permiten las relaciones N:M.

5. E) No se permite que exista un hijo con más de un padre.
6. F) Para cualquier acceso a la información almacenada, es obligatorio acceder por la raíz del árbol, excepto en el caso de utilizar un índice secundario.
7. G) El árbol debe recorrer siempre de acuerdo a un orden prefijado: el camino jerárquico.
8. H) La estructura del árbol, una vez creada, no se modifica.

Las estructuras jerárquicas se clasifican como:

- Lineales: es un caso particular y simple donde que cada tipo de registro padre sólo tiene un tipo de registro hijo, ver interrelación entre DEPARTAMENTO y EMPLEADO.
- Arborescente propiamente dicha: un tipo de registro padre puede tener varios tipos de registro descendientes.

5.7. Esquema y ocurrencia

Un esquema jerárquico consiste en una descripción de un determinado universo mediante un árbol, en el que los nodos representan los tipos de registro (entidades) y los arcos, los tipos de interrelaciones jerárquicas existentes entre los mismos.

Una ocurrencia de dicho esquema también es un árbol, pero sus nodos representan las ocurrencias de los registros, y los arcos, las interrelaciones jerárquicas entre dichas ocurrencias.

Una base de datos jerárquica está formada por una colección o bosque de árboles disjuntos.

5.7.1. Diagramas de Ocurrencias (DO)

- Cada uno de los rectángulos representa la instancia de un tipo de registro.
- Los arcos definen la relación existente entre las instancias de dos niveles contiguos del árbol.

- No se especifican los campos de los registros.
- Cuando la información almacenada en la base de datos es muy grande, se convierte en inmanejable.
- Si no aparece alguna instancia de alguno de los niveles del árbol, la relación asociada no se representa.

5.7.2. Diagrama de Bachman (DA)

- Describe gráficamente el esquema lógico de una base de datos jerárquica.
- Los rectángulos representan los tipos de registro.
- Los arcos representan la relación existente entre los tipos de registro.
- Los arcos se etiquetan porque puede existir más de una relación entre dos tipos de registro (no en el modelo jerárquico).
- La flecha simple-doble indica que existe un único padre para cada hijo, pero pueden existir cero o más hijos de un padre.

5.8. Problemas

El modelo de datos jerárquico tiene importantes inconvenientes, propios de su rigidez, la cual deriva de la falta de capacidad de las organizaciones jerárquicas para representar, sin redundancias, ciertas estructuras muy difundidas en la realidad, como son las interrelaciones reflexivas y N:M. Ver la figura **??**

La poca flexibilidad de este modelo obliga a la introducción de redundancias cuando es preciso instrumentar, mediante el modelo jerárquico, situaciones del mundo real que no responden a una jerarquía; así ocurre con el esquema no jerárquico de la figura **??** al adaptar al modelo jerárquico, donde es preciso crear dos árboles e introducir redundancias, ya que los registros D, H e I aparecen en ambas jerarquías. Se trata de una pobre solución de diseño.

Se calcula un índice de redundancia mediante

$$ir\ \varepsilon\ \frac{N1 \times 100}{N1.totaldenodos}$$

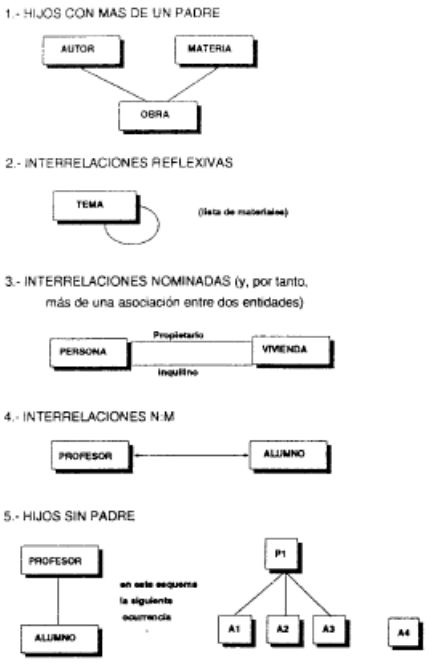


Figura 5.4: Estructuras no admitidas en el modelo jerárquico puro.

En nuestro caso, el índice de redundancia se calcula como

$$ir\ \varepsilon\ \frac{3 \times 100}{20\ por\ ciento}$$

5.9. Limitaciones

La redundancia calculada es lógica que, en general, no coincide con la redundancia física. Esto es debido a que al almacenar los datos en el dispositivo físico no se repiten los registros completos, sólo los identificadores o se introducen punteros. En general, los productos ofrecen algún tipo de solución a estos problemas.

Otra limitación importante del modelo jerárquico es no representar interrelaciones N:M, como la existente entre profesores y alumnos. Se pensaría en emplear la estructura que aparece en la Figura 5.5, duplicando las ocurrencias del registro ALUMNO.

Figura 5.5: Ejemplo base de datos jerárquica

Además del problema que provocan estas redundancias no controladas por el sistema, otro importante inconveniente en este tipo de solución es la no conservación de las simetrías naturales existentes en el mundo real.

Las actualizaciones en bases de datos jerárquicas también originan problemas, debido a las restricciones inherentes al modelo:

- Todo ingreso que **NO** corresponda a un nodo raíz, tiene un padre. Es imposible insertar en la base de datos un alumno que no tenga asignado profesor. Se resuelve el problema creando un padre ficticio para este registro, pero con ello se complica la labor del usuario de la base de datos, que tiene, cuando requiera conectar el registro a su padre real, eliminarlo de la base de datos y volver a introducirlo ligado a su verdadero padre. Además, el sistema no distingue entre padre ficticio y el real, cuando cuente el número de profesores de la base de datos da uno más.

- Eliminar un registro implica que desaparezca todo el subárbol que tiene a dicho registro como nodo raíz, con lo que desaparecen datos importantes en la base de datos.

Los SGBD basados en el modelo jerárquico facilitan instrumentos que los dotan de una mayor flexibilidad para representar estructuras no estrictamente jerárquicas.

Por lo que respecta a las restricciones de usuario, el modelo jerárquico no ofrece posibilidad de definirlas.

En cuanto a las ventajas proporcionadas por los SGBD de tipo jerárquico, destacan su sencillez de comprensión y la facilidad de instrumentación en los soportes físicos, aunque esto depende en gran medida de los productos.

5.10. Transformación

Hemos señalado los inconvenientes que presenta el modelado del mundo real en esquemas jerárquicos, e indicado una técnica de diseño jerárquico que consiste en introducir redundancias.

Se evita la pérdida de simetrías introduciendo mayor redundancia con la transformación de un esquema E/R con dos entidades y una interrelación N:M es un esquema jerárquico en el que existen dos árboles, de modo que se conservan las simetrías naturales, ya que los algoritmos para dos preguntas simétricas, como son recuperar los alumnos de un profesor y los profesores de un alumno, también son simétricos.

Soluciones de este tipo no son en absoluto eficientes.

A partir del modelo E-R analizamos la forma de transformar algunos tipos de interrelaciones al modelo jerárquico.

1. A) Interrelaciones 1:N con cardinalidad mínima 1 en la entidad padre. En este caso no existe problema y el esquema jerárquico resultante es prácticamente el mismo que en el ME/R.

2. B) Interrelaciones 1:N con cardinalidad mínima 0 en el registro propietario. El problema es que pueden existir hijos sin padre, por lo que o se crea un padre ficticio para estos casos o se crean dos estructuras arborescentes. La primera estructura arborescente tiene como nodo padre el tipo de registro A y como nodo hijo los identificadores del tipo de registro B. Así no se introducen redundancias, estando los atributos de la entidad B en la segunda arborescencia, en la cual sólo existe un nodo raíz B sin descendientes.

3. C) Interrelaciones N:M La solución es muy parecida, también creando dos arborescencias. Esta solución es independiente de las cardinalida-

des mínimas. Se podría suprimir, en la primera arborescencia o en la segunda, el registro hijo, pero no se conservaría la simetría.

4. D) Interrelaciones reflexivas.

La jerarquía A) se utiliza siempre que se desee obtener la explosión.

La aplicación de estas normas de diseño evita la introducción de redundancias, así como la pérdida de simetría, pero complica el esquema jerárquico resultante que está constituido por más de un árbol, lo que no resulta fácilmente comprensible a los usuarios.

5.11. Lenguaje

Una instrucción de un lenguaje de manipulación consta:

- Un operador que indica el tipo de operación a realizar.
- Los datos sobre los que se lleva a cabo la operación.
- Una condición, que servirá para seleccionar el conjunto de datos sobre el que se desea trabajar, y que es una expresión de tipo lógico, es decir, constantes y variables unidas por operadores de comparación y del álgebra de Boole.

5.12. Manipulación

La manipulación de datos jerárquicos primero localiza (selecciona) los datos que va a recuperar o actualizar. A) Localización o selección.

La función de selección jerárquica es tipo navegacional, es decir, trabaja registro a registro. Dada la sencillez del modelo, la función de selección también es sencilla, existiendo las siguientes formas básicas de búsqueda:

- Seleccionar un determinado registro que cumpla cierta condición. En el lenguaje DL/I se realiza este tipo de selección mediante una sentencia (GET UNIQUE -GU-) que activa el primer registro que cumpla la condición especificada en el predicado de la sentencia.

- Seleccionar el siguiente registro, que se encuentra definido al existir un único camino jerárquico. También en este caso se especifica una condición que debe cumplir el registro para ser seleccionado. En DL/I se utiliza una sentencia (GET NEXT -GN-) que selecciona y, al mismo tiempo, recupera el siguiente registro en el preorden.
- Seleccionar el siguiente registro dentro de un padre. Esta sentencia (GET NEXT PARENT -GNP-) es análoga a la anterior, pero termina cuando no haya más descendientes de ese padre.
- Seleccionar el registro padre de otro dato (activado previamente) se conoce como normalización jerárquica ascendente, mientras que la selección de descendientes se llama normalización jerárquica descendente.

B) Acción Una vez seleccionado un registro, se realiza sobre él una acción de recuperación o actualización. La recuperación, asociada a la selección en el DL/I, consiste en llevar el registro marcado como activo al área de entrada/-salida. Se utiliza la sentencia GET.

En cuanto a la actualización, es preciso distinguir entre:

- Insertar un conjunto de datos (INSERT -ISRT-)
- Borrar un conjunto de datos (DELETE -DLET-)
- Reemplazar -modificar- uno o varios campos de un registro (REPLACE -REPL-)

Debido a la naturaleza jerárquica de las conexiones entre registros, las inserciones y borrados de registros requieren consideraciones especiales:

- Cuando un nuevo registro se inserta en una base de datos jerárquica, excepto para la raíz, tiene que ser conectado a un nodo padre previamente seleccionado mediante alguna sentencia de selección. El nuevo registro se inserta como hijo del registro padre seleccionado.
- Cuando un registro se borra en una base de datos jerárquica, excepto si se trata de una hoja, se han de borrar los registros descendientes de él.

5.13. Caso típico

Sea una determinada empresa de ámbito nacional con sucursales por todo el país; esta empresa tiene centralizadas todas las compras de material en la oficina central, para lo cual dispone de una base de datos jerárquica para almacenar los datos de sus proveedores.

La base de datos de proveedores, denominada PROVEBAS, presenta cinco segmentos. El segmento raíz en el que se almacenan los datos que son comunes a los proveedores, como: Nombre de la empresa, Director de la empresa, RIF, entro otros. Este segmento se denomina DATGEN.

En el segundo nivel del árbol hay tres segmentos dependientes del segmento raíz. El primero de ellos contiene las direcciones de las sucursales de la empresa proveedora, indicando: Calle, Número, ciudad, Tipo de dirección, entre otros. El nombre de este segmento es DIRPRO.

El segundo segmento que ocupa este nivel es el que contiene los datos de los productos suministrados por cada una de las empresas proveedoras, Este segmento se denota como PRODUC. El último segmento del segundo nivel guarda las diferentes notas informativas que sobre un proveedor envían las sucursales a la oficina central, este segmento se denomina NOTINF.

El tercer nivel del árbol está ocupado por un sólo segmento que almacena las zonas de distribución de cada uno de los productos suministrados por los diferentes proveedores, este segmento se conoce como AREDIS y depende del segmento PRODUC.

Una OCURRENCIA de un segmento de una base de datos jerárquica es el conjunto de valores particulares que toman los campos que lo componen en un momento determinado.

Un REGISTRO de la base de datos es el conjunto formado por una ocurrencia del segmento raíz y las ocurrencias del resto de los segmentos de la base de datos que dependen jerárquicamente de dicha ocurrencia raíz. La relación PADRE/HIJO en la que se apoyan las bases de datos jerárquicas, determina que el camino de acceso a los datos sea ÚNICO; es denominado CAMINO SECUENCIA JERÁRQUICA, comienza siempre en una ocurrencia del segmento raíz y recorre la base de datos de arriba a abajo, de izquierda a derecha y, por último, de adelante a atrás.

Veamos el siguiente caso, el registro de un empleado (nodo hijo) se relaciona

con el registro de su departamento (nodo padre), pero no al contrario. Esto implica que sólo se consulta la base de datos desde los nodos hoja hacia el nodo raíz. La consulta en el sentido contrario requiere una búsqueda secuencial por todos los registros de la base de datos (como consultar todos los empleados de un departamento). En las bases de datos jerárquicas no existen índices para esta tarea.

5.14. IMS

IBM Information Management System (IMS) es un gestor de bases de datos jerárquicas, es transaccional con alta capacidad de proceso. Es un producto de software IBM. Fue uno de los primeros sistemas de base de datos comerciales. Su primera versión, AIMS/360 V1, apareció en 1968.

IBM diseñó el IMS con Rockwell y Caterpillar en 1966 debido al Programa Apolo. El desafío de IBM era inventariar la lista de materiales del cohete lunar Saturno V y de la nave Apolo.

El primer mensaje **IMS READY** apareció en un terminal IBM 2740 en Downey, California un 14 de agosto de 1968. IMS todavía se usa extensamente 40 años después y, con el tiempo, ha visto interesantes desarrollos como el sistema IBM Sistema/360, hoy convertido en z/OS y Sistema z9. IMS soporta aplicaciones desarrolladas en Java, JDBC, XML y Servicios Web.

IMS DB

Las funcionalidades de IMS relacionadas con bases de datos reciben el nombre de **IMS DB** (IMS DataBases). Hay tres tipos de bases de datos jerárquicas:

- Bases de datos Full function
    - Descendientes directas de las bases de datos **Data Language I** desarrolladas para el Apolo, pueden tener índices primarios y secundarios accedidos usando llamadas DL/I desde la aplicación, algo similar a llamadas SQL a Oracle o DB2 (de hecho, SQL debe su herencia a DL/I).

- Este tipo de bases de datos tiene una gran variedad de métodos de acceso, aunque los dominantes son: **Hierarchical Direct (HDAM)** (Acceso directo jerárquico) y **Hierarchical Indexed Direct (HIDAM)** (Acceso directo jerárquico indexado). Los otros métodos son **Simple Hierarchical Indexed Sequential (SHISAM)** (Acceso simple jerárquico indexado secuencial), **Hierarchical Sequential (HSAM)** (Acceso jerárquico secuencial) y **Hierarchical Indexed Sequential (HISAM)** (Acceso jerárquico indexado secuencial).
- Los datos se almacenan usando **VSAM**, un tipo de acceso nativo de z/OS, u **Overflow Sequential (OSAM)** (Acceso de desbordamiento secuencial), un tipo de acceso propio de IMS que optimiza la Entrada/Salida de algunos patrones.

- Bases de datos Fast Path
  - Las bases de datos Fast Path pueden ser bien **Data Entry Databases** (DEDB) o **Main Storage Databases** (MSDB). Ambos tipos carecen de indexación, pero a cambio están optimizados para ofrecer elevados índices de acceso.
  - Las MSDB están destinadas a desaparecer debido a sus fuertes restricciones. No permiten altas ni bajas y las referencias directas se resuelven mediante búsquedas binarias.
  - Las DEDB pueden llegar a ser tan complejas como las de DL/I mientras mantienen su rendimiento, por lo que están sustituyendo a MSDB.
  - Actualmente se aprovecha la riqueza estructural de las DEDB y la mejora de rendimiento de las MSDB gracias a la opción de poner el memoria virtual áreas de una DEDB. Esta funcionalidad comparte una misma base de datos entre distintos sistemas usando poniendo las áreas en la **coupling facility**.
- Bases de datos High Availability Large Databases (HALDB)
- IMS V7 introdujo HALDBs, como una extensión de la base de datos Full Function para proveer una mejor disponibilidad de la misma,

mejor manejo de grandes volúmenes de datos, y, con IMS V9, reorganización online para soportar alta disponibilidad.

IMS TM

Las funcionalidades de IMS relacionadas con la gestión transaccional reciben el nombre de IMS TM (IMS Transaction Manager), anteriormente conocido como IMS DC **IMS DataControl**.

5.15. Aplicaciones

Las bases de datos jerárquicas organizan la información en forma de árbol. Los datos dependen todos de una entidad raíz, **padre**. Los datos dependientes del raíz son **hijos** suyos. A su vez estos hijos tienen hijos y así sucesivamente (Ver la figura 5.6).

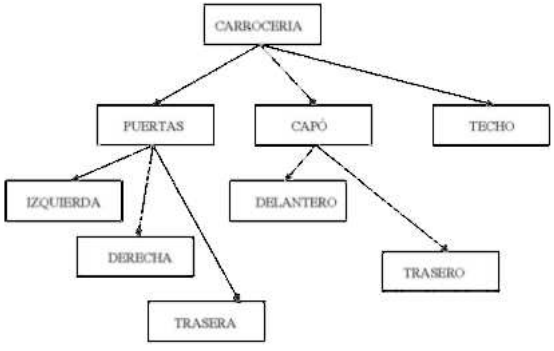


Figura 5.6: Base de datos jerárquica.

La relación entre padre e hijos es siempre de uno a muchos, de manera que un hijo siempre tienen un sólo padre, pero un padre puede tener varios hijos. Para acceder a una entidad, se parte siempre del raíz y se recorren los hijos según un orden preestablecido. Cuando se elimina un padre, se eliminan todos sus hijos también. Estas bases de datos son muy apropiadas para datos

que se prestan bien a una estructura ramificada. Se utilizan en buscadores de INTERNET, donde son muy eficientes.

- El producto comercial de tipo jerárquico más extendido y el único que ha llegado hasta nuestros días es el IMS de IBM con su lenguaje de datos DL/I2.
- Otro sistema Jerárquico, el System 2000 tuvo una alta aceptación comercial y fue adquirido por el Instituto SAS.

Capítulo 6

Modelo de Red

6.1. Introducción

Este modelo representa las entidades en forma de nodos (grafo) y las asociaciones o interrelaciones entre estas, mediante los arcos que unen a dichos nodos.

Este modelo se caracteriza por ser navegacional, es decir, la recuperación y la actualización de la base de datos se lleva a cabo registro a registro. Otra característica es que su implementación se lleva a cabo por medio de punteros.

6.2. Definiciones

En este modelo las entidades se representan como nodos y sus relaciones son las líneas que los unen.

En esta estructura cualquier componente se relaciona con otro. Tipos de interrelaciones Tipos de registros: Se definen habitualmente como colecciones de elementos de los datos lógicamente relacionados.

Los conjuntos Un conjunto expresa una interrelación uno a muchos, o uno a uno entre dos tipos de registros.

En el modelo CODASYL, el conjunto constituye el elemento básico para la representación de las interrelaciones.

Por medio de los conjuntos se establecen asociaciones 1:N (las asociaciones

1:1 son un caso particular de las 1:N) a dos niveles, en las que el nodo raíz se denomina propietario y los nodos dependientes se denominan miembros. Componentes estáticos del modelo en red y componentes dinámicos del modelo en red. Vea la figura 6.1.

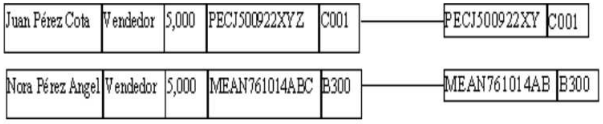


Figura 6.1: Registros relacionados.

6.3. Conceptos básicos

Una Base de Datos de Red se compone de dos conjuntos, a saber:

- El Conjunto de los Registros. Un conjunto de instancias múltiples de varios tipos de registros.
- El Conjunto de las Relaciones. Un conjunto de instancias múltiples de varios tipos de relaciones.

El modelo de red organiza datos en construcciones, registros y conjuntos. Los registros contienen campos (que son organizados jerárquicamente, como en el lenguaje COBOL). Los conjuntos (no confundir con conjuntos matemáticos) definen de una a varias relaciones entre registros: un propietario, muchos miembros.

Un registro puede ser un propietario en cualquier número de conjuntos, y un miembro en cualquier número de conjuntos.

El modelo de red es una variación del modelo jerárquico, al grado que es construido sobre el concepto de múltiples ramas (estructuras de nivel inferior) emanando de uno o varios nodos (estructuras de nivel alto), mientras el modelo se diferencia del modelo jerárquico en esto, las ramas pueden estar unidas a múltiples nodos. El modelo de red representa la redundancia en datos de una manera más eficiente que en el modelo jerárquico.

Las operaciones del modelo de red son de navegación en el estilo: un programa mantiene una posición corriente, y navega de un registro al otro por siguiente las relaciones en las cuales el registro participa.

Los registros también pueden ser localizados por suministrando valores claves. Aunque esto no sea un rasgo esencial del modelo, las bases de datos de red generalmente ponen en práctica las relaciones de juego mediante indicadores que directamente. dirigen la ubicación de un registro sobre el disco. Esto da el funcionamiento de recuperación excelente, a cargo de operaciones como la carga de base de datos y la reorganización. La mayor parte de bases de datos de objeto usan el concepto de navegación para proporcionar la navegación rápida a través de las redes de objetos, generalmente usando identificadores de objeto como indicadores ïnteligentes"de objetos relacionados. Objectivity/DB, por ejemplo, los instrumentos llamados 1:1, 1:muchos, muchos:1 y muchos:muchos, llamados relaciones que pueden cruzar bases de datos. Muchas bases de datos de objeto también apoyan SQL, combinando las fuerzas de ambos modelos.

6.4. Caracteristicas

Un conjunto es una colección nominada de dos o más tipos de registros que representa una interrelación 1:N Cada conjunto debe tener un tipo de registro propietario y uno o más registros miembros. Pueden existir conjuntos singulares en los que el propietario es el sistema.

Ejemplos de bases de datos en red son ADABAS, TOTAL, IMAGE, entre otras.

- Cualquier registro puede ser declarado propietario de uno o varios conjuntos.
- Cualquier registro puede ser declarado miembro de uno o varios conjuntos.
- Cualquier registro puede ser declarado propietario en un conjunto y miembro en otro conjunto distinto.

Una base de datos en red consiste en un conjunto de registros conectados entre si mediante punteros.

Los registros son en muchos aspectos parecidos a las entidades del modelo entidad-relación (E-R).

Cada registro es un conjunto de campos (atributos), cada uno de los cuales sólo contiene un valor de datos.

Vea la figura 6.2.

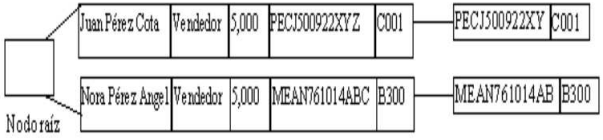


Figura 6.2: Nodo raíz (puntero) con sus registros.

Los punteros son asociaciones entre dos registros. Por tanto, los punteros se consideran una forma binaria de relación en el sentido del modelo E-R.

6.5. Ventajas

- Naturaleza distribuida de algunas aplicaciones.
- Aumento de la fiabilidad y la disponibilidad.
- Compartir los datos pero con control local.
- Facilidad de ampliación.
- Resulta más flexible que el jerárquico.

6.6. Desventajas

- Complejidad.
- En este tipo de modelo de datos su estructura comienza a tomar apariencia de **tela de araña** con apuntadores que salen en todas direcciones.
- Aumento de costes (mantenimiento, red...).

- Seguridad en los datos.
- Falta de estándares.
- Diseño complicado.

6.7. Esquema

Cualquier registro puede ser propietario de uno o varios SET. Podrán existir SET singulares en los que el propietario es el sistema (una entidad se interrelaciona consigo mismo). Un SET es una colección nominada de dos o más tipos de registros que representan un tipo de interrelación 1:N (en consecuencia también 1:1). Cualquier registro puede ser miembro de uno o varios SET. Vea la figura 6.3.

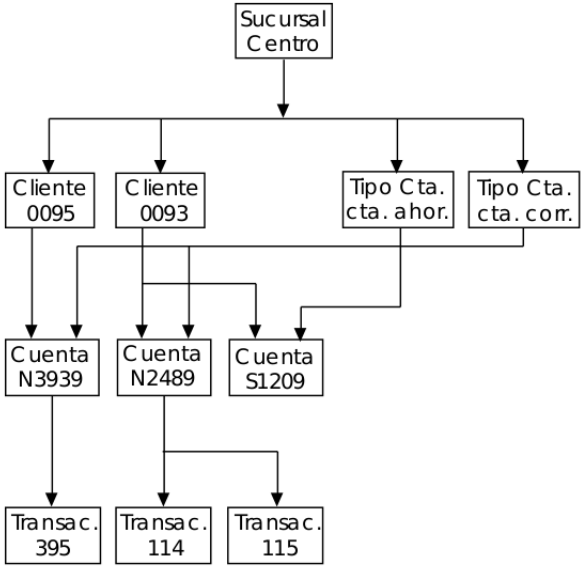


Figura 6.3: Diagrama de un modelo de red.

El número de SET en el sistema es ilimitado. Cada SET tendrá un tipo de registro propietario y uno o más tipos de registros miembro.

Gestión centralizada del almacenamiento físico.

Independencia del almacenamiento físico.

Seguridad.

Uso concurrente. Objetivos Flexibilidad para los usuarios. Estrategias de búsqueda diversas. Independencia respecto a los lenguajes. Descripción de datos independientes. Independencia de los programas respecto a los datos. Flexibilidad en el modelo de datos. Facilidad para el usuario. Interfaces con múltiples lenguajes.

- En este modelo es posible representar asociaciones M:N, para ello es necesario transformarlas en dos asociaciones 1:M unidas a través de un tipo de registro de intersección denominado **NUB**. Con esto se elimina la redundancia generada en el modelo jerárquico.
- Los problemas de insertar, eliminar y modificar la base de datos planteados en el modelo jerárquico, son simples.
- Hay que tener cuidado al eliminar registros que intervienen en asociaciones M:N, pues es posible que se creen inconsistencias si **NO** se eliminan los registros NUB involucrados.

6.8. Problemas

En el modelo de red no existen restricciones, si queremos representar que un cliente puede tener varias cuentas, cada una de las cuáles sólo puede tener un titular, y cada cuenta ésta en una sola sucursal, que por supuesto puede ser compartida por varias cuentas, éste sería el esquema:

Cliente Cuenta Sucursal

Con el modelo relacional tendríamos ambas entidades definidas de la siguiente forma:

Cliente = (Nº Cliente: Acceso Principal; Nombre, Dirección, Nº Cuenta: Acceso Ajeno)

Cuenta = (Nº Cuenta: Acceso Principal; Saldo)

Se puede diseñar una base de datos de red la cual parta de un esquema de entidad de relacion (ER). Los Pasos pueden ser:

- Trabajar con entidades normales y por cada entidad crear un tipo de registros con muchos o todos sus atributos cuyos campos pueden ser simples o compuestos.
- Trabajar con entidades comunes, las cuales son entidades que dependen de otra para existir.
- Para cada entidad débil se crea un tipo de registro que lo represente. Además, debemos relacionarla con la entidad de la que depende, para ello la entidad fuerte de la que depende la débil viene ser propietario y la débil, miembro.
- Trabajar con vínculos de uno-uno (1:1) y uno-muchos no recursivos. En el caso de la relación de uno a uno se elige cualquiera de los dos registros como propietario y al otro como miembro.
- Si la relación es de muchos (1:N) se escoge como propietario al registro que representa a la entidad que está al lado 1 de la relación y como miembro al registro que representa a la entidad que esta al lado N de la relación.
- Trabajar con relaciones de muchos a muchos (N:M). Por lo que se tiene que crear un tipo de registro, el cual será miembro de los dos registros que representan a las entidades de la relación.
- Trabajar con vínculos recursivos con vínculos de 1:1 o 1:N. Para ambos casos se crea un nuevo registro. El cual se unirá al registro que representa la entidad a través de tipo de conjuntos .
- Por ultimo se trabaja con los vínculos que relacionan a más de dos entidades.Por lo que se tiene que crear un nuevo tipo de registro, el cual será el registro miembro de los registros que representan a las entidades; los cuales vendrían a ser los registros propietarios.

La programación facilita realizar una o varias tareas, tales como buscar, leer, insertar, eliminar y modificar los registros. Por lo que es necesario el uso del programa de manipulación de datos (DML). El genera órdenes de registro por registro incorporadas en un lenguaje de programación de aplicación general, se llama lenguaje anfitrión.

Para comprender perfectamente es necesario entender algunos términos.

- Registros actuales: Son registros específicos identificados de la base da datos.
- Indicadores de actualidad, los cuales son los responsables de llevar el control de varios registros y ocurrencias; existen tres tipos de indicadores de actualidad.
- Actual de tipo de registros
- Actual de tipo de conjuntos
- Actual de unidad de ejecución
- Área de trabajo del usuario: Es el conjunto de variables locales las cuales tienen diferentes tipos de registros para que el programa anfitrión pueda manipularlos.

6.9. Limitaciones

Es una simplificación del modelo en red general.

Sólo se admiten ciertos tipos de interrelaciones y, además, se incluyen otras restricciones adicionales. Éstas, no limitan demasiado la flexibilidad original del modelo en red general, pero permiten tener una instrumentación eficiente.

Campo o Elemento de dato Unidad de datos más pequeña a referenciar. Pueden ser de distintos tipos, y definirse como dependiente de valores de otros elementos. Un campo tiene un nombre y una ocurrencia del mismo contiene un valor que puede ser de distinto tipo (booleano, numérico, etc.)

Agregado de datos Se asemeja a los campos de un archivo o a los atributos de otros modelos. Puede ser un vector con número fijo de elementos (como una fecha compuesta de día, mes y año), o bien un grupo repetitivo (como un conjunto de salarios por diferentes conceptos).

Registro Colección nominada de elementos de datos. Es la unidad básica de acceso y manipulación de la base de datos. Se asemeja a los registros en archivos y a las entidades en el modelo E/R.

Conjunto (SET o COSET) Colección nominada de dos o mas tipos de registros que establece una vinculación entre ellos. Es una colección de dos o

más tipos de registros que establece una vinculación entre ellos, Constituye el elemento clave y distintivo de este modelo. Origen de muchas restricciones. Las interrelaciones 1:N se representan aquí mediante SET.

Área Subdivisión nominada del espacio direccionable de la base de datos que contiene ocurrencias de registros.es la subdivisión del espacio de almacenamiento direccionable de la BD que contiene ocurrencias de registros (páginas de disco, cilindros, etc.). En un área puede haber ocurrencias de más de un tipo de registro y las de un mismo tipo de registro pueden estar contenidas en distintas áreas, aunque una ocurrencia determinada tiene que estar siempre asignada a un área y sólo a una.

Clave de base de datos Identificador interno único para cada ocurrencia de registro que da su ubicación en la base de datos.

Proporciona su dirección en la base de datos. Es un obstáculo para conseguir la independencia lógica / física. Suponía problemas el reutilizar una clave cuando se reorganizaba la base de datos.

En rigor, como elementos del modelo de datos lógico únicamente consideraríamos los cuatro primeros, ya que tanto el área como la clave de base de datos son elementos de tipo físico.

6.10. Transformación

6.10.1. Conversión Compleja-Simple

- Reduce el problema de la pérdida de información asociado a las redes complejas.
- La idea es convertir una relación muchos-a-muchos en dos relaciones uno-a-muchos, mediante la inserción de un nuevo tipo de registro.
- Tenemos un **registro intersección** si contiene algún tipo de información que se denomina **datos de la intersección**.
- En otro caso, se denomina **registro de enlace**.

6.11. Lenguaje

El lenguaje de manipulación de datos es de tipo navegacional (opera registro a registro), procedimental (requiere el conocimiento de la estructura física de la base de datos por parte del programador) y tiene que estar embebido en un lenguaje de programación anfitrión.

Como característica importante, destacar que en él se distingue la localización o selección de la acción (recuperación o actualización).

La navegación por la base de datos se lleva a cabo apoyándose en los indicadores de registro activo. En el modelo en red no se ha especificado ningún tipo de restricción en lo que respecta a las interrelaciones.

Quizás, esto quizá haga al modelo en red sencillo de utilizar, pero no deja de tener un carácter general y provoca que, en la práctica, su instrumentación no resulte fácil.

Es por esto, que los SGBD que se basan en el modelo en red, deben añadir una serie de restricciones a fin de implementar la base de datos físicamente y obtener un mayor rendimiento del sistema.

Un modelo de datos de este tipo, es el denominado **CODASYL**.

6.12. Manipulación

6.13. Caso típico

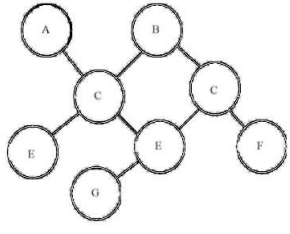


Figura 6.4: Nodo raíz (puntero) con sus registros.

Capítulo 7

Modelo Relacional

7.1. Introducción

Hemos mencionado tres tipos de modelado: conceptual, lógico y físico.

El modelo E-R se considera un modelo conceptual que a un nivel alto ver con claridad la información utilizada en algún problema o negocio.

En este capítulo nos concentraremos en desarrollar un buen modelo **lógico** que se conoce como **esquema de la base de datos** a partir del cual se podrá realizar el modelado físico en el DBMS, es importante mencionar que es un paso necesario, no se puede partir de un modelo conceptual para realizar uno físico.

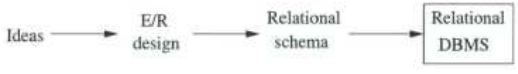


Figura 7.1: Trayectoria del modelo relacional

Una base de datos relacional consiste en un conjunto de tablas, a cada tabla se le asigna un nombre exclusivo y se compone por registros.

El modelo relacional utiliza una colección de tablas para representar datos y las relaciones entre ellos. Cada tabla tiene múltiples columnas, cada columna tiene un nombre único. Las tablas también son conocidas como relaciones. El modelo relacional es un modelo basado-en-registros.

Estos modelos se llaman así ya que la base de datos está estructurada en registros de formato fijo de diversos tipos. Cada tabla contiene registros de un tipo particular. Cada registro-tipo define un número fijo de campos, o atributos. Las columnas de la tabla corresponden a los atributos del registro-tipo.

En este capítulo nos concentramos en desarrollar un modelo **lógico** que se conoce como **esquema de la base de datos** a partir del cual realizamos el modelado físico en el SGBD, es importante mencionar que es un paso necesario, no se puede partir de un modelo conceptual para realizar uno físico.

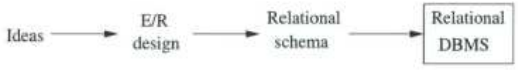


Figura 7.2: Trayectoria del modelo relacional

7.2. ¿Por qué modelo relacional?

Resulta confuso el concepto de modelo entidad-relación vs modelo relacional, quizás porque comparten casi las mismas palabras. Como se mencionó, el objetivo del modelo relacional es crear un **esquema**, lo cual como se menciona posteriormente consiste de un conjunto de **tablas** que representan **relaciones** entre los datos.

Estas tablas, son construidas de diversas maneras:

- Creando un conjunto de tablas iniciales y aplicar operaciones de normalización hasta conseguir el esquema óptimo. Las técnicas de **normalización** se explican adelante en este capítulo.
- Convertir el diagrama E-R a tablas y aplicar operaciones de normalización hasta conseguir el esquema óptimo.

La primer técnica fue la primera en existir y, como es de suponerse, la segunda es más conveniente en varios aspectos:

- Un diagrama visual es útil para apreciar los detalles, de ahí que se llame modelo conceptual.

- Crear las tablas iniciales es más simple a través de las reglas de conversión.

Pensaríamos que es lo mismo porque hay que **normalizar** las tablas, pero la ventaja de partir del modelo E-R es que la normalización, por lo general, es mínima.

Esto tiene otra ventaja, aún cuando se normalice de manera deficiente, se garantiza un esquema aceptable; en la primer técnica no es así.

7.3. Características del Modelo Relacional

El modelo de datos relacional, perteneciente al grupo de modelos de datos orientados a registros, es el modelo de mayor uso y difusión en los distintos tipos de organizaciones, aunque con importantes cambios y adecuaciones realizados a través del tiempo.

El concepto fundamental, en el Modelo Relacional, es que los datos se representan de una sola manera, en el nivel de abstracción que es visible al usuario, y es, específicamente, como una estructura tabular – conformada por filas y columnas – o como una tabla con valores.

Los conceptos básicos que se utilizarán, a partir de este momento, son propios del modelo y se resumen en:

- Relación
- Tupla o fila
- Cuerpo: conjunto de tuplas
- Columna o atributo
- Dato
- Cardinalidad: número de tuplas
- Dominio

7.4. Propiedades de las relaciones

Estas propiedades fijan limitaciones en el modelo, que son las que diferencian las relaciones de las tablas implementadas. Tienen su origen en la Matemática de conjuntos.

- Orden de las tuplas en una relación
- Orden de los atributos
- Todas las filas son distinta

7.5. Definiciones

Entidad: es un objeto del mundo real que es distinguible de todos los demás. Puede tratarse de cualquier tipo de objeto, persona, empresa, cosa, informe, etc. Las entidades se representan gráficamente como rectángulos, apareciendo su nombre en el interior. Las entidades a su vez, tienen atributos o propiedades. Cada ejemplar de una entidad se denomina instancia.

Existen dos tipos de entidades, fuertes y débiles. Una entidad fuerte es independiente de la existencia de otra, mientras que una entidad débil depende de la existencia de otra.

Las entidades débiles se representan mediante un doble rectángulo, es decir, un rectángulo con doble línea.

Atributos: describen propiedades que poseen cada entidad o relación. Gráficamente se representan por círculos unidos mediante líneas a los demás objetos. Cada atributo posee un conjunto de valores asociados llamado dominio. Éste, se define como los posibles valores que puede tomar dicho atributo.

Se distinguen varios tipos de atributos:

- Simples o compuestos: es aquel que tiene un único componente, indivisible en partes pequeñas; mientras que uno compuesto se puede dividir en otros pequeños con significado propio, y que mantienen una afinidad común.
- Derivados: es aquel que representa un valor formado a partir del valor de uno o varios atributos.

- Monovalorados o multivalorados: un atributo monovalorado es aquel que sólo tiene un valor para cada ocurrencia de la entidad o relación, como la estatura en una persona. Por el contrario, uno multivalorado tiene varios valores para cada ocurrencia de la relación o entidad, como el número de hijos.

Relación: es una correspondencia o asociación entre dos o más entidades. Cada relación describe su función mediante un nombre. Su representación gráfica es un rombo etiquetado en su interior mediante un verbo. Además, dicho rombo debe unirse mediante líneas con las entidades que relaciona (los rectángulos).

Identificador: es un atributo o conjunto de ellos que determina de modo único cada ocurrencia de esa entidad. Un identificador de una entidad tiene que cumplir:

1. No existen dos ocurrencias de la entidad con el mismo identificador.
2. Si se omite un atributo del identificador, la condición anterior deja de cumplirse.

Toda entidad tiene al menos un identificador y puede tener varios alternativos. Las relaciones no tienen identificadores.

7.6. Conceptos básicos

Los conceptos básicos a utilizar, a partir de este momento, son propios del modelo y se resumen en:

- Esquema
- Relación
- Tupla o fila
- Cuerpo: conjunto de tuplas
- Columna o atributo
- Dato

- Cardinalidad: número de tuplas
- Dominio
- Formalmente, los diagramas E-R son un lenguaje gráfico para describir conceptos.
- Informalmente, son simples dibujos o gráficos que describen la información que trata un sistema de información y el software que lo automatiza.

Atributos de la entidad **persona**:

- Cédula de Identidad (identificativo).
- Nombre.
- Apellidos.
- Dirección.
- Ciudad.
- Estado.
- Municipio.
- Parroquia

7.6.1. Diagramas Extendidos

Los diagramas Entidad-Relación no cumplen su propósito con eficacia debido a que tienen limitaciones semánticas. Por ese motivo se utilizan los diagramas Entidad-Relación extendidos que incorporan algunos elementos adicionales al lenguaje.

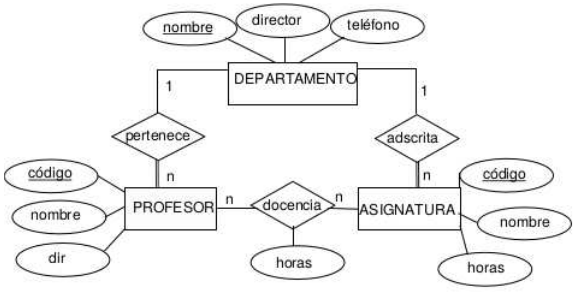


Figura 7.3: Diagrama Entidad-Relación

7.7. Características

Estas tablas, son construidas de diversas maneras:

- Creando un conjunto de tablas iniciales y aplicar operaciones de normalización hasta conseguir el esquema óptimo. Las técnicas de **normalización** se explican adelante en este mismo capítulo.
- Convertir el diagrama E-R a tablas y aplicar operaciones de normalización hasta conseguir el esquema óptimo.

La primer técnica fue la primera en existir y, como es de suponerse, la segunda es más conveniente en varios aspectos:

- Un diagrama visual es útil para apreciar los detalles, de ahí que se llame modelo conceptual.
- Crear las tablas iniciales es simple a través de las reglas de conversión.

Pensaríamos que es lo mismo porque hay que **normalizar** las tablas, pero partir del modelo E-R es que la normalización es mínima por lo general.

Esto tiene otra ventaja, aún cuando se normalice de manera deficiente, se garantiza un esquema aceptable; en la primer técnica no es así.

7.8. Conceptos básicos

7.8.1. Tablas

El modelo relacional proporciona un manera simple de representar los datos: una tabla llamada relación.

Relación Películas

Título	Año	Duración	Tipo
Star Wars	1977	124	color
Mighty Ducks	1991	104	color
Wayne's World	1992	95	color

La relación Películas gestiona la información de las instancias en la entidad Películas, cada renglón corresponde a una entidad película y cada columna corresponde a cada uno de los atributos de la entidad. Sin embargo, las relaciones representan más que entidades, como se explica más adelante.

7.9. Estructura Básica

Peter Chen propone el modelo entidad-relación como un concepto de modelado para bases de datos, con el propósito de **visualizar** los objetos que pertenecen a la base de datos como entidades (similar al modelo de programación orientada a objetos) las cuales tienen unos atributos y se vinculan mediante relaciones.

El modelo relacional fue propuesto por E.F. Codd en los laboratorios de IBM en California. Se trata de un modelo lógico, que establece una estructura sobre los datos, aunque posteriormente éstos puedan ser almacenados de múltiples formas para aprovechar características físicas concretas de la máquina sobre la que se implante la base de datos realmente.

El modelo relacional se ha establecido como el principal modelo de datos para las aplicaciones de procesamiento de datos. Ha conseguido la posición principal debido a su simplicidad y a que facilita el trabajo del programador en comparación con otros modelos anteriores como el de red y el jerárquico.

Actualmente, para la mayoría de las aplicaciones de gestión que utilizan ba-

ses de datos, el modelo más empleado es el relacional, por su gran versatilidad, potencia y por los formalismos matemáticos sobre los que se basa.

Este modelo representa la información del mundo real de una manera intuitiva, introduciendo conceptos cotidianos y fáciles de entender por cualquier informático.

El modelo de datos relacional, perteneciente al grupo de modelos de datos orientados a registros, es el modelo de mayor uso y difusión en los distintos tipos de organizaciones, aunque con importantes cambios y adecuaciones realizados a través del tiempo.

El concepto fundamental, en el Modelo Relacional, es que los datos se representan de una sola manera, en el nivel de abstracción que es visible al usuario, y es, específicamente, como una estructura tabular – conformada por filas y columnas – o como una tabla con valores.

Los conceptos básicos que se utilizarán, a partir de este momento, son propios del modelo y se resumen en:

- Relación
- Tupla o fila
- Cuerpo: conjunto de tuplas
- Columna o atributo
- Dato
- Cardinalidad: número de tuplas
- Dominio

7.10. Propiedades de las relaciones

Estas propiedades fijan limitaciones en el modelo, que son las que diferencian las relaciones de las tablas implementadas. Tienen su origen en la Matemática de conjuntos.

- Orden de las tuplas en una relación.

- Orden de los atributos.
- Todas las filas son distinta.

7.11. Conversión del modelo E-R a un esquema de base de datos

El modelo es una representación visual que gráficamente da una perspectiva de como se encuentran los datos involucrados en un proyecto.

Pero el modelo no presenta propiamente una instancia de los datos, es un ejemplo que muestra con claridad algunas datos selectos y cómo se relacionan en realidad. Por eso es conveniente crear un **esquema**, el cual consiste de tablas las cuales, en sus renglones (tuplas), contienen instancias de los datos.

7.11.1. Conversión a tablas desde un modelo con relaciones (1-1,1-m,m-m)

Veamos las siguientes reglas a seguir para crear dicho esquema:

- Modelo E-R conversión a tablas
- una tabla por cada conjunto de entidades
- nombre de tabla = nombre de conjunto de entidades
- una tabla por cada conjunto de relaciones m-m
- nombre de tabla = nombre de conjunto de relaciones
- definición de columnas para cada tabla
- conjuntos fuertes de entidades

columnas $=$ nombre de atributos
conjuntos débiles de entidades
columnas = clave_primaria (dominante) $\cup$ atributos(subordinado)
conjunto de relaciones R(m-m) entre A, B
columnas (R) = clave_primaria (A) $\cup$ clave_primaria (B) $\cup$ atributos(R)
conjunto de relaciones R (1-1) entre A y B

columnas (A) = atributos(A) ∪ clave primaria(B) ∪ atributos(R)
conjunto de relaciones R (1-m) entre A y B
columnas (B) = atributoss(B) ∪ clave primaria(A) ∪ atributos(R)

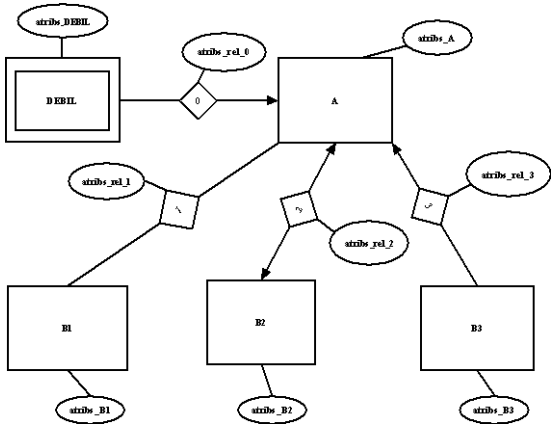


Figura 7.4: Tabla de conversión

Hasta ahora hemos estado definiendo el modelo relacional, y sus relaciones con el modelo Entidad-Relación. Pero ¿Cómo se convierte el modelo entidad-relación en el modelo relacional? Es decir, a partir de un esquema entidad-relación, ¿Cómo obtengo sus correspondientes tablas?.

En esta primera parte vamos a ver como convertir del modelo entidad-relación simple (llamémosle así para diferenciarlo del extendido) al modelo relacional. Para ello, aplicamos: Esta diferenciación se debe a que todas las claves ajenas deben hacer referencia a las claves primaria de otras tablas y consecuentemente no pueden ser nulas. Dicho de otra manera, toda referencia ajena debe hacerse a un campo único.

Propagando la de 1 en la de muchos (creando un campo en la de muchos que referencie a la de 1) si cada elemento de la entidad que participa con muchos aparece en la entidad de uno, es decir, si TODOS los elementos de la entidad de muchos tienen asociado uno de la entidad de uno.

Relaciones 1:1

Transformar la relación en tabla si no todos los elementos de la entidad que participa con muchos tienen asociado un elemento de la entidad que participa

MODELO ENTIDAD/-RELACIÓN	MODELO RELACIONAL
Entidad	Tabla
Atributo	Columna/Campo
Identificador Único	Clave Primaria
Relaciones N:M	Nueva tabla con clave primaria la concatenación de las claves de las entidades que la forman (la relación pasa a ser una tabla, y en esa tabla se pone como C.A. las entidades que une)
Relaciones 1:M	Transformar la relación en una tabla si no todos los elementos de la entidad que participa con muchos tienen asociado un elemento de la entidad que participa con uno.

con uno.

Propagar la clave (igual que en la de 1:M) si cada elemento de la entidad que participa con muchos aparece en la entidad de uno, es decir, si TODOS los elementos de la entidad de muchos tienen asociado uno de la entidad de uno.

Ejemplo: transformar el esquema entidad/relación al modelo relacional de una tienda de antigüedades. En este caso, tenemos dos entidades (cada una

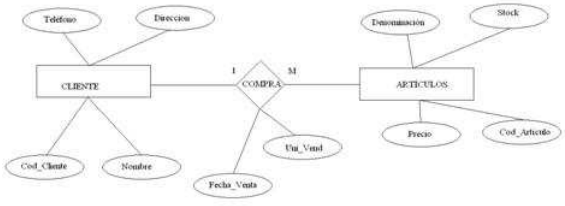


Figura 7.5: Caso de una tienda comercial.

con cuatro atributos) y una relación 1:M en la que no todos los artículos deben ser comprados por un cliente sino que ingresamos el artículo, a espera de ser comprado por un cliente, existiendo artículos en inventario que no han sido vendidos nunca. Esta relación tiene dos atributos propios de la entidad. Tal como vimos en la tabla, la solución consistirá en tres tablas al no ser una relación obligatoria, una por cada entidad (clientes y artículos) y otra para la

relación. Así, quedarían las siguientes tablas.

CLIENTE:(TELEFONO: numérico), (DIRECCION: texto), (COD.CLIENTE: numérico), (NOMBRE: texto)

ARTICULOS:(STOCK: numérico), (DENOMINACION: texto), (PRECIO: numérico), (COD.ARTICULO: numérico)

COMPRA:CP(CLIENTE:Numérico),(ARTICULO: Numerico),CP,(FECHA_VENTA: Fecha),(UNID_VENDIDAS: Numérico)

COMPRA $\longrightarrow$ CLIENTE $\longrightarrow$ CLIENTE

COMPRA $\longrightarrow$ ARTICULO $\longrightarrow$ ARTICULOS

Las claves (principales y ajenas) aparecen subrayadas y las claves de la tabla COMPRA aparecen doblemente subrayadas. ¿Cómo se pueden distinguir? A través del diagrama referencial, en donde leemos que en la tabla COMPRAS, el campo CLIENTE hace referencia a la tabla CLIENTES y en la misma tabla COMPRAS, el campo ARTÍCULOS hace referencia a la tabla ARTÍCULOS. Las claves subrayadas que no aparezcan en este diagrama referencial (COD.CLIENTE y COD.ARTÍCULO) se suponen claves principales al igual que las claves doblemente subrayadas.

Veamos otro ejemplo. Es similar al anterior, sólo que en este caso la relación es obligatoria. Vuelve a darse el caso de dos entidades y una relación 1 a mu-

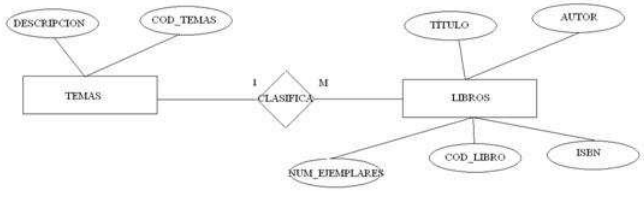


Figura 7.6: Caso de una biblioteca personal.

chos. Es el mismo caso que antes pues suponemos que todos los libros tienen un tema. Las tablas y el diagrama referencial serían: TEMAS:(DESCRIPCION: texto), (COD.TEMAS: numérico)

LIBROS:(TITULO: texto), (AUTOR: texto), (NUM_EJEMPLARES: nu-

mérico), (COD.LIBRO: numérico), (ISBN: numérico), (CA.COD.TEMAS: numérico)

LIBROS $\longrightarrow$ C.A.COD.TEMAS $\longrightarrow$ TEMAS

Veamos otro caso que cambia la relación a $1:1$

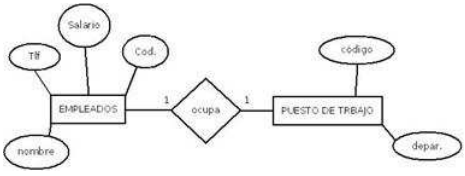


Figura 7.7: Caso recursos humanos en una empresa.

En este caso ¿Cuál de las dos entidades incrustamos en la otra? La respuesta es 'da igual'. Ésta es la solución propuesta:

EMPLEADO:(COD.EMP: numérico), (NOMBRE: texto), (TLF: numérico), (SALARIO: numérico)
PUESTO_DE_TRABAJO:(COD.TRA: numérico), (DEPT: texto), (C.A.COD.EMP: numérico)
PUESTO_DE_TRABAJO $\longrightarrow$ C.A.COD.EMP. $\longrightarrow$ EMPLEADO
Hemos insertado al EMPLEADO en la tabla PUESTO DE TRABAJO.

Finalmente, un ejemplo de una relación M:M

En caso de una relación N:M creamos una tabla independiente: ALUMNOS:(NOMBRE: Texto),(CI: Numérico),(DIRECCIÓN: Texto),(TELÉFONO: Numérico)
CURSO:(CÓDIGO: Numérico),(DENOMINACIÓN: Texto),(PROFESOR: Texto)
MATRICULA:CP(ALUMNO: Numérico),(CURSO: Numérico)CP

Conversión a tablas desde un modelo con generalización

Modelo E-R de generalización a tablas.

Dos posibilidades:

crear una tabla para el conjunto de entidades A de mayor nivel columnas (A) = atributos(A)

Para cada conjunto de entidades B de menor nivel, crear una tabla tal que:

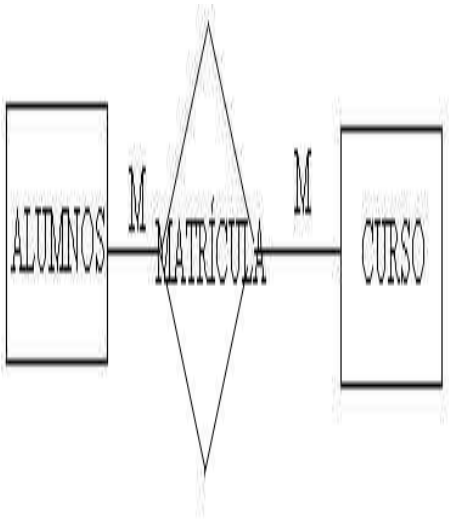


Figura 7.8: Caso estudiante y curso.

columns (B) = atributos (B) ∪ clave_primaria (A)

si A es un conjunto de entidades de mayor nivel para cada conjunto de entidades B de menor nivel, crear una tabla tal que:

columnas (B) = atributos (B) ∪ atributos (A)

La generalización se convierte al siguiente esquema:

1)
A LLP_A atribs_A
B1 atribs_B1 LLP_A
B2 atribs_B2 LLP_A
B3 atribs_B3 LLP_A
donde:
$LLP\_X es la clave primaria de la entidad X (un subconjunto de atribs_X)$
2)
B1 atribs_B1 LLP_A atribs_A
B2 atribs_B2 LLP_A atribs_A
B3 atribs_B3 LLP_A atribs_A
donde:

LLP_X es la clave primaria de la entidad X (un subconjunto de atribs_X)

Es importante mencionar que a pesar de que existen 2 métodos para convertir una generalización a tablas, no hay una regla exacta de cual usar en determinado caso. A continuación algunos consejos útiles para la determinación de cual método emplear:

- Si la entidad de nivel superior está relacionada con otra(s) entidades puede sugerirse emplear el método (1) ya que de esa manera la tabla (A) será la única involucrada en la relación, de otra forma se tendrían tres tablas (B1,B2 y B3) formando parte de la relación
- Es importante tomar en cuenta la pertenencia de instancias, si se considera que hablamos de una generalización disjunta, donde no se puede pertenecer a varias entidades de nivel inferior, quizás sea recomendable el método (1), en otro caso se podría pensar en el método (2).

Es importante analizar ambos casos con respecto a las 'consultas'que se deseen realizar ya que esto determina, en muchos casos, el método a emplear.

7.11.2. Descubrimiento de claves en las relaciones

Las claves resultantes en las relaciones de un esquema se infieren de la siguiente manera:

1. Cada tabla que provenga de una entidad contiene por si misma una clave
2. Para las tablas resultado de una relación se toman las claves primarias de ambas entidades y éstas conforman la nueva clave primaria, excepto en un caso como el que sigue:

Observamos que existe una relación m-m entre áctor'y 'serie', demostrando que un actor puede actuar en muchas series y que muchas series tendrán a los mismos actores.

La tabla que se crea es:

actor_serie si se considera íd_actor, id_serie'como la clave primaria caemos en un problema porque esa combinación no identifica de manera única a la

id_actor	id_serie	id_personaje
Joaquín Pardavé	Génesis	Abel hermano bueno
Evita Muñoz (Chachita)	Qué bonita familia	Dulce abuelita
Joaquín Pardavé	Génesis	Caín hermano malo
Evita Muñoz (Chachita)	Hermelinda linda	Bruja Hermelinda

tupla, como es el caso de 'Joaquín Pardavé, Génesis'ya que en la primer tupla tenemos que determina a Ábel hermano bueno'y en la tercera a 'Caín hermano malo'.

La relación es correcta ya que un actor puede representar a varios personajes en la misma obra, pero entonces la clave íd_actor, id_serieño es la correcta y en este caso lo más recomendable sería emplear los tres atributos de la relación íd_actor, id_serie, id_personaje'.

7.12. Dominios atómicos

Habitualmente el diseño de una BD termina en el paso del modelo E-R al relacional. No obstante siempre que se diseña un sistema, no sólo se diseña una base de datos, sino también se diseña un tipo de solución informática que determina la calidad de la misma, y si no cumplen determinados criterios de calidad, entonces hay que realizar de forma iterativa **REFINAMIENTOS** en el diseño para alcanzar la calidad deseada.

La normalización se refiere a modelar una base de datos relacional y saber cómo se genera una estructura relacional de datos válida.

Las formas normales pretenden alcanzar 2 objetivos:

1. Almacenar en la BD cada hecho sólo una vez, es decir, evitar la redundancia de datos. De esta manera se reduce el espacio de almacenamiento.

2. Que los hechos distintos se almacenen en sitios distintos. Esto evita ciertas anomalías a la hora de operar con los datos.

Normalizar una BD es transformar un conjunto de datos que tienen una cierta complejidad en su entendimiento y que su distribución en el modelo provoca problemas de lógica en las acciones de manipulación de datos.

7.13. Dependencia funcional

Una dependencia funcional es una conexión entre uno o más atributos. Por ejemplo si se conoce el valor de DNI tiene una conexión con Apellido o Nombre.

Las dependencias funcionales del sistema se escriben utilizando una flecha, de la siguiente manera:

FechaDeNacimiento —>Edad

De la normalización (lógica) a la implementación (física o real) puede ser sugerible tener éstas dependencias funcionales para lograr la eficiencia en las tablas.

7.13.1. Propiedades de la dependencia funcional

Existen tres axiomas de Armstrong:

1. Dependencia funcional reflexiva Si z.está incluido en "x.entonces x —>y

   A partir de cualquier atributo o conjunto de atributos siempre puede deducirse él mismo. Si la dirección o el nombre de una persona están incluidos en el DNI, entonces con el DNI podemos determinar la dirección o su nombre.

2. Dependencia funcional Aumentativa

   $X \longrightarrow Y$ Entonces $X \longrightarrow YZ$

   Entonces

   DNI $\longrightarrow$ nombre

   DNI, dirección $\longrightarrow$ nombre, dirección

   Si con la CI se determina el nombre de una persona, entonces con la CI más la dirección también se determina el nombre y su dirección.

3. Dependencia funcional transitiva Sean X, Y, Z tres atributos (o grupos de atributos) de la misma entidad. Si Y depende funcionalmente de X y Z de Y, pero X no depende funcionalmente de Y, se dice entonces que Z depende transitivamente de X. Simbólicamente sería:

   X $\longrightarrow$YZ entonces X $\longrightarrow$ Z

FechaDeNacimiento —>Edad

Edad $\Leftarrow$ Conducir

FechaDeNacimiento $\longrightarrow$ Edad $\longrightarrow$ Conducir

Tenemos que FechaDeNacimiento determina a Edad y la Edad determina a Conducir, indirectamente sabemos a través de FechaDeNacimiento a Conducir (En muchos países, una persona necesita ser mayor de cierta edad para conducir un automóvil, por eso se utiliza este ejemplo).

C es un dato simple (dato no primario), B es un otro dato simple (dato no primario), A es la llave primaria (PK). Decimos que C depende de B y que B depende funcionalmente de A."

7.13.2. Definición

En el diseño de esquemas de bases de datos, el concepto de dependencia funcional es vital para eliminar la **redundancia**, otros factores para ello son el manejo de dependencias multivaluadas y las restricciones de integridad referencial.

Una dependencia funcional en una relación R es un enunciado de la forma 'si dos tuplas de R concuerdan en los atributos $A_1, A_2, ..., A_n$ que tienen los mismos valores para cada uno, por lo que también deben concordar con otro atributo $B_1, B_2, ..., B_n$. Esta FD se escribe como:

$$A_1, A_2, ..., A_n \longrightarrow B \tag{7.13.1}$$

y se dice que $A_1, A_2, ..., A_n$ determina funcionalmente a B.

Por otro lado, si un conjunto de atributos $A_1, A_2, ..., A_n$ determina funcionalmente más de un atributo, tenemos:

$$A_1, A_2, ..., A_n \longrightarrow B1 \tag{7.13.2}$$
$$A_1, A_2, ..., A_n \longrightarrow B2 \tag{7.13.3}$$
$$A_1, A_2, ..., A_n \longrightarrow Bm \tag{7.13.4}$$
$$\tag{7.13.5}$$

entonces, simplemente escribimos este conjunto de FD como: $A_1, A_2, ..., A_n \longrightarrow B_1, B_2, ..., B_n$

Título	Año	Duración	Tipo	Estudio	Artista
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Estevez
Wayne's World	1992	95	color	Paramount	Dana Carvey
Wayne's World	1992	95	color	Paramount	Mike Meyers
...					

Películas(Título, Año, Duración, Tipo, Estudio, Artista)

$$Ttulo, Ao \longrightarrow Duracin \tag{7.13.6}$$
$$Ttulo, Ao \longrightarrow filmType \tag{7.13.7}$$
$$Ttulo, Ao \longrightarrow studioName \tag{7.13.8}$$
$$Ttulo, Ao - /- > starName \tag{7.13.9}$$
$$\tag{7.13.10}$$

afirmamos que: Título, Año $\longrightarrow$ Duración, Tipo, Estudio

Quizás las dependencias funcionales más evidentes sean las claves.

Decimos que un conjunto $A_1, A_2, ..., A_n$ es una clave de un relación si:

- Esos atributos determinan funcionalmente a 'todos'los demás atributos de una relación.
- No hay un subconjunto de $A_1, A_2, ..., A_n$ que determine funcionalmente a 'todos'los demás atributos (incluyendo al resto del conjunto $A_1, A_2, ..., A_n$)

La definición anterior de clave es similar a la que se mencionó anteriormente de superclave; sin embargo, las superclaves no son claves **mínimas**, recordemos que una clave primaria se escoge de entre el conjunto de superclaves mínimas. Es importante hacer notar que una clave mínima no se refiere al número de atributos incluidos, se puede tener una clave mínima ABC y otra DE donde ambas son mínimas aún cuando una de ellas sea todavía de menor tamaño que la otra.

Al conjunto de dependencia funcionales de una relación R se le denomina F.

7.13.3. Axiomas de Armstrong

- Reflexividad: sea un conjunto de atributos y entonces $\longrightarrow$ *
- Aumentación: si $\longrightarrow$ y es un conjunto de atributos entonces $\longrightarrow$
- Transitividad: si $\longrightarrow$ y $\longrightarrow$ entonces $\longrightarrow$

*Nota: De manera general una dependencia funcional de la forma –>se considera 'dependencia funcional trivial'si:

- Si al menos algún elemento **no pertenece a** se considera una dependencia no trivial.
- Si ningún elemento **pertenece a** entonces se considera una dependencia completamente no trivial

7.13.4. Reglas adicionales

- Unión: si $\longrightarrow$ y $\longrightarrow$ entonces $\longrightarrow$
- Decomposición: si $\longrightarrow$ entonces $\longrightarrow$ y $\longrightarrow$
- Pseudotransitividad: si $\longrightarrow$ y $\longrightarrow$ entonces $\longrightarrow$

7.13.5. Cerradura de un conjunto de atributos

Para un esquema R y un conjunto de atributos , si $\longrightarrow$ R entonces es superclave.

para determinar lo anterior se puede encontrar +, todos los atributos que dependen funcionalmente de:

teniendo R(A,B,C,D,E)

si A+=(A,B,C,D,E), entonces A$\longrightarrow$ R entonces A es superclave

La cerradura se calcula siguiendo el siguiente algoritmo:

```
entrada: , F
salida: +
result= while changes to result do
for each ( \longrightarrow F ) do
```

```
begin
if    result then
result=result
end
end
=
=result
result \longrightarrow (reflexividad) \longrightarrow ,
\longrightarrow (transitividad)
```

teniendo R (A,B,C,D,E,F) y F las dependencias: AB$\longrightarrow$ C, BC$\longrightarrow$ AD, D–>E, CF$\longrightarrow$ B.

Comprobar que A,B+ =A,B,C,D,E

Si A,B+ = A,B,C,D,E,F entonces afirmamos que AB es superclave, pero para ello es necesaria alguna dependencia adicional ej. AB $\longrightarrow$ CF

El calcular la cerradura es empleado para:

Verificar si es una superclave, calculando + y revisando si + contiene a todos los atributos de la relación R. Verificar una dependencia funcional $\longrightarrow$ (es decir, si pertenece a F+) checando si +. Calcular F+ (la cerradura de todo el conjunto de dependencias F en una relación R): Para cada R se calcula + y para cada elemento S + se obtiene una dependencia funcional $\longrightarrow$ S.

7.14. Normalización

7.14.1. Primera Forma Normal (1FN)

Una relación se encuentra en la primera forma normal si, y sólo si, todos los dominios simples subyacentes de los atributos contienen valores atómicos y monovalentes.

Una vez creadas las tablas hay que verificarlas y revisar si aún se puede reducir u optimizar de alguna manera.

Los problemas como la redundancia ocurren cuando se unen demasiados datos en un sola relación. Estos problemas son llamados anomalías y sus principales tipos son:

Redundancia la información se repite innecesariamente en muchas tuplas.

Título	Año	Duración	Tipo	Estudio	Artista
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Estevez
Wayne's World	1992	95	color	Paramount	Dana Carvey
Wayne's World	1992	95	color	Paramount	Mike Meyers

En la relación siguiente, Duración y Tipo.

Anomalías de actualización al cambiar la información en una tupla se descuida actualizarla en otra. Si en la relación encontramos que el Duración de StarWars es 125 podríamos cambiarlo para la primer tupla y olvidar actualizar las demás.

Anomalías de eliminación si un conjunto de valores están vacíos y se llega a perder información relacionada como un efecto de la eliminación. Si eliminamos al actor Emilio Estevez, perdemos la tupla de la película MightyDucks.

Una tabla se encuentra en 1era. NF, si todos sus atributos son atómicos (indivisibles)

El ejemplo clásico:

nombre dirección teléfono

En 1era. NF

nombre apellido_paterno apellido_materno dirección teléfono

7.14.2. Segunda Forma Normal (2FN)

Una relación se encuentra en la segunda forma normal si, y solo si, se encuentra en 1FN, y además, cada atributo que no forma parte de la clave tiene dependencia completa de la clave principal.

Es necesaria una clave compuesta, conformada por varios atributos, para verificar esta situación. En el caso de la clave primaria simple, no hay razones para analizar la Segunda Forma Normal, pues se cumple por defecto.

Ejemplo: COMPRAS (IdProducto, IdProveedor, Nombre, Cantidad, Fecha).

7.14.3. Tercera Forma Normal (3FN)

Una relación se encuentra en la tercera forma normal si, y solo si, se encuentra en 2FN; y si ningún subconjunto de atributos no claves tiene dependencia funcional entre sí.

Un esquema normalizado hasta 3FN debe cumplir con el juramento siguiente:

7.14.4. Forma Normal de Boyce-Codd (FNBC)

Una relación está en FNBC si, y solo si, todo determinante es una clave candidata. Determinante es el atributo del que depende, funcionalmente y de manera completa, otro atributo o conjunto de éstos. Una relación está en la FNBC si primero está en la 3FN y luego, si, y solo si, las únicas dependencias funcionales triviales se encuentran dadas entre la clave primaria y uno o varios atributos.

El proceso de normalización de bases de datos tiene como objetivo optimizar técnicamente el diseño de las mismas. Así, es posible minimizar redundancias y evitar anomalías relacionadas con la manipulación de los datos.

Existen distintos niveles de normalización, según diferentes autores. Sin embargo, es común encontrar que, en la práctica, llegar a la tercera forma normal es suficiente. La mayoría de la literatura, enfatiza también la cuarta forma normal, la forma normal de Boyce-Codd, y la quinta forma normal. Cada paso de una forma a la siguiente implica fragmentar la base de datos en más tablas.

7.14.5. Cuarta forma normal

La cuarta forma normal (4NF) es una forma normal usada en la normalización de bases de datos. La 4NF se asegura de que las dependencias multivaluadas independientes estén correctas y eficientemente representadas en un diseño de base de datos. La 4NF es el siguiente nivel de normalización después de la forma normal de Boyce-Codd (BCNF).

7.14.6. Características

Una tabla está en 4NF si y solo si esta en Tercera forma normal o en BCNF (Cualquiera de ambas) y no posee dependencias multivaluadas no triviales. La definición de la 4NF confía en la noción de una dependencia multivaluada. Una tabla con una dependencia multivaluada es una donde la existencia de dos o más relaciones independientes muchos a muchos causa redundancia; y es esta redundancia la que es suprimida por la cuarta forma normal.

7.14.7. Dependencia multivaluada

Sea R un esquema de relación. La dependencia multivaluada $X \rightarrow Y$ vale en R si los pares de tuplas t1 y t2 en R, tal que $t1[X] = t2[X]$ existen las tuplas t3 y t4 en R tales que:

t1 [X] = t2 [X] = t3 [X] = t4 [X]

t3 [Y] = t1 [Y]

t3 [R–X–Y] = t2 [R–X–Y]

t4 [Y] = t2 [Y]

t4 [R–X–Y] = t1 [R–X–Y]

En otras palabras se dice que: $X \rightarrow Y$, si dado un valor de X, hay un conjunto de valores Y asociados, y que este conjunto de valores de Y NO está relacionado (ni funcional ni multifuncional) con los valores de R - X -Y (donde R es el esquema), es decir Y es independiente de los atributos de R-X-Y. Una dependencia multivaluada de la forma $X \rightarrow Y$, es trivial cuando el conjunto de atributos X,Y conforma el total de los atributos del esquema.

7.15. Ventajas

- La información presente en la base de datos, ya sea sobre entidades o conexiones, es expresada en forma de tablas, lo que da al modelo una gran homogeneidad en lo que a estructura se refiere, facilita que

los operadores trabajen cómodos y que, relativamente, sean fáciles de implementar.

- Este modelo es el más utilizado en la actualidad para reducir problemas de administración de datos dinámicamente.
- En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia.
- La información es recuperada o almacenada por medio de consultas, que ofrecen una amplia flexibilidad y potencia para administrar la información.
- Reduce los datos redundantes.
- Asegura la integridad de los datos.
- Se ocupa de la seguridad de los datos.
- Los datos pueden ser accedidos concurrentemente por muchas personas.
- Soporta datos compartidos.
- Se adapta a los cambios fácilmente.

7.16. Desventajas

- Imposibilidad de representar conocimiento en forma de reglas.
- Inexistencia de mecanismos de herencia de propiedades (y por supuesto de métodos).
- Falta de poder expresivo (para representar jerarquías).
- Dificultad para gestionar datos no atómicos (los valores estructurados de una estructura de rasgos).
- Incompatibilidad entre los tipos de estructuras de datos que se transfieren o desadaptación de impedancia.

7.17. Reglas de formación de modelo relacional

Una base de datos relacional es algo más que un conjunto de datos organizados en tablas. El modelo de datos relacional está basado en teorías matemáticas consistentes como el álgebra y cálculo relacionales.

Con la intención de evitar que la potencia del modelo relacional se distorsione debido a sistemas de dudoso corte relacional, Cood definió un conjunto de reglas que un sistema de gestión de base de datos debe satisfacer para que sea considerado relacional.

Regla 1: Representación de la información Toda información almacenada en una base de datos relacional debe ser presentada de forma explícita y única a nivel lógico, por medio de valores en tablas.

Regla 2: Acceso Garantizado todo dato (valor atómico) debe ser accesible mediante la combinación de un nombre de tabla, un valor de su cable y el nombre de una columna.

Regla 3: Tratamiento sistemático de valores nulos: se ofrece el valor nulo para dar soporte a la representación de información desconocida o inaplicable de forma sistemática, independientemente del tipo de dato.

Regla 4: Catálogo dinámico en línea basado en el modelo relacional: la descripción de la base de datos se debe representar en el nivel lógico de la misma manera que los datos ordinarios, de forma que los usuarios autorizados puedan consultarla utilizando el mismo lenguaje relacional que usan para acceder a los datos normales.

Regla 5: Lenguaje de datos completo: el sistema relacional debe incluir, al menos, un lenguaje que permita expresar los siguientes elementos: definición de datos, de vistas, manipulación de datos (interactiva y programada), restricciones de integridad, autorizaciones y control de transacciones.

Regla 6: Actualización de vistas: toda vista teóricamente debe poder actualizarse en el sistema.

Regla 7: Inserciones, modificaciones y eliminaciones de alto nivel: la capacidad de manejar una relación o una vista como operando único debe existir, no solo recuperar la información, sino también en la inserción, la actualización y el borrado de datos.

Regla 8: Independencia física de los datos: los programas de aplicación y

las actividades terminales de la base de datos deben mantenerse inalterados desde el punto de vista lógico, sean cuales sean los cambios que se introduzcan en los mecanismos de almacenamiento y acceso de la base de datos.

Regla 9: Independencia lógica de los datos: los programas de aplicación y las actividades terminales de la base de datos no deben verse afectadas por aquellos cambios que preserven la información y que, desde el punto de vista teórico, estén permitidos.

Regla 10: Independencia de la integridad: las reglas de identidad de una base de datos deben ser definibles por medio del sublenguaje de datos relacional y han de almacenarse en el catalogo de la base de datos, no en los programas de aplicación.

Regla 11: Independencia de la distribución: debe existir un sublenguaje de datos que pueda soportar base de datos distribuidas sin que haya que alterar los programas de aplicación cuando se distribuyen los datos por primera vez o se redistribuyen estos con posterioridad.

Regla 12: Regla de la no subversión: si un sistema de gestión de base de datos soporta un lenguaje de bajo nivel que permite el acceso fila a fila, éste no puede utilizarse para saltarse las reglas de integridad y las restricciones expresadas por medio del lenguaje de más alto nivel.

El nombre de modelo relacional viene de la estrecha relación que existe entre el elemento básico de este modelo, y el concepto matemático de relación. Decimos que una relación R sobre los conjuntos D1, D2,.., Dn, se define como:

$$R^{D_1 \times D_2 \times ... \times D_n}$$

Donde los conjuntos D1, D2,.., Dn pueden ser cualesquiera, e incluso estar repetidos.

Los matemáticos definen las relaciones como subconjuntos del producto cartesiano de la lista de dominios.

7.18. Esquema de la Base de Datos

Cuando hablamos de la base de datos, diferenciamos entre el esquema de la base de datos, que el diseño lógico y la instancia de la base de datos, el cual es un reflejo o copia de los datos en la base de datos en un instante en el

tiempo.

El concepto de relación corresponde a la noción del lenguaje de programación de una variable, mientras que el concepto de esquema-relación corresponde a la noción del lenguaje de programación de una variable un tipo de definición.

En general, un esquema-relación consiste en una lista de atributos y sus dominios correspondientes. No buscamos la definición precisa de dominio de cada atributos hasta discutir el lenguaje SQL.

El Esquema de una Base de datos (en Inglés Database Schema) describe la estructura de una Base de datos, en un lenguaje formal soportado por un Sistema administrador de Base de datos (DBMS). En una Base de datos Relacional, el Esquema define sus tablas, sus campos en cada tabla y las relaciones entre cada campo y cada tabla.

Niveles de Esquema de Base de datos Esquema Conceptual: Un mapa de conceptos y sus relaciones. Esquema Lógico: Un mapa de las entidades y sus atributos y las relaciones. Esquema Físico: Una aplicación de un esquema lógico. Esquema Objeto: Base de datos Oracle Objeto. Resulta conveniente dar un nombre a los esquemas de las relaciones, igual que se dan nombres a las definiciones de tipos en los lenguajes de programación.

7.19. Claves

Es necesario tener una forma de especificar cómo distinguir las entidades dentro de un conjunto de entidades dado y las relaciones dentro de un conjunto de relaciones dado. Los valores de los atributos de una entidad deben ser tales que identifiquen unívocamente a la entidad.

Una clave identifica un conjunto de atributos suficiente para distinguir las entidades entre sí. Las claves también ayudan a identificar unívocamente a las relaciones y así distinguir las relaciones entre sí.

Una superclave es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar de forma única una entidad en el conjunto de entidades. Por ejemplo, el atributo id-cliente del conjunto de entidades cliente es suficiente para distinguir una entidad cliente de las otras. Así, id-cliente es una superclave. Análogamente, la combinación de nombre-cliente

e id-cliente es una superclave del conjunto de entidades cliente. El atributo nombre-cliente de cliente no es una superclave, porque varias personas podrían tener el mismo nombre.

A menudo interesan las superclaves tales que los subconjuntos propios de ellas no son superclave. Tales superclaves mínimas se llaman claves candidatas. Es posible que conjuntos distintos de atributos pudieran servir como clave candidata. Supóngase que una combinación de nombre-cliente y calle-cliente es suficiente para distinguir entre los miembros del conjunto de entidades cliente. Entonces, los conjuntos id-cliente y nombre-cliente, calle-cliente son claves candidatas.

Aunque los atributos id-cliente y nombre-cliente juntos puedan distinguir entidades cliente, su combinación no forma una clave candidata, ya que el atributo id-cliente por sí solo es una clave candidata.

Se usará el término clave primaria para denotar una clave candidata que es elegida por el diseñador de la base de datos como elemento principal para identificar las entidades dentro de un conjunto de entidades.

7.20. Lenguajes de Consulta

Un lenguaje de consulta es uno en el que un usuario solicita información de la base de datos. Estos lenguajes suelen ser de un nivel superior a los lenguajes de programación habituales. Los lenguajes de consulta pueden clasificarse como procedimentales o no procedimentales.

- En los lenguajes procedimentales el usuario instruye al sistema para que lleve a cabo una serie de operaciones en la base de datos para calcular el resultado deseado.
- En los lenguajes no procedimentales el usuario describe la información deseada sin dar un procedimiento concreto para obtener esa información.

Un lenguaje de manipulación de datos completo no sólo incluye consultas, sino también opción para la modificación de las bases de datos. Incluyen órdenes para insertar y borrar tuplas, así como para modificar partes de las tuplas existentes.

7.21. Operaciones en el modelo relacional

Podemos considerarlo como un esquema de una relación es también un conjunto, de esto se derivan algunas propiedades importantes:

- No hay orden en las tuplas.
- No hay orden en los atributos.
- No hay tuplas duplicadas.
- El esquema de toda relación incluye una clave primaria.
- Los valores que puede tomar un atributo en una relación son atómicos, en el sentido de que no tienen estructura, son escalares.

7.22. Operaciones en el Modelo Entidad - Relación

Es un modelo que ofrece una visión más textual y esquemática de cómo será la base de datos. Está basado en la lógica de predicados y en la teoría de conjuntos. Actualmente es el modelo más usado para resolver problemas reales y administrar datos dinámicamente.

Su objetivo es visualizar los objetos que pertenecen a la base de datos como entidades, junto con sus atributos y relaciones. En el modelo, se representan:

- Los datos vistos como entidades.
- Atributos o características de dichas entidades.
- Relaciones entre ellas.
- Cierta semántica del problema.
- Ciertas restricciones.

7.23. Modelo E-C-A (Evento-Condición-Acción)

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

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

En el modelo ECA una regla tiene tres componentes:

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

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

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

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

Los disparadores relacionales tienen dos niveles de granularidad: a nivel de fila y a nivel de sentencia. En el primer caso, la activación tiene lugar para cada tupla involucrada en la operación y se dice que el sistema tiene un comportamiento orientado a tuplas. En el segundo caso, la activación tiene lugar sólo una vez para cada sentencia SQL, refiriéndose a todas las tuplas invocadas por la sentencia, con un comportamiento orientado a conjuntos. Además, los disparadores tienen funcionalidad inmediata o diferida. La evaluación de los disparadores inmediatos normalmente sucede inmediatamente después del evento que lo activa (opción después), aunque también puede precederlo (opción antes) o ser evaluados en lugar de la ejecución del evento (opción en lugar de). La evaluación diferida de los disparadores tiene lugar al finalizar la transacción en donde se han activado (tras la sentencia COMMIT).

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

7.24. Caso Práctico)

Se determina que un empleado puede tomar varios cursos y un curso puede ser tomado por varios empleados. Se usa el modelo Entidad-relación. Ver Modelo

7.24.1. Segunda forma normal

Una tabla se encuentra en 2da NF, si está en 1a NF y cada atributo que NO es clave es 'completamente'dependiente de la clave.

Si tenemos la tabla:

calificaciones_cursos NO se encuentra en 2da NF ya que:

id_estudiante	depto	clave_curso	descripción	calificación

depto	clave_curso	descripción

$$id, clave, depto \longrightarrow descripcion \tag{7.24.1}$$
$$clave, depto \longrightarrow descripcion \tag{7.24.2}$$
$$\tag{7.24.3}$$

Analizando todas las dependencias funcionales:

$$id, clave, depto \longrightarrow descripcion \tag{7.24.4}$$
$$clave, depto \longrightarrow descripcion \tag{7.24.5}$$
$$id, clave, depto \longrightarrow calificacion \tag{7.24.6}$$
$$\tag{7.24.7}$$

Para realizar la normalización (2NF) la relación se descompone en:
curso estud_curso La descomposición se basa básicamente en:

- La intuición.
- Las dependencias funcionales.

Es importante que al descomponer una relación exista:

- Descomposición sin pérdida.
- Preservación de dependencias funcionales.

7.24.2. Descomposición sin pérdida

Al descomponer una relación R en varias relaciones R1 y R2 se debe verificar que no haya pérdidas, es decir, que al volver a combinar las relaciones (producto entre R1 y R2) se obtengan exactamente las mismas tuplas.
Decimos que para una descomposición en R1 y R2 no hay pérdida si:

id	depto	clave_curso	calificación

R1 R2 $\longrightarrow$ R1 F+

o bien si

R1 R2 $\longrightarrow$ R2 F+

Para el ejemplo anterior la relación

id_estudiante depto clave_curso descripción calificación

F= id,clave,depto $\longrightarrow$ descripción, clave,depto $\longrightarrow$ descripción, id,clave,depto $\longrightarrow$ calificación

tiene F+= id,clave,depto $\longrightarrow$ id,clave,depto,descripción,calificación, clave,depto $\longrightarrow$ descripción

y dicha relación se descompone en:

depto clave_curso descripción

id_estudiante depto clave_curso calificación

donde R1 R2 = depto,clave_curso donde depto,clave_curso $\longrightarrow$ descripción

y depto,clave_curso $\longrightarrow$ descripción id,clave,depto $\longrightarrow$ id,clave,depto,descripción,calificaci clave,depto $\longrightarrow$ descripción

7.24.3. Preservación de dependencias

Al descomponer una relación R en varias relaciones R1,R2,..Rn es importante revisar que se preserven las dependencias funcionales iniciales. De esta manera se garantiza que una actualización no provoque una relación inválida, verificando las FDs o bien a través de combinaciones de relaciones(productos o joins) aunque esto último no es muy eficiente.

Para ello se analizan todas la dependencias funcionales Fi para cada Ri que deben ser un subconjunto de F+

De manera que F'= F1 F2 $\cdots$ Fn

y la preservación se verifica si F'+= F+

para el ejemplo anterior teniendo:

```
F={ { id , clave , depto} \longrightarrow descripcion , {clave , depto

F+= { { id , clave , depto} \longrightarrow id , clave , depto , descripc

F1= depto , clave\_curso\longrightarrow descripcion
```

```
F2= id\_estudiante , depto , clave\_curso \longrightarrow calificac
F' = F1 F2
depto , clave\_curso\longrightarrow descripcion

id\_estudiante , depto , clave\_curso \longrightarrow calificacion

F'+= { { id , clave , depto} \longrightarrow id , clave , depto , descrip
```

y como F'+= F+ entonces si hay preservación de dependencias.

7.24.4. Forma normal de Boyce-Codd (BCNF)

Características

```
Un esquema relacional se encuentra en BCNF si para toda dependo

X $\longleftrightarrow$ A es una dependencia funcional trivial
o
X es una superclave
```

BCNF no necesariamente preserva las dependencias funcionales F'+ != F+
Algoritmo general de descomposición tratando de alcanzar BCNF

```
result= {R}
done=false
calcular F+
while (! done) do
if(existe un esquema Ri en result que no esta en BCNF) then
si \longrightarrow es una dependencia funcional no trivial en
tal que \longrightarrow Ri no esta en F+ y     = 0
result= ( result – Ri )      ( Ri –  )      (  ,  )
else
done=true
end
```

Ejemplo:

R(A,B,C,D)

B$\longrightarrow$ C
(B)+ = {CD}
B$\longrightarrow$ D

La superclave es {AB} por lo tanto no cumple con BCNF (B$\longrightarrow$ CD y B no es superclave).

Descomponiendo usando B–>CD

(A,B) (B,C,D)

Esta última en BCNF.

7.24.5. Tercera forma normal

Características

Un esquema relacional se encuentra en 3NF si para toda dependencia funcional X –>A:

X –>A es una dependencia funcional trivial

o

X es una super clave

o

A es miembro de una clave candidata de R

Lo anterior no quiere decir que una sola clave candidata deba contener a todos los atributos de A, cada atributo de A puede estar contenido en claves candidatas diferentes.

Se observa que las 2 primeras restricciones son las mismas que para BCNF pero existe una tercera que da flexibilidad a las relaciones.

Afirmamos que: 'Si una relación está en BCNF, también está 3NF; pero si una relación está en 3NF no necesariamente está en BCNF'.

Ejemplo, dada la relación:

branch-name

customer-name

banker-name

office-number

banker-name $\longrightarrow$ branch-name office-number

customer-name branch-name $\longrightarrow$ banker-name

Se observa customer-name branch-name determina al resto de los atributos así que es la superclave de R.

No está en 3NF porque:

- Las DFs no son triviales
- En la primer dependencia, 'banker-nameño es superclave de R
- Se observa que office-number y banker-name no son parte de alguna clave candidata

se descompondría en:

banker-name

branch-name

office-number

banker-name $\longrightarrow$ branch-name office-number

customer-name

branch-name

banker-name

customer-name branch-name $\longrightarrow$ banker-name

banker-name $\longrightarrow$ branch-name

Esta descomposición si está en 3NF porque:

- No hay dependencias funcionales triviales.
- En la segunda relación, la segunda DF no cumple que banker-name es superclave
- En la segunda relación, la segunda DF, branch-name es miembro de la clave candidata

customer-name, branch-name

Al cumplirse la 3er regla se confirma que la descomposición está en 3NF.

nombre_depto	extensión	id_jefe

id_empleado	nombre_depto	id_jefe

Se observa que al no cumplir con las dos primeras no está en BCNF pero gracias al relajamiento si está en 3era. NF

Otro ejemplo:

deptos nombre_depto $\longrightarrow$ extensión, jefe

empleados id_empleado $\longrightarrow$ nombre_depto, id_jefe

nombre_depto $\longrightarrow$ id_jefe

No está en 3era.NF porque:

- Las DFs no son triviales
- En la dependencia ñombre_depto$\longrightarrow$ id_jefe'de la segunda relación, ñombre_deptoño es superclave de R
- Se observa nuevamente para la segunda relación que id_jefe no es parte de alguna clave candidata

Aplicando la descomposición:

deptos nombre_depto $\longrightarrow$ extensión, jefe

empleados Esta descomposición si está en 3era.NF porque:

No hay dependencias funcionales triviales

Para toda dependencia X$\longrightarrow$ A , X es superclave.

Se observa como la relación no solo está en 3era.NF sino también en BCNF por cumplir con la segunda regla.

Algoritmo general de descomposición tratando de alcanzar 3era.NF

Forma canónica de las FDs (Fc)

La forma canónica de F es aquel conjunto mínimo de dependencias funcionales equivalentes a F, sin dependencias redundantes o partes redundantes de

nombre_depto	extensión	id_jefe

id_empleado	nombre_depto
id_empleado $\longrightarrow$ nombre_depto	

dependencias.

Para obtener la Fc se deben extraer todos los miembros éxtraños', suponga un conjunto F de dependencias funcionales y la dependencia $\longrightarrow$ está en F.

El atributo A es extraño en si A y F lógicamente implica (F - $\longrightarrow$) (- A) $\longrightarrow$

Ejemplo:

F = A $\longrightarrow$ C , AB $\longrightarrow$ C B es extraño en AB $\longrightarrow$ C porque A $\longrightarrow$ C, AB $\longrightarrow$ C lógicamente implica A $\longrightarrow$ C (el resultado de quitar B de AB $\longrightarrow$ C).

El atributo A es extraño en si A y el conjunto de dependencias (F - $\longrightarrow$) $\longrightarrow$ (- A) implica lógicamente a F

Ejemplo: F = A $\longrightarrow$ C , AB $\longrightarrow$ CD C es raro en AB $\longrightarrow$ CD porque A B $\longrightarrow$ C es inferido después de eliminar C.

Test para verificar si un atributo es extraño.

Dado un conjunto de dependencias F y $\longleftrightarrow$ está en F

Para verificar si A es extraño en calcular (- A)+ usando las dependencias en F verificar si (- A)+ contiene a , si lo hace entonces A es extraño.

Para verificar si A es extraño en:

calcular + usando sólo las dependencias en:

F'=(F - $\longrightarrow$) $\longrightarrow$ (- A)

verificar si + contiene a A, si lo hace entonces A es extraño.

Algoritmo basado en Fc

Fc: Forma canónica de las FDs

Para cada dependencia X $\longrightarrow$ Y en Fc,

crear una relación Ri (X,Y)

Si ninguna de las Ris contiene una de las superclaves de la relación,

crear Ri(X) donde X es una de las superclaves de la relación original

sid	name	street	city	zip	student

Si Ri y Rj tienen una clave en común, mezclar Ri y Rj.

Eliminar relaciones redundantes

El algoritmo anterior garantiza que en una descomposición no haya pérdida (al incluir por lo menos en una relación una de las superclaves) y que se preserven las dependencias funcionales (al incluir cada una de ellas).

Ejemplo:

Fc: sid $\longrightarrow$ name,street,city street, city$\longrightarrow$ zip zip $\longrightarrow$ city

Descomponer en 3NF

R1(sid, name, street, city), R2(zip, street, city) , R3(zip, city) - - Eliminar R3
sid
name
street
city
R1
sid$\longrightarrow$name,street,city zip
street
city
R2
street, city$\longrightarrow$zip zip $\longrightarrow$ city

BCNF vs 3NF

Como se mencionó: 'Si una relación está en BCNF, también está 3era.NF; pero si una relación está en 3era.NF no necesariamente está en BCNF'.

En la práctica, la mayoría de los esquemas en 3era.NF también están en BCNF, veamos:

(Sucursal, Cliente, Banquero)

banquero $\longrightarrow$ sucursal

sucursal, cliente $\longrightarrow$ banquero

está en 3era.NF pero no en BCNF puesto que 'banqueroño es una superclave, normalizando:

suc-banquero (sucursal, banquero)

suc-cliente (sucursal, cliente)

Nuevamente se presentan las pérdidas de dependencias.

¿Qué es mejor? ¿BCNF o 3era.NF?

- De manera general se puede decir que ambas son buenas.
- El caso ideal es conseguir BCNF sin pérdidas y con preservación de dependencias.
- Si se alcanza BCNF pero no hay preservación de dependencias se puede considerar una 3era.NF (recordando que 3era.NF debe carecer de pérdidas y debe preservar dependencias).

7.25. Caso típico

El actor principal es el responsable formativo o director de estudios (DOS, de aquí en adelante). Es la persona, en un centro de formación, encargada de la elaboración de los horarios, la gestión de las incidencias (que ahora veremos) y la asignación de profesores y aulas a los grupos que se formen.

Su necesidad es el control de todos los horarios del centro de formación. Es decir, tiene que saber dónde está cada profesor en todo momento, qué clases no se han impartido y por qué y saber qué profesores hay disponibles y qué aulas están libres en un momento determinado.

También necesita saber cuántas horas ha trabajado cada profesor en un mes y cuántas horas un profesor tenía que haber trabajado, y el motivo de las cancelaciones.

Así, el DOS recibe la petición de formar un grupo nuevo para uno de los cursos definidos, porque se han matriculado muchos alumnos y no hay plazas suficientes. El DOS entonces primero decide el horario que tendrá el grupo. En este caso, el curso para el cual hay que crear el grupo define 3 horas por semana (curso extensivo), que se impartirán Lunes, Miércoles y Viernes de

19:00 a 20:00. Con esta información definida, el DOS necesita buscar un profesor disponible en ese horario, y un aula que también lo esté. Una vez localizada la información necesaria para el horario, hay dejar anotado el nombre de los alumnos que asistirán a este grupo.

Con toda esta información definida, el grupo puede empezar. Normalmente, se le proporciona al profesor a principio de mes una hoja de asistencia, que es un informe en el que se detallan el horario de cada grupo con sus alumnos, para que el profesor pueda hacer un seguimiento de la asistencia de los alumnos y de las incidencias que puedan aparecer.

Una vez el grupo ha comenzado, y los alumnos están asistiendo a clase, al director de estudios se le pueden presentar varias situaciones a manejar:

- El grupo cambia de profesor, de horario o aula. Quizá porque el profesor se vaya del centro de formación o por la razón que sea, de vez en cuando hay que cambiar el profesor de un grupo. Sin embargo, a la hora de aplicar este cambio es importante tener en cuenta que debemos mantener la información histórica. Es decir, cuando haya un cambio de horario sabemos qué profesor había antes, cuál hay ahora, y cuándo se produjo el cambio. Es imprescindible para el cálculo de total de horas trabajadas.

- Entran alumnos nuevos, u otros dejan el grupo, cuando los alumnos se ingresan e incorporan al grupo, o cuando un alumno cambia de grupo porque el horario le viene mejor. Como en el caso anterior, tenemos que saber por qué grupos ha pasado el alumno y cuándo se ha incorporado a cada grupo.

- Una clase no se imparte, porque el profesor se ha puesto enfermo y no se ha encontrado sustituto a tiempo, o porque el centro estaba cerrado. En este caso, tenemos que saberlo, porque el número de horas trabajadas por el profesor cambia, y hay algunos casos (cuando se ha contratado un número de horas de formación concreto) en los que esa clase debe recuperarse en otro momento, ya sea otro día de la semana o alargando la duración del grupo un poco.

- Un profesor sustituye a otro. Es el caso cuando un profesor enferma o ha tomado un día libre, y otro profesor le sustituye. Así tenemos

que saber quién tenía que dar la clase (ha impartido una hora menos) y quién ha dado la clase en realidad (hay impartido una hora más).

- Un grupo se cierra. Se retiran todos los alumnos o (lo que es más común), se retiran tantos alumnos que no compensa mantener el grupo abierto, así que se reúnen dos grupos con pocos alumnos y se crea un grupo con un número de alumnos más acorde con las necesidades tanto del centro como de los propios alumnos.

7.25.1. Descripción de la entidades

Después de la descripción del proceso, detallo las entidades (tablas) que utilizamos para modelizarlo:

- Cursos: referencia a un tipo de educación formal que no necesariamente está inscripto dentro de los currículo.
- Profesores: personal que imparte las clases.
- Grupos: oferta de horarios y días para que los alumnos elijan la clase a asistir, dentro de cada curso.
- Horarios: Para cada grupo, almacenamos su horario (puede tener una estructura compleja, como: Lunes y Miércoles de 19:00 a 20:00 y Viernes de 13:00 a 14:00). Además, hay que almacenar el historial de horarios por los que ha pasado el grupo, incluyendo los profesores.
- Clases: Cada uno de los días de clase dentro del horario del grupo. Si tenemos un horario Lunes de 19:00 a 20:00, entre el 1 y el 30 de Abril, debe haber un registro para cada Lunes entre esas dos fechas, para después gestionar si las clases se han impartido o no, cada una por separado.
- Tipos de tarea: Hay situaciones en que debemos diferenciar qué se hace en los horarios del grupo. Podemos tener grupos de formación y también tener grupos que gestionen las tareas administrativas de los profesores (preparación de clases, etc.) o incluso la agenda del personal no docente, simplemente diferenciando por tipo de tarea.

- Tipos de cancelación: Necesitamos saber cuando una clase no se imparte, por lo que hará falta una gestión de motivos de cancelación, y por tanto una entidad que los defina.
- Alumnos: persona que se inscribe en el centro de formación.
- Alumnos en grupos: Tenemos que almacenar la información de qué alumnos están asignados a qué grupos y, además, nos hace falta saber, según el caso de uso, cuándo entró y cuándo salió cada alumno de su grupo.
- Asistencia: Para cada una de las clases, debemos saber qué alumnos estaban apuntados a ella, y para cada uno de ellos, si asistió a clase o no.

Gráficamente: Con PK se marcan las claves primarias, y con FK, las claves foráneas. Algunas líneas se cruzan, no lo puedo evitar. Las flechas indican que una tabla es **hija** de otra, con la punta de flecha apuntando al padre.

Sólo están indicados los campos que forman el modelo de datos, las claves primarias y las foráneas, que en cualquier caso deben estar ocultas al usuario final.

7.25.2. Campos

Como siempre, escribo los campos que considero relevantes para la descripción del modelo de datos, y no repito los campos de clave foráneas para las relaciones, que están descritos en el gráfico. La definición de cursos y grupos es: **Cursos**

- nombre: varchar(100)
- descripción: Memo. Se utiliza en el contrato que se imprime para el alumno, para hacer una descripción larga del curso en la se está matriculando. Tenemos, en lugar de tener un campo memo, tendremos una tabla separada en la que se guardan, por tipologías, distintos campos memo, que se imprimen en variados lugares del contrato.
- fechainicio: date

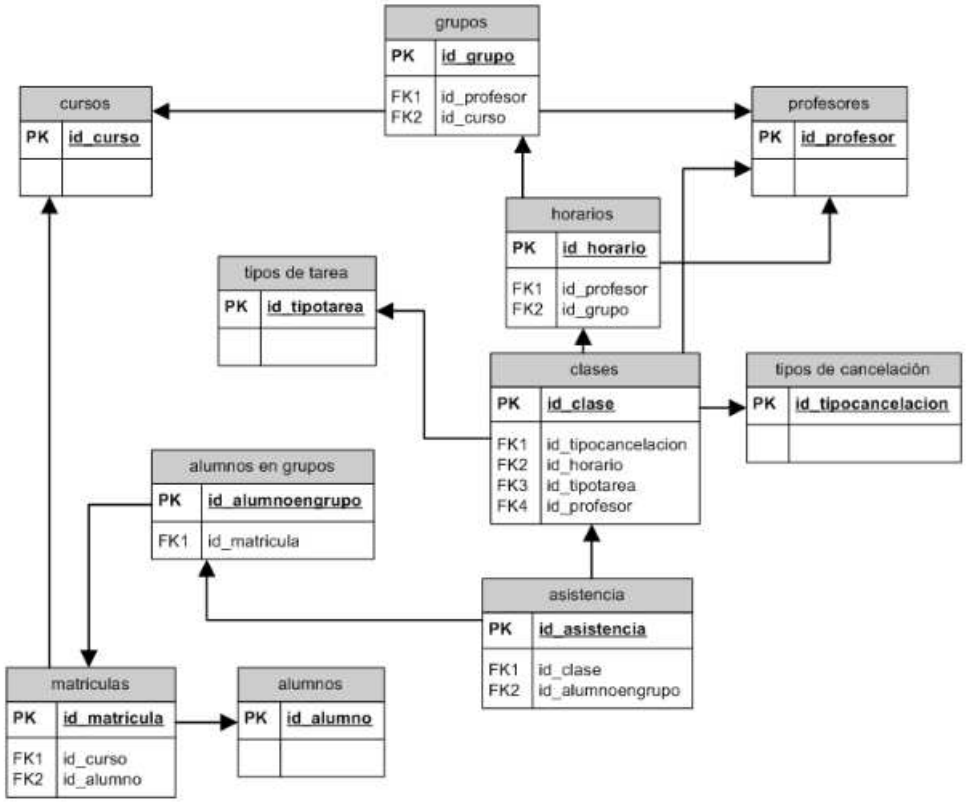


Figura 7.9: Modelo entidad-relación, caso típico

- fechafin: date

Profesores

- nombre, primerapellido, segundoapellido: varchar(100)
- nombrecompleto: es una costumbre crear un campo calculado que almacene el nombre completo del profesor. Así, las búsquedas se hacen sobre este campo y es fácil encontrar al profesor en cuestión.
- dirección: Memo

- teléfono, movil: varchar(15) el largo del campo es 12, más vale pasarse que dejarlo con 9 dígitos y descubrir que tienes que poner el +58 delante para mandar SMS después.
- CI, Pasaporte: en campos separados.

Grupos

- nombre: varchar(100)
- código: varchar(20). Es correcto tener una codificación además del nombre. En algunos sistemas lo utilizamos para guardar el código del grupo en la carrera.
- fechainicio: date
- fechafin: date. Por defecto, las del curso al que pertenece el grupo, y además estas fechas no pueden estar fuera del rango de las fechas del curso al que pertenecen.
- lugar: varchar(100) aula o salón o ubicación del grupo. En general, hacemos una gestión de aulas.
- notas: memo, del grupo
- horario: varchar(100) del grupo. En realidad, el horario se trata como una tabla por debajo de ésta.
- maximoalumnos: Integer. Máximo número de alumnos permitidos en el grupo.
- numeroalumnos: Integer. Es el número de alumnos existentes en el grupo. Este campo es de sólo lectura para el usuario, y es calculado, a través de una serie de Triggers en la base de datos, para saber el número de alumnos activos en cada grupo sin tener que estar sumando.

Horarios

- fechainicio, fechafin : date. Son las fechas del grupo con el que estamos trabajando, pero si cambiamos de horario a mitad del grupo, habrá que cerrar el horario anterior (cambiar fechafin) y crear un horario nuevo.

- L,M,X,J,V: Boolean. Hay dos formas de hacerlo, poner campos booleanos para cada día de la semana, y si el horario tiene L = sí y X = sí, entonces el horario es Lunes y Miércoles, o poner un campo **día** de tipo entero y tener un registro de horario para cada día de la semana. Los días de la semana no son algo que vaya a cambiar, y la interfase y forma de mostrar el horario es fácil de implementar e intuitivo para el usuario cuando usas una sola fila que cuando tienes 3 para el mismo horario.
- horainicio, horafin: Time
- idprofesor: por defecto, es el profesor definido del grupo, pero podemos tener la situación en la que haya dos profesores en el mismo grupo (como pasa muy a menudo con los seminarios y cursos intensivos).
- idtipotarea: Integer. Es el tipo de tarea por defecto que tendrán todas las clases del horario. También existe la posibilidad de poner un tipo de cancelación por defecto. Lo vemos más abajo, en las clases.

Clases

- fecha: Date. A estas alturas, ya no tenemos fecha de inicio y fecha de fin, esto es ya cada uno de los días de clase individuales, y por tanto lo que hay es una fecha
- horainicio, horafin: Time. Por defecto son las del horario al que pertenece la clase. Están también en la tabla de clases porque eso te permite gestionar excepciones, del tipo que la clase de hoy dura media hora más para recuperar parte de una clase anterior que se canceló, cosas así.
- idtipotarea: el tipo de tarea del horario, por defecto. Igual que antes, está replicado en esta tabla para permitir excepciones.
- idtipocancelacion: Tenemos dos formas de proceder, definimos un tipo de cancelación por defecto, que será la que tenga todas las clases (nosotros utilizamos **clase impartida**, para indicar que no ha habido cancelación), o poner el tipo de cancelación en el horario, de forma que todas las clases de ese horario tendrán ese tipo de cancelación y

luego nosotros cambiamos el tipo de cancelación para las clases que no se imparten.

- Notas pedagógicas: Memo. Suele ser buena idea poner un campo de notas pedagógicas que, el profesor, al rellenar la asistencia (lo vemos más abajo) puede completar. Esto hace que, cuando un profesor tiene que hacerse cargo de un grupo que ya ha empezado, puede leer las notas que dejó el otro profesor (lo que se llama el Class Record) y saber por dónde van y qué es lo siguiente que hay que hacer.

Tipos de Tarea

- nombre: varchar(100)
- lectiva, transporte, administrativa : boolean. Con estos campos podemos sumar después fácilmente las horas según los tipos de tarea. Podemos tener un tipo de tarea **Horas lectivas** y otra **Horas lectivas en Sábado**, que se pagan de distinta forma y queremos tener controladas, pero sin embargo todas son horas lectivas, y a la hora calcular totales de horas, las queremos agrupadas.

Tipos de cancelación

- Nombre : varchar(100)
- claseimpartida: boolean. Así podemos tener distintos tipos de cancelación que indican que la clase no se ha impartido, y luego sumar todas juntas para saber cuántas horas se han impartido y cuántas no.
- implicapagoprofesor: Según el tipo de cancelación y el convenio, hay tipos de cancelación que, aunque no cuenten como clase impartida, se pagan al profesor.
- implicacobroalumno: En el caso de los grupos para empresa o en los individuales, normalmente se factura por horas, y en estos casos se da la circunstancia de que las cancelaciones no son sólo por parte del profesor o del centro de formación: en estos casos, es posible (de hecho, es lo más común) que sea el cliente quien cancele la clase. Normalmente, se definen en este caso dos tipos de cancelación: Cancelación

en plazo, que es cuando el cliente ha avisado de que no va a darse la clase, y entonces no se le factura, o Cancelación fuera de plazo, que es el caso tan típico en el que el profesor se presenta y no hay nadie a quién darle clase. En este caso, normalmente, la clase se factura aunque no haya sido impartida, pero es importante para el DOS saber qué clases han sido canceladas de esta manera. A final de curso, cuando se evalúa el avance del alumno, es importante tener en cuenta a cuántas clases el alumno ha acudido realmente, no cuántas se le ha facturado al cliente.

Alumnos

- nombre: varchar(100). Lo mismo que en clientes, pero lo requerido es nombre y primer apellido (en clientes es sólo nombre por si
- primerapellido: varchar(100)
- segundoapellido: varchar(100)
- nombrecompleto: varchar(300):
- etc. de datos personales (profesión, teléfonos, email, etc.)

Alumnos en grupos

- fechainicio: date
- fechafin: date. Suele ser una intersección entre la duración del grupo y la matrícula, pero cuando el alumno cambia de grupo, para una matrícula puede haber varios registros de alumnos en grupos. Hay que tener en cuenta también la posibilidad de que en un mismo curso, pagando más, un alumno pueda asistir a varios grupos (esto también lo he visto).

Asistencia

- asiste: boolean. Normalmente, cuando se crean los registros de asistencia (que se crean automáticamente cuando se crean las clases), el campo asiste está inicializado a NULL, para que así sepamos que esta información todavía no está disponible. Cuando el profesor introduzca la asistencia indica si el alumno asiste o no. Mientras tanto, no se sabe.

- faltajustificada: boolean. Todo el proceso que está definido en este apartado se basa en la idea de que la asistencia de los alumnos y la cancelación de una clase son cosas separadas, porque la gestionan personas separadas. La información de cancelación de clases implica que se indique el motivo de dicha cancelación, y dicha información va a afecta a la facturación al cliente y al pago a los profesores, y por tanto es responsabilidad del DOS. La información de asistencia de los alumnos tiene valor desde el punto de vista académico (para calcular los porcentajes de asistencia de los alumnos a la hora de evaluar el rendimiento) y se usa en conjunción con la información de calificaciones. Como actualizar esta información es muy trabajoso por el volumen de datos que representa, los sistemas en general están pensados para que sean los profesores los que introduzcan esta información. Como la información de cancelaciones puede afectar directamente a sus pagos, los profesores no la introducen en el sistema. Por tanto, que todos los alumnos tengan asiste = NO no implica que la clase esté cancelada, es lo tendrá que introducir el DOS, incluyendo el motivo.

7.25.3. Disparadores y procedimientos almacenados

En el proceso previamente descrito, vemos por la definición de las tablas, que contiene muchos procesos **automáticos**, propios de la base de datos. Principalmente:

- Al crear un horario, habrá un disparador[1] que inserte en la base de datos todas las clases de ese horario, teniendo en cuenta la fecha de inicio, de finalización; así como los días de clase, incluyendo la gestión de días no lectivos según calendario.

- De la misma forma, habrá un disparador que al crear clases o incluir alumnos en grupos, inserte en la base de datos los registros de asistencia de los alumnos y las clases a las que pertenecen.

Es importante que estos dos procesos estén incluidos en la base de datos. Esta parte es el trabajo fundamental del DOS y la fuente principal de información de producción en la institución, y por tanto es imperativo que estos

[1] trigger en lenguaje técnico

procesos sean ágiles. En las primeras versiones este proceso sea hacía en el lado cliente del sistema, y el rendimiento dejaba muchísimo que desear.

7.26. Conclusiones

El modelo relacional de bases de datos se basa en un modelo formal especificado de acuerdo a la teoría de conjuntos. Una base de datos relacional es un conjunto de relaciones o tablas de la forma R($A_1, \ldots, A_n$), donde R es el nombre de la relación, que se define por una serie de atributos A_i.

Sobre las tablas relacionales se definen diferentes restricciones. La integridad de entidad es una restricción que indica que cada entidad representada por una tupla tiene que ser diferente de las demás en su relación, es decir, debe haber algunos atributos cuyos valores identifiquen unívocamente las tuplas. La integridad referencial indica que una clave foránea sólo contiene valores nulos o que existan en la relación referenciada por la clave foránea.

7.26.1. El proceso de normalización

Consiste en comprobar si el esquema original está en 1FN, 2FN y 3FN, analizando las dependencias funcionales en cada paso.

Un caso completo

En una empresa los puestos de trabajo están regulados por ley, tal que los salarios están fijos por el puesto. Creamos el siguiente esquema relacional.

EMPLEADOS(nss, nombre, puesto, salario, email)

Siendo nss la clave primaria.

nss	nombre	puesto	salario	email
111	Juan Pérez	Jefe de Área	3000	juanp@ecn.es; jefe2@ecn.es
222	José Sánchez	Administrativo	1500	jsanchez@ecn.es
333	Ana Díaz	Administrativo	1500	adiaz@ecn.es; ana32@gmail.com
...	...	...	...	...

Primera forma normal (1FN)

Una tabla está en 1FN si sus atributos son valores atómicos. En la tabla anterior, el atributo email tiene más de un valor, lo que viola a 1FN.

En general, tenemos una relación R con clave primaria K. Si un atributo M viola la condición de 1FN, hay dos opciones:

- Solución 1: duplicar los registros con valores repetidos Esta solución pasa por sustituir R por una nueva relación modificada R', en la cual:
  - Se elimina el atributo M que viola a 1FN.
  - Se incluye un nuevo atributo M' que contiene valores simples, de modo que si R'[M'] es uno de los valores de R[M], entonces R'[K] = R[K]. En otras palabras, para una tupla con n valores duplicados en M, la nueva relación tendrá n tuplas, que varían en que cada una de ellas guarda uno de los valores contenidos en M.
  - La clave primaria de R' es (K, M'), ya que podría haber valores de K repetidos, para los valores multivaluados en M.

  Siguiendo el caso, tenemos el siguiente esquema para la nueva tabla EMPLEADOS'(a) con clave primaria (nss, email):

nss	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@ecn.es
222	José Sánchez	Administrativo	1500	jsanchez@ecn.es
333	Ana Díaz	Administrativo	1500	adiaz@ecn.es
...	...	...	...	...

- Solución 2: separar el atributo que viola 1FN en una tabla. En general, esta solución pasa por:
  - sustituir R por una nueva relación modificada R'que no contiene el atributo M.
  - Crear una nueva relación N(K, M'), es decir, una relación con una clave ajena K haciendo referencia a R', junto al atributo M', que es la variante mono-valuada del atributo M.

- La nueva relación N tiene como clave (K, M').

Ahora, el esquema para la nueva tabla EMPLEADOS'(b) es:

nss	nombre	puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500
...	...	...	...

Y tenemos una nueva tabla EMAILS con clave primaria (nss, email):

nss	email
111	juanp@ecn.es
111	jefe2@ecn.es
222	jsanchez@ecn.es
333	adiaz@ecn.es
333	ana32@gmail.com
...	...

Segunda forma normal (2FN)

Un esquema está en 2FN si:

- Está en 1FN. Todos sus atributos que no son parte de la clave principal tienen dependencia funcional completa respecto de las claves existentes en el esquema. Para determinar cada atributo no clave se necesita la clave primaria completa, no una subclave.
- La 2FN se aplica a las relaciones que tienen claves primarias compuestas por dos o más atributos. Si una relación está en 1FN y su clave primaria es simple (un sólo atributo), también está en 2FN.

Por tanto, de las soluciones anteriores, la tabla EMPLEADOS'(b) está en 1FN (y la tabla EMAILS no tiene atributos no clave), por lo que el esquema está en 2FN. Sin embargo, debemos examinar las dependencias funcionales de los atributos no clave de EMPLEADOS'(a). Las dependencias funcionales son las siguientes:

- nss→nombre, salario, email
- puesto→salario

Como la clave es (nss, email), las dependencias: nombre, salario y email son incompletas, por lo que la relación no está en 2FN.

Observamos los atributos no clave que dependan como parte de la clave.

Para solucionar este problema, en los gupos de atributos con dependencia incompleta M, hacemos:

- Eliminar de R el atributo M.
- Crear una nueva relación N con el atributo M y a la parte de la clave primaria K de la que depende, la llamaremos K'.

La clave primaria de la nueva relación es K'.

Creamos una nueva relación con los atributos con dependencia incompleta:

nss	nombre	puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500
...	...	...	...

Y al eliminar estos atributos de la tabla original, obtenemos:

nss	email
111	juanp@ecn.es
111	jefe2@ecn.es
222	jsanchez@ecn.es
333	adiaz@ecn.es
333	ana32@gmail.com
...	...

Llegamos es la misma solución que en la opción para el problema de 1FN.

Tercera forma normal (3FN)

Una relación está en tercera forma normal si, y sólo si:

- está en 2FN y
- cada atributo no incluido en la clave primaria no depende transitivamente de ella.

En un esquema 2FN buscamos dependencias funcionales entre atributos que no estén en la clave.

En general, tenemos que buscar dependencias transitivas de la clave, es decir, secuencias de dependencias como la siguiente: K→A y A→B, donde A y B no pertenecen a la clave. La solución a este tipo de dependencias está en separar el(los) atributo(s) B en una tabla adicional N, y como clave primaria de N, el atributo que define la transitividad A.

Detectamos la siguiente transitividad:

- nss→puesto
- puesto→salario

Por tanto, la descomposición es:

nss	nombre	puesto
111	Juan Pérez	Jefe de Área
222	José Sánchez	Administrativo
333	Ana Díaz	Administrativo
...	...	...

En la nueva tabla PUESTOS, la clave es puesto, que también es clave foránea con referencia a la tabla EMPLEADOS. El resto de las tablas quedan igual.

Capítulo 8

Modelado de datos orientado a objetos

8.1. Introducción

Los modelos de bases de datos tradicionales (relacional, red y jerárquico) cubrieron con éxito las necesidades de las aplicaciones de gestión tradicionales en cuanto a bases de datos.

Sin embargo, existen deficiencias cuando se trata de aplicaciones complejas o sofisticadas como el diseño y fabricación en ingeniería (**CAD/CAM, CIM**), los experimentos científicos, los sistemas de información geográfica, la inteligencia artificial, los relacionales extendidos, las bases de datos deductivas o los sistemas multimedia.

La tecnología de bases de datos está en un momento de transición del modelo relacional a otros modelos. Entre éstos, se encuentran el multidimensional para sistemas **OLAP**, el semiestructurado para bases de datos **XML** de intercambio electrónico de información, el modelo dimensional para creación de **Data Warehouse** y el orientado a objetos.

En **SGBDOO**, los componentes se almacenan como objetos y no como datos, según ocurre en una base relacional cuya representación son las tablas.

Los requerimientos y las características de estas nuevas aplicaciones difieren en gran medida de las típicas de gestión:

- la estructura de los objetos es compleja,
- las transacciones son de larga duración, se necesitan nuevos tipos de datos para almacenar imágenes y textos,
- hace falta definir operaciones no estándar, específicas para cada aplicación.

Las bases de datos orientadas a objetos (**SGBDOO**) se crearon para satisfacer las necesidades de las nuevas aplicaciones arriba mencionadas, nos ofrecen flexibilidad para manejar algunos de estos requisitos y no están limitadas por los tipos de datos y por los lenguajes de consulta de los sistemas de bases de datos tradicionales.

Una característica clave de **SGBDOO** es la potencia que proporcionan al diseñador al permitirle especificar tanto la estructura de objetos complejos, como las operaciones que se aplican sobre dichos ellos. Además del creciente uso de los lenguajes orientados a objetos para desarrollar aplicaciones OO.

Las bases de datos son fundamentales en muchos sistemas de información donde las bases de datos tradicionales son difíciles de utilizar cuando las aplicaciones las acceden desde lenguajes de programación orientado a objetos como C++, Python, Smalltalk o Java.

Las bases de datos orientadas a objetos se han diseñado para integrarse con aplicaciones desarrolladas en lenguajes orientados a objetos, adoptando muchos de sus conceptos.

Los fabricantes de los **SGBDR** conocen las necesidades en el modelado de datos, por lo que sus sistemas incorporan muchas de las propuestas para las bases de datos orientadas a objetos, como caso Informix y Oracle. Dando lugar al modelo relacional extendido, y sus implementaciones se denominan sistemas objeto–relacionales.

La nueva versión de SQL, SQL:1999[1], incluye algunas de las características de la orientación a objetos.

Con la aparición de prototipos experimentales de **SGBDOO**, y sus derivados en sistemas comerciales, surgió la necesidad de establecer un modelo estándar y un lenguaje. Para ello, fue formado un grupo denominado **ODMG**

[1] Se cita como SQL3 o SQL99, por la versión anterior, SQL92; Este nombre no se utiliza para evitar el efecto 2000 en futuras versiones.

(Object Database Management Group), que propuso el estándar ODMG–93 que ha evolucionando hasta ODMG 3.0.

Para identificar el modelo de base de datos orientado a objetos, partiendo de su utilidad, usamos el esquema de la figura 8.1, con el concepto y las características de los OODBMS, para su aplicación en los sistemas informáticos.

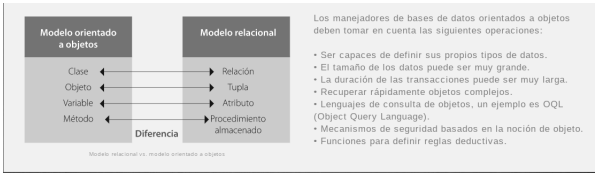


Figura 8.1: Modelo relacional vs. modelo orientado a objetos.

El uso de estándares proporciona portabilidad, lo que facilita que una aplicación se ejecute sobre sistemas distintos con mínimas modificaciones. Los estándares también proporcionan interoperabilidad, tal que una aplicación acceda a varios sistemas diferentes. Y una tercera ventaja de los estándares es que los usuarios decidan, entre distintos sistemas comerciales, qué partes del estándar utilizarán.

El Modelo de Datos Orientado a Objetos (**OODM**) es el soporte del Modelo de Base de Datos Orientada a Objetos (**OODBM**). Y en consecuencia, el Sistema de Gestión de Bases de Datos Orientado a Objetos (**OODBMS**) maneja todo esto.

Los problemas del mundo real cada vez más complejos, demandan un modelo de datos que mejor lo represente. Con el modelo de datos orientado a objetos, tanto los datos como sus relaciones están contenidos en una única estructura conocida como objeto.

Veamos, de forma esquemática, una serie de conceptos básicos relacionados con el modelo de datos orientado a objetos, para entender la importancia y uso de este tipo de sistemas.

8.2. Definiciones

La orientación a objetos representa el mundo real y resuelve problemas a través de objetos, ya sean tangibles o digitales. Este paradigma considera un

sistema como una entidad dinámica formada de componentes. Un sistema sólo se define por sus componentes y como interactúan.

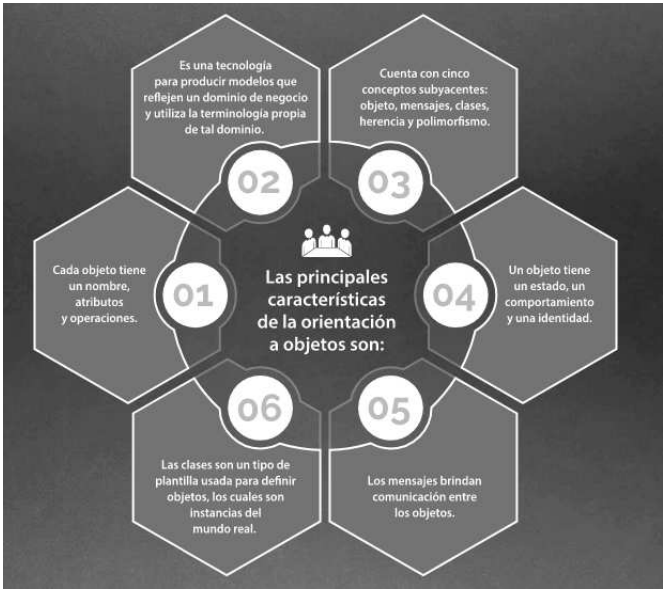


Figura 8.2: Principales características del modelo orientado a objetos.

Objetos complejos Los objetos complejos son creados a partir de objetos simples —tipos de datos, los cuales son:

- Enteros
- Caracteres
- Cadenas de bytes
- Expresiones del tipo booleano
- Números de punto flotante

Los objetos complejos son:

- Conjuntos —sets—

- Listas
- Arreglos
- Diccionarios

Un OODBMS debe tener como mínimo conjuntos —set—, listas y tuplas.

Esto tiene como ventaja que no será necesario el uso de dos lenguajes de programación para construir una aplicación; actualmente, el desarrollo de aplicaciones se hace con lenguajes de programación orientada a objetos almacenando datos en bases relacionales, por lo que el desarrollador debe utilizar un lenguaje para la aplicación (Java, PHP, C++) y otro para la base de datos (SQL), vea la figura 8.3.

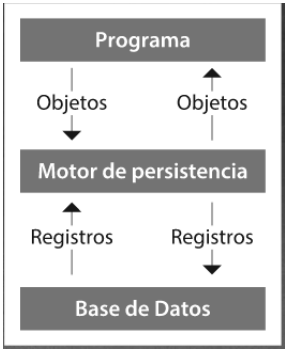


Figura 8.3: Esquema de operación.

8.2.1. Atributos

- Las clases tienen atributos que representan alguna propiedad de la clase que comparten todos los objetos de dicha clase.
- Un atributo es una propiedad nombrada de una clase, que describe un rango de valores que puede tomar esa propiedad en las instancias.
    - nombre, edad o peso son atributos del objeto Persona.
- Cada nombre de atributo es único dentro de una clase, pero cada atributo tiene un valor para cada instancia de la clase.

- Diferentes instancias de objetos pueden tener los mismos o distintos valores para un atributo dado.
- La identidad implícita del objeto lo distingue en caso de que todos los valores de los atributos sean idénticos.

- Un atributo debería ser un valor de datos puro, no un objeto.
  - Los valores de datos puros, a diferencia de los objetos, no tienen identidad.

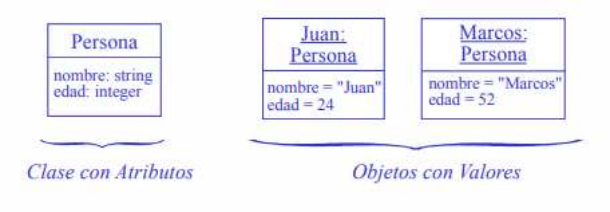


Figura 8.4: Situaciones de la vida real.

8.2.2. Identidad de objetos

Ésta, siempre ha existido en los lenguajes de programación, pero es reciente en bases de datos. El objetivo es contar con objetos que tengan una existencia independiente de sus valores. Así, dos objetos pueden ser idénticos si son el mismo objeto o ser iguales si tienen los mismos valores.

La identidad adquiere relevancia cuando un objeto se comparte con otros y cuando se actualiza. En un modelo basado en identidad, dos objetos pueden compartir un objeto hijo.

Pensemos en dos personas: Arturo y Valeria tienen un hijo llamado Jaime (cada uno tiene un hijo: Jaime, aunque Jaime es el mismo objeto). En la vida real, hay dos posibles situaciones, vea la figura 8.5:

OODBMS soporta la identidad de objetos, facilitando:

- Operaciones como asignación de objetos.
- Copiado de objetos.

Figura 8.5: Situaciones de la vida real.

- Comprobación de la identidad o igualdad de objetos.

La forma principal de implementar la identidad de objetos es mediante un objeto identificador, **OID**, independiente de los valores de los atributos del objeto y que es implementado por el sistema, lo que mejora el rendimiento.

Según Date, los OID son innecesarios e indeseables en el nivel del modelo. En comparación con las claves primarias, los OID están ocultos al usuario, mientras que aquéllas no. El uso de estos OID no elimina el uso de claves primarias, ya que son necesarias para unir al sistema con la realidad en la que se inserta; pensemos, en los documentos de facturación.

8.2.3. Encapsulación

La idea de encapsulación es tomada de los lenguajes de programación en los que todo objeto existe, vea la figura 8.6:

Una parte visible	Una parte invisible
Ésta permite especificar el conjunto de operaciones que pueden ser realizadas sobre el objeto.	Ésta contiene los datos que almacena ese objeto.

Figura 8.6: Un objeto encapsula programas y datos.

En una base de datos, un objeto encapsula (oculta) programas y datos.

En un sistema relacional, un empleado es representado por una tupla. Para ser consultado desde una aplicación, debemos usar un lenguaje para programar procedimientos que serán almacenados fuera de la base datos; es más, ese lenguaje puede ser, en realidad, la combinación de un lenguaje de alto nivel con el estándar relacional **SQL**.

En SGBDOO definimos al empleado como un objeto que tiene una parte de datos —probablemente similar al registro definido en el sistema relacional— y una parte de operaciones, como aumentodeSalario() y bajaDefinitiva() que accederían a los datos del empleado. Cuando se almacene un nuevo empleado, los datos y operaciones son guardados en la base y no fuera de ella.

Clase	Objetos	Atributos/datos
Empleado	Juan Pérez	Edad: 25
		Puesto: Psicóloga social
		Salario: 8000
	María Suárez	Edad: 23
		Puesto: Pedagoga
		Salario: 15 000

Figura 8.7: Clase, objetos y atributos/datos.

8.2.4. Herencia

Una de las herramientas disponibles en lenguajes orientados a objetos más potente es la herencia. Esta propiedad facilita a los objetos ser construidos a partir de otros objetos. El objetivo es la reutilizabilidad o reutilización, es decir, reutilizar código desarrollado.

La herencia supone una clase base y una jerarquía de clases que contiene las clases derivadas de la clase base. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos a ellas, incluso cambiar aquellos elementos de la clase base que necesita sean diferentes.

Clase Base

Clase derivada Clase derivada Clase derivada

¿Qué es la herencia?

Es un concepto similar a la herencia en la vida real, un profesor tiene atributos comunes con un alumno, como nombre, dirección, CI, y tener atributos independientes, como profesor tiene titulación y alumno tiene curso. Podríamos definir persona como CI, nombre, dirección.

Los profesores tienen un atributo que se llama **titulación** y el resto de los atributos heredados de persona.

Los alumnos heredan todos los atributos de persona pero tiene uno que se llama **curso**.

Herencia simple o jerárquica

En esta jerarquía cada clase tiene como máximo una sola superclase. La herencia simple permite que una clase herede las propiedades de su superclase en una cadena jerárquica.

Herencia múltiple o en malla

Una malla o retícula consta de clases, cada una de las cuales puede tener una o más superclases inmediatas. Una herencia múltiple es aquella en la que cada clase puede heredar métodos y variables de cualquier número de superclases.

La clase C tiene dos superclases, A y D. Por consiguiente, la clase C hereda las propiedades de las clases A y D. Evidentemente, esta acción puede producir un conflicto de nombres, donde la clase C hereda las mismas propiedades de A y D.

Herencia repetida

Otro de los problemas graves que produce la herencia múltiple es la herencia repetida. Este tipo de herencia se produce cuando una clase hereda de dos o más superclases que a su vez heredan de la misma superclase. La mayoría de los lenguajes de programación no permiten la duplicación estática de la

superclase, pero eso no se producirá siempre, y así se puede dar el caso de que el compilador duplique la clase que se hereda dos o más veces.

Jerarquía de clases

En una base de datos existen objetos que responden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y tipo. Sería inútil definir cada uno de estos objetos por separado por lo tanto se agrupan los objetos similares para que formen una clase, a cada uno de estos objetos se le llama instancia de su clase. Todos los objetos de su clase comparten una definición común, aunque difieran en los valores asignados a las variables.

Así que básicamente las bases de datos orientadas por objetos tienen la finalidad de agrupar aquellos elementos que sean semejantes en las entidades para formar un clase, dejando por separado aquellas que no lo son en otra clase.

Retomemos la relación alumno-cursa-materia agregándole la entidad maestro; donde los atributos considerados para cada uno son alumno: Nombre, Dirección, Teléfono, Especialidad, Semestre, Grupo; Maestro: Nombre, Dirección, Teléfono, Número económico, Plaza, RFC; Materia: Nombre, Créditos, Clave.

Los atributos de nombre, dirección y teléfono se repiten en la entidad alumno y maestro, así que podemos agrupar estos elementos para formar la clase Persona con dichos campos. Quedando por separado en alumno: Especialidad, semestre, Grupo. Y en maestro: Número económico, Plaza y RFC; la materia no entra en la agrupación (Clase persona) ya que la clase específica los datos de solo personas, así que queda como clase materia. La herencia tiene dos ventajas:

- Es una herramienta potente de modelado, ya que presenta una descripción precisa del mundo.
- Ayuda a simplificar la implementación de las aplicaciones.

Para entender el manejo de la herencia en los sistemas de bases de datos orientados a objetos, asumamos que tenemos empleados y estudiantes.

En un sistema relacional, el diseñador de bases de datos definiría una relación empleado y estudiante y escribiría el código para la operación de au-

mentar sueldo. Para la relación empleado, tendría que escribir el código para la operación de obtener promedio.

En un sistema orientado a objetos, usando la herencia, vemos que empleado y estudiante son personas y comparten los atributos nombre y edad. Entonces, declararíamos una clase empleado como un tipo especial de la clase persona, que incluiría una operación para aumentar Sueldo() y un atributo de salario. Similarmente, se declararía el estudiante como un tipo especial de la clase persona con el atributo adicional de conjunto de grados y la operación especial para obtener Promedio().

Figura 8.8: Empleado hereda de persona.

El modelo es más cercano a la realidad y nos ahorra código de programación. Por esto, se dice que la herencia ayuda a reutilizar código, ya que cada programa está disponible para ser compartido.

Ventajas

- Mayor capacidad de modelado.
- Ampliabilidad.
- Lenguaje de consulta más expresivo.
- Adecuación a las aplicaciones avanzadas de base de datos.
- Mayores prestaciones.

Desventajas

- Carencia de un modelo de datos universal.
- Carencia de experiencia.
- Carencia de estándares.
- Competencia. Con respecto a los SGBDR y los SGBDOR (redes).
- La optimización de consultas compromete la encapsulación.
- El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.

8.2.5. Sobreescritura y sobrecarga

En la programación orientada a objetos, tenemos la ventaja de reescribir métodos con el mismo nombre. Esto significa contar con varios métodos que se llamen igual, pero que realicen distintas operaciones. Para programar estos métodos llamados sobrecargados, es necesario que cambie algo en sus parámetros, como el número, orden o tipo de dato. Esto mismo es posible en una base de datos orientada a objetos.

8.3. Conceptos básicos

8.3.1. Definición de objeto

Un objeto en la programación orientada a objetos (**POO**) representa alguna entidad de la vida real, es decir, alguno de los objetos que pertenecen al negocio con que estamos trabajando o al problema que nos enfrentamos y con los que interactuamos. A través del estudio de ellos se adquiere el conocimiento necesario para, mediante la abstracción y la generalización, agruparlos según sus características en conjuntos.

Estos conjuntos determinan las clases de objetos con las que estamos trabajando. Primero existen los objetos; luego aparecen las clases en función de la solución que estemos buscando. Ésta es la forma de adquirir conocimiento, aunque no es la única. En ocasiones, cuando el observador es un experto

del negocio (o del problema), el proceso puede ser a la inversa y comenzar el análisis en una base teórica abstracta, sustentada por el conocimiento previo que da lugar primeramente a clases de objetos que satisfagan las necesidades de la solución.

Estos conceptos son parte de la base teórica de la idea de objeto y clases utilizados en la POO. Los objetos tienen características fundamentales para conocerlos mediante la observación, identificación y el estudio posterior de su comportamiento; estas características son:

- Identidad.
- Comportamiento.
- Estado.

En las ciencias de la computación, estrictamente en matemáticas, el término **objeto** es usado en sentido puramente matemático para referirse a cualquier **cosa**. Dicha interpretación resulta útil para discutir sobre teorías abstractas, pero no es suficientemente concreta para servir como definición de un Tipo de dato - Tipo primitivo - en discusiones más específicas, como en la programación, que está más cerca de cálculos reales y el procesamiento de información.

8.3.2. Identidad

La identidad es la propiedad que diferencia a un objeto y lo distingue de otros. Generalmente esta propiedad da nombre al objeto. Tomemos el **verde** como un **objeto** concreto de la clase **color**; la propiedad que da identidad única a este objeto es precisamente su çolor"verde. Tanto es así que para nosotros no tiene sentido usar otro nombre para el objeto que no sea el valor de la propiedad que lo identifica.

En programación la identidad de todos los objetos sirve para comparar si dos objetos son iguales o no (Principio de identidad). No es raro encontrar que en muchos lenguaje de programación la identidad de un objeto esté determinada por la dirección de memoria de la computadora en la que se encuentra el objeto, pero este comportamiento puede ser variado redefiniendo la identidad del objeto a otra propiedad.

8.3.3. Comportamiento

El comportamiento de un objeto está directamente relacionado con su funcionalidad y determina las operaciones que realiza o a las que responde ante mensajes enviados por otros objetos.

Los sistemas de bases de datos orientados a objetos son la tecnología más prometedora para los próximos años, aunque carecen de un modelo de datos común y de fundamentos formales, además que su comportamiento en seguridad y manejo de transacciones no están a la altura de los programas actuales de administradores de bases de datos, vea la figura 8.9.

Clase	Objetos	Atributos/datos
Empleado	Juan Pérez	Edad: 25
		Puesto: Psicóloga social
		Salario: 8000
	María Suárez	Edad: 23
		Puesto: Pedagoga
		Salario: 15 000

Figura 8.9: Almacenamiento de datos en un OODBMS.

La funcionalidad de un objeto está determinada, primariamente, por su responsabilidad. Una de las ventajas fundamentales de la programación orientada a objetos es la reusabilidad del código; un objeto es fácil de reutilizarse en tanto su responsabilidad sea mejor definida y concreta.

Una tarea fundamental al diseñar una aplicación informática es definir el comportamiento que tendrán los objetos de las clases involucradas en la aplicación, asociando la funcionalidad requerida por ella a las clases adecuadas.

8.3.4. Estado

El estado de un objeto se refiere al conjunto de atributos y sus valores en un instante de tiempo dado. El comportamiento de un objeto modifica su estado. Cuando una operación de un objeto modifica su estado se dice que tiene **efecto colateral**.

Esto tiene especial importancia en aplicaciones que crean varios hilos de ejecución. Si un objeto es compartido por varios hilos y en el transcurso de sus

operaciones, éstas modifican el estado del objeto, es posible que se generen errores por el hecho de que algunos de los hilos asuman que el estado del objeto no cambiará.

8.4. Características

- Persistencia.
- Concurrencia.
- Recuperación.
- Gran almacén secundario.
- Consultas.
- Abstracción.
- Encapsulación.
- Modularidad.
- Jerarquía.
- Tipos.
- Genericidad.

Según Atkinson, basado en dos criterios, un sistema, debe ser:

- orientado a objetos.
- un gestor de base de datos

En 1989 se presentó el Manifiesto de los sistemas de base de datos orientados a objetos. Este manifiesto propuso trece características obligatorias para un SGBDOO y cuatro opcionales. El creador del manifiesto, Atkinson, lo plantea como una declaración de intenciones.

Sin embargo, para garantizar que la industria de las bases de datos siga estándares para desarrollar SGBDOO son necesarias organizaciones o grupos

que completen las características que todo SGBDOO debe tener y exija su cumplimiento para favorecer la interoperabilidad entre sistemas de diferentes fabricantes.

En los SGBDR, los estándares **ISO** y **ANSI** velan por SQL. En **SGBDOO** la organización encargada de estandarizar todo lo relacionados con ellos es **ODMG** (Object Data Management Group).

Las trece características obligatorias son:

1. Almacén de Objetos complejos: los SGBDOO deben construir objetos complejos aplicando constructores sobre objetos básicos.

2. Identidad de los objetos: todos deben tener un identificador que sea independiente de los valores de sus atributos.

3. Encapsulación: los programadores sólo tendrán acceso a la interfaz de los métodos, de modo que sus datos e implementación estén ocultos.

4. Tipos o clases: el esquema de un SGBDOO incluye únicamente un conjunto de clases (o un conjunto de tipos).

5. Herencia: un subtipo o una subclase heredará los atributos y métodos de su supertipo o superclase, respectivamente.

6. Ligadura dinámica: los métodos deben poder aplicarse a diferentes tipos (sobrecarga). La implementación de un método dependerá del tipo de objeto al que se aplique. Para proporcionar esta funcionalidad, el sistema deberá asociar los métodos en tiempo de ejecución.

7. Completitud de cálculos usando el lenguaje de manipulación de datos (**DML** – Data Management Language).

8. El conjunto de tipos de datos debe ser extensible: Además, no habrá distinción en el uso de tipos definidos por el sistema y los definidos por el usuario.

9. Persistencia de datos: deben mantenerse (de forma transparente) después de que la aplicación que los creo haya finalizado. El usuario no tiene que hacer ningún movimiento o copia de datos explícita para ello.

10. Debe ser capaz de manejar gran cantidad de datos: debe disponer de mecanismos transparentes al usuario, que proporcionen independencia entre los niveles lógico y físico del sistema.

11. Concurrencia: debe tener un mecanismo de control de concurrencia similar a los sistemas convencionales.

12. Recuperación: debe tener un mecanismo de recuperación ante fallos similar a los sistemas relacionales (igual de eficientes).

13. Método de consulta sencillo: debe tener un sistema de consulta de alto nivel, eficiente e independiente de la aplicación (similar al SQL de los sistemas relacionales).

Característica	Orientado a Objetos	Objeto - Relacional
Contenido Semántico	Incrementado mediante el uso de objetos (atributos y métodos).	Incrementado mediante el tipo de datos estructurado (atributos y métodos).
Tipos de Datos Base	Se extienden mediante el uso los tipos de datos abstractos.	Se extiende mediante el uso de nuevos tipos de datos (referencia, colección y row) y definidos por el usuario.
Tipos de Datos Complejos	Objetos Complejos.	Tipos Estructurados Complejos.
Relaciones	1:M, M:N, "is a" y "extends"	1:M, M:N y "extends".
Reutilización	Mediante Herencia de Objetos.	Mediante Herencia de Tipos Estructurados y Tablas Tipadas.
Encapsulado	Igual que en la Programación Orientada a Objetos.	Igual que en la Programación Orientada a Objetos.
Portabilidad	Baja. Estándares poco aceptados.	SQL 2003.
Modelo de Datos	Sin Fundamento Teórico.	Sin Fundamento Teórico.
Curva de Aprendizaje	Elevada.	Moderada.
Integridad Referencial	Sí, Referencias Cruzadas.	Sólo con Claves Ajenas.
Comercialización	Mercado Inmaduro.	Más aceptado que el OO.
Acceso	Navegacional y Orientado a Conjuntos.	Igual que en el Modelo Relacional.
Rendimiento	Consultas eficientes gracias al uso de referencias entre objetos.	Consultas eficientes gracias al uso de referencias entre tipos de datos estructurados.
Eficiencia	Optimiza el Acceso a Disco (Agrupa los Datos Relacionados, Objetos).	Semejante al Modelo Relacional.

Figura 8.10: Comparación SGBDOO y SGDBR.

8.5. Esquema

la clase representa un conjunto de propiedades que es la estructura y comportamiento que tendrá nuestro programa por medio de objetos al que llamamos instancia de clases.

Diagrama de clases: esquema, patrón o plantilla para describir muchos casos posibles de datos. Describe clases de objetos.

Diagrama de objetos: describe cómo se relacionan un grupo particular de objetos entre sí.

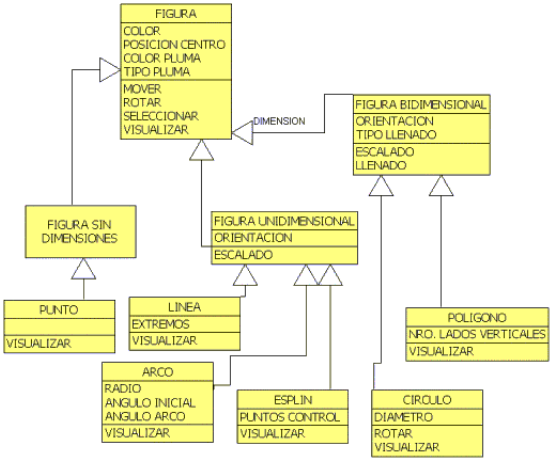


Figura 8.11: Notación de generalización y herencia.

8.6. Problemas

En la actualidad, hay mucha atención hacia los OODBMS, tanto en el terreno de desarrollo como en el teórico, no hay una definición estándar de lo que estos sistemas significan. Existen tres problemas principales que impiden una definición generalizada:

- La falta de un modelo de datos común entre los diferentes sistemas. Los sistemas de bases de datos relacionales tienen especificaciones claras dadas por Codd, pero los orientados a objetos no tienen algo así. Se pueden encontrar muchos textos que describen diferentes modelos, pero no hay uno estándar.
- La carencia de fundamentos formales. El fundamento teórico de la programación orientada a objetos es escaso en comparación con otras áreas como la programación lógica. Además, se carece de definiciones de diversos conceptos.

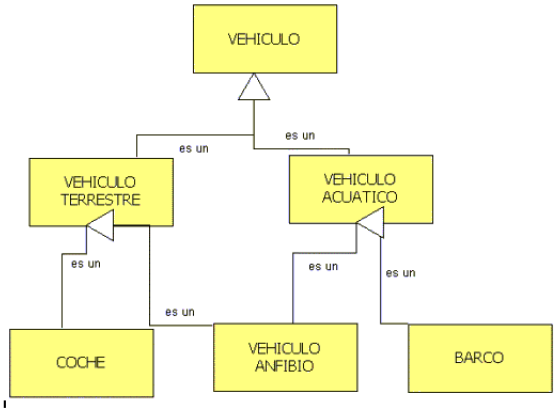


Figura 8.12: Acercamiento a la forma natural de pensar.

- Una actividad experimental muy fuerte. Existe mucho trabajo experimental, la mayoría de los desarrollos son sistemas prototipo no comerciales, por eso, no hay trabajo de conceptualización y definición de estándares. El diseño de estos sistemas está orientado por las aplicaciones que los requieren y no por un modelo común.

El problema de estos sistemas es similar al de las bases de datos relacionales a mitad de los setenta. La gente desarrollaba implementaciones en vez de definir las especificaciones para luego hacer la tecnología para implementarlas. Se espera que de los prototipos y desarrollos actuales de los OODBMS surja un modelo. Aunque también se corre el riesgo de que alguno de éstos se convierta en el estándar por su demanda en el mercado. A manera de definición, decimos que un OODBMS satisface dos criterios:

- Debe ser un DBMS. Por lo que debe incluir características como persistencia, administración de almacenamiento secundario, concurrencia, recuperación y facilidad de consultas personalizadas.
- Debe ser un sistema orientado a objetos, consistente con los lenguajes de programación orientada a objetos.
- Criterio con características que se comparten con la programación orientada a objetos: objetos complejos, identidad de objetos, encap-

sulación, herencia, sobreescritura y sobrecarga y completa capacidad computacional —computational completeness—.

8.7. Limitaciones

Las debilidades y limitaciones de los SGBDOO son:

- Pobre representación de las entidades del "mundo real".
- Sobrecarga y poca riqueza semánticas.
- Soporte inadecuado para las restricciones de integridad y empresariales.
- Estructura de datos homogénea.
- Operaciones limitadas.
- Dificultades para gestionar las consultas re cursivas.
- Desadaptación de impedancias.
- Problemas asociados a la concurrencia, cambios en los esquemas y el inadecuado acceso navegacional.
- No ofrecen soporte para tipos definidos por el usuario (sólo dominios).

8.8. Transformación

8.9. Lenguaje

Los administradores de bases de datos relacionales cuentan con un lenguaje para realizar procesos computacionales sobre los datos: el SQL. Además, adicionan un lenguaje procedimental que permite la definición de variables, manejo de excepciones, ciclos y estructuras condicionales. Algunos de estos lenguajes son:

- pl/sql para Oracle

- pl/pgsql para PostgreSQL
- Transact-SQL para SQL Server de Microsoft

Los administradores de bases de datos orientadas a objetos también deben contar con un lenguaje que puede realizar cualquier procesamiento. En este sentido, lo más común es que los SGBDOO integren lenguajes computacionalmente completos dentro de la base de datos. Estos pueden ser los que ya existen en el mercado y que se usan como lenguajes aplicación general (Java, C++, etc.).

Algunas bases de datos orientadas a objetos han sido diseñadas para trabajar con lenguajes de programación orientados a objetos, **LPOO**, tales como Delphi, Ruby, Python, Perl, Java, Visual Basic.NET, etc.

A partir de 2004, la base de datos orientada a objetos ha vuelto a experimentar un crecimiento debido al surgimiento de bases de datos orientadas a objetos de código abierto. Son fáciles de usar y asequibles. Están totalmente escritas en lenguaje de programación orientado a objetos, como Smalltalk, Java o C.

8.10. Manipulación

ODMG está compuesto por:

Modelo de Objeto En el modelo de objetos ODMG tanto los diseños como las implementaciones son portables entre los sistemas que lo soportan. Dispone de las siguientes primitivas de modelado:

- Los componentes básicos de una base de datos orientada a objetos son los objetos y los literales.
- Un objeto es una instancia auto contenida en una entidad de interés del mundo real.
- Los objetos tienen algún tipo de identificador único.
- Un literal es un valor específico, como **Amparo** o **36**. Los literales no tienen identificadores. Un literal no tiene que ser necesariamente un sólo valor, puede ser una estructura o un conjunto de valores relacionados que se guardan bajo un único nombre.

- Los objetos y literales se categorizan en tipos.
- Cada tipo tiene un dominio específico compartido por todos los objetos y literales de ese tipo.
- Los tipos también pueden tener comportamientos.

Cuando un tipo tiene comportamiento, todos los objetos de ese tipo comparten el mismo comportamiento. En el sentido práctico, un tipo puede ser una clase de la que se crea un objeto, una interfaz o un tipo de datos para un literal (como **integer**. Un objeto se piensa como una instancia de un tipo. Lo que un objeto sabe hacer son sus operaciones.

Cada operación puede requerir datos de entrada (parámetros de entrada) y puede devolver algún valor de un tipo conocido. Los objetos tienen propiedades, que incluyen sus atributos y las relaciones que tienen con otros objetos. El estado actual de un objeto viene dado por los valores actuales de sus propiedades. Una base de datos es un conjunto de objetos almacenados que se gestionan de modo que puedan ser accedidos por múltiples usuarios y aplicaciones.

La definición de una base de datos está contenida en un esquema creada por el lenguaje de definición de objetos **ODL** (Object Definition Language) que es el lenguaje de manejo de datos definido como parte del estándar propuesto para las bases de datos orientadas a objetos.

8.10.1. Lenguaje de definición de objeto ODL

Es un lenguaje de especificación para definir tipos de objetos para sistemas compatibles con ODMG. ODL es el equivalente del **DDL** (lenguaje de definición de datos) de los SGBD tradicionales. Define los atributos y las relaciones entre tipos, y especifica la signatura de las operaciones.

La sintaxis de ODL extiende el lenguaje de definición de interfaces (IDL)de la arquitectura **CORBA** (Common Object Request Broker Architecture).

8.10.2. Lenguaje de Consulta de objetos OQL

Es un lenguaje declarativo del tipo de SQL para realizar consultas de modo eficiente sobre bases de datos orientadas a objetos, incluyendo primitivas de

alto nivel para conjuntos de objetos y estructuras. Está basado en SQL-92, proporcionando un súper conjunto de la sintaxis de la sentencia SELECT.

OQL no tiene primitivas para modificar el estado de los objetos.

La sintaxis básica de OQL es una estructura SELECT . . . FROM . . . WHERE . . . , como en SQL.

8.11. PostgreSQL y la orientacion a objetos

El argumento a favor de las bases de datos objeto-relacionales sostiene que facilita una migración gradual de sistemas relacionales a los orientados a objetos y, en algunas circunstancias, coexisten ambos tipos de aplicaciones durante algún tiempo.

El problema de este enfoque es que no es fácil lograr la coexistencia de dos modelos de datos diferentes como son la orientación a objetos y el modelo relacional. Es necesario equilibrar de alguna manera los conceptos de uno y otro modelo para que NO entren en conflicto.

Uno de los puntos fundamentales en la orientación a objetos es la clase. Existen dos enfoques para asociar el concepto de clase con el modelo relacional, las clases definen los tipos de:

1. tablas.
2. columnas.

Dado que en el modelo relacional, las columnas están definidas por tipos de datos, lo natural es hacer corresponder las columnas con las clases. PostgreSQL implementa los objetos como tuplas y las clases como tablas. También es posible definir nuevos tipos de datos mediante los mecanismos de extensión.

Dado que las tablas son clases, se definen como herencia de otras. Las tablas derivadas son polimorfas y heredan los atributos (columnas) de la tabla padre (incluida su clave primaria). Se deben manejar con precaución, porque las tablas polimorfas pueden conducir a errores de integridad al duplicar claves primarias.

PostgreSQL soporta algunas extensiones del lenguaje SQL para crear y gestionar este tipo de tablas.

8.12. Tablas

Las tablas creadas en PostgreSQL incluyen, por defecto, varias columnas ocultas que almacenan información acerca del identificador de transacción en que pueden estar implicadas, la localización física del registro dentro de la tabla (para localizarla rápidamente) y, lo importante, el **OID** y **TABLEOID**. Éstas, están definidas con un tipo de dato especial llamado identificador de objeto OID que se implementa como un entero positivo de 32 bits. Cuando se inserta un nuevo registro en una tabla se le asigna un número consecutivo como OID, y el TABLEOID de la tabla que le corresponde.

En la programación orientada a objetos, el concepto de OID es importante, ya que se refiere a la identidad propia del objeto, lo hace único de los demás objetos.

Para observar las columnas ocultas, hacemos referencia a ellas específicamente en el comando select:

```
demo=# select oid, tableoid, * from persona;
oid |tableoid |nombre |direccion
-----+---------+-----------------------+-
17242 | 17240 | Alejandro Magno | Babilonia
17243 | 17240 | Federico Garcia Lorca | Granada 65
(2 rows)
```

OID de PostgreSQL presenta algunas deficiencias:

- Todos los OID de una base de datos se generan a partir de una única secuencia centralizada lo que provoca, en bases de datos con mucha actividad de inserción y eliminación de registros, que el contador de 4 bytes se desborde y suministrar OID ya entregados. Esto sucede, por supuesto, con bases de datos grandes.
- Las tablas enlazadas mediante OID no tienen ventaja al utilizar operadores de composición en términos de eficiencia respecto a una clave primaria convencional.
- Los OID no mejoran el rendimiento. Son, en realidad, una columna con un número entero como valor.

Los desarrolladores de PostgreSQL proponen la siguiente alternativa para usar OID de forma segura:

- Crear una restricción de tabla para que el OID sea único, al menos en cada tabla. El SGBD incrementa secuencialmente el OID hasta encontrar uno sin usar.
- Usar la combinación OID - TABLEOID si se necesita un identificador único para un registro válido en toda la base de datos.

Por los motivos anteriores, no es recomendable el uso de OID hasta que nuevas versiones de PostgreSQL lo corrijan. En caso de usarlos, conviene seguir las recomendaciones anteriores.

Es posible crear tablas que no incluyan la columna OID mediante la siguiente notación:

```
create table persona (
nombre varchar(30),
direccion varchar(30)
)without oids;
```

8.12.1. Herencia

PostgreSQL ofrece como característica particular la herencia entre tablas, tal que una tabla herede las características de otra previamente definida, según la definición de herencia que hemos visto.

Retomemos la tabla persona definida como sigue:

```
create table persona (
nombre varchar (30),
direccion varchar (30)
);
```

A partir de esta definición, creamos la tabla estudiante como derivada de persona:

```
create table estudiante (
demo(# carrera varchar(50),
demo(# grupo char,
```

```
demo(# grado int
demo(# ) inherits ( persona );
CREATE
```

En la tabla estudiante se definen las columnas carrera, grupo y grado, pero al solicitar información de la estructura de la tabla observamos que también incluye las columnas definidas en persona:

```
demo=# \d estudiante
Table "estudiante"
Column |Type | Modifiers
-----------+-------------------------+-----------
nombre | character varying(30) |
direccion | character varying(30) |
carrera | character varying(50) |
grupo | character(1) |
grado | integer |
```

En este caso, a la tabla persona la llamamos padre y a la tabla estudiante, hija.

Cada registro de la tabla estudiante contiene 5 valores porque tiene 5 columnas:

```
demo=# insert into estudiante values (
demo(# 'Juan Rulfo' ,
demo(# 'Treboles 21',
demo(# 'Ingenieria en Computacion',
demo(# 'A',
demo(# 3
demo(# );
INSERT 24781 1
```

La herencia no sólo facilita que la tabla hija contenga las columnas de la tabla padre, sino que establece una relación conceptual es-un.

La consulta del contenido de la tabla estudiante muestra, por supuesto, un sólo registro. Es decir, no se heredan los datos, únicamente los campos (atributos) del objeto:

```
demo=# select * from estudiante;
```

```
nombre |direccion |carrera |grupo | grado
-------+----------+--------+------+-------
Juan Rulfo | Treboles 21 | Ingenieria en Computacion | A | 3
(1 row)
```

Además, la consulta de la tabla persona muestra un nuevo registro:

```
demo=# select * from persona;
nombre | direccion
-----------------------+-----------
Federico Garca Lorca | Granada 65
Alejandro Magno | Babilonia
Juan Rulfo | Treboles 21
(3 rows)
```

El último registro es el insertado en la tabla estudiante; sin embargo, la herencia define una relación conceptual en la que un estudiante es-una persona. Por lo tanto, al consultar cuántas personas están registradas en la base de datos, se incluye en el resultado a los estudiantes. Para consultar únicamente a las personas utilizamos el modificador ONLY:

```
demo=# select * from only persona;
nombre | direccion
-----------------------+-----------
Alejandro Magno | Babilonia
Federico Garcia Lorca | Granada 65
(2 rows)
demo=#
```

No es posible borrar una tabla padre si no se borran primero las tablas hijo.

```
demo=# drop table persona;
NOTICE: table estudiante depende de table persona
ERROR: no se elimina table persona porque otros objetos depende
HINT: Use DROP ... CASCADE para eliminar ademas los objetos dep
```

Como es lógico, al eliminar la fila del nuevo estudiante insertado, se elimina de las dos tablas. Tanto si lo eliminamos desde la tabla persona, como desde la tabla estudiante.

8.12.2. Herencia y OID

Los OID establecen la diferencia en los registros de las tablas, aunque sean heredadas: nuestro estudiante tiene el mismo OID en las dos tablas, ya que se trata de una única instancia de la clase estudiante:

```
demo=# select oid,* from persona ;
oid |nombre | direccion
-------+-----------------------+-------------
17242 | Alejandro Magno | Babilonia
17243 | Federico Garcia Lorca | Granada 65
17247 | Juan Rulfo | Treboles 21
(3 rows)
demo=# select oid,* from estudiante ;
oid |nombre |direccion |carrera |grupo | grado
-------+--------+-----------+-----------------------------------+-----
17247 | Juan Rulfo | Treboles 21 | Ingenieria en Computacion |
(1 row)
```

Dado que no se recomienda el uso de OID en bases grandes, y debe incluirse explícitamente en las consultas para examinar su valor, es conveniente utilizar una secuencia compartida para padres y sus descendientes si se requiere un identificador.

En PostgreSQL, una alternativa para no utilizar OID es crear una columna de tipo serial en la tabla padre, así es heredada en la hija. El tipo serial define una secuencia de valores que se incrementa de forma automática, y por lo tanto constituye una buena forma de crear claves primarias, al igual que el tipo $AUTO_INCREMENT$ en MySQL.

```
demo=# create table persona (
demo(# id serial,
demo(# nombre varchar (30),
demo(# direccion varchar(30)
demo(# ) without oids;
NOTICE: CREATE TABLE will create implicit sequence 'persona_id_
NOTICE: CREATE TABLE / UNIQUE will create implicit index 'perso
```

La columna id se define como un entero y se incrementa utilizando la función nextval() tal como indica la información de la columna:

```
demo=# \d persona
Table "persona"
Column |Type |Modifiers
---------+-------------------------+-----------------------------------------------------
id| integer |not null default nextval('"persona_id_seq"'::text)
nombre|character varying(30)|
direccion|character varying(30)|
Unique keys: persona_id_key
```

Al definir un tipo serial, creamos una secuencia independiente de la tabla y consultamos las secuencias de nuestra base de datos mediante el comando ds:

```
demo=# \ds
List of relations

Schema |Name |Type |Owner
---------+-------------------------+----------+----------
public | productos_clave_seq | sequence | postgres
(1 row)
```

Nuevamente, creamos la tabla estudiante heredando de persona:

```
create table estudiante (
demo(# carrera varchar(50),
demo(# grupo char,
demo(# grado int
demo(# ) inherits ( persona );
CREATE
```

El estudiante hereda la columna id y se incrementa utilizando la misma secuencia:

```
demo=# \d persona
Table "persona"
Column |Type |Modifiers
---------+-------------------------+----------
id| integer |not null default next-val('"persona_id_seq"'::text
nombre|character varying(30)|
direccion|character varying(30)|
carrera |character varying(50)|
```

```
grupo | character(1)|
grado | integer|
```

Insertamos en la tabla algunos registro, omitiendo el valor para la columna id:

```
demo=# insert into persona(nombre,direccion)
values ('Federico Garcia Lorca', 'Granada 65');
demo=# insert into persona(nombre,direccion)
values ('Alejandro Magno', 'Babilonia');
demo=# insert into estudiante(nombre,direccion,carrera,grupo,gr
```

La tabla estudiante contiene un sólo registro, pero su identificador es el número 3.

```
demo=# select * from estudiante;
id |nombre |direccion |carrera |grupo |grado
---+---------+----------+----------+------+------
3 | Elizabeth | Pino 35 | Psicologia | B | 5
(1 row)
```

Todos los registros de persona siguen una misma secuencia sin importar si son padres o hijos:

```
demo=# select * from persona;
id |nombre |direccion
---+-----------------------+-----------
1 | Federico Garca Lorca | Granada 65
2 | Alejandro Magno | Babilonia
3 | Elizabeth | Pino 35
(3 rows)
```

La herencia es útil para definir tablas que conceptualmente mantienen elementos en común, pero también requieren datos que los hacen diferentes. Uno de los elementos que conviene definir como comunes son los identificadores de registro.

8.12.3. Persistencia en el modelo orientado a objetos

La persistencia es una característica necesaria de los datos en un sistema de bases de datos. Recordemos que consiste en la posibilidad de recuperar datos

en el futuro. Esto implica que los datos se almacenan sin considerar que el programa de aplicación continúa o no. En resumen, todo administrador de base de datos establece la persistencia a sus datos.

En el caso de los sistemas de gestión de base de datos orientada a objetos (OODBMS), la persistencia implica almacenar los valores de atributos de un objeto con la transparencia necesaria para que el desarrollador de aplicaciones no tenga que implementar ningún mecanismo distinto al mismo lenguaje de programación orientado a objetos.

8.13. Caso típico UML

Elementos de un caso de uso:

- Conjunto de secuencias de acciones, cada secuencia representa un posible comportamiento del sistema.
- Actores, se tratan de los roles que pueden jugar los agentes que interactúan con el sistema.. Los roles son jugados por personas, dispositivos, u otros sistemas. Podríamos distinguir entre actores primarios, para los cuales el objetivo del caso de uso es esencial y actores secundarios, que interactúan con el caso de uso, pero cuyo objetivo no es esencial.
- Variantes, son versiones especializadas, un caso de uso que extiende a otro o un caso de uso que incluye a otro.

Como veremos a continuación, en los diagramas de casos de uso se muestran: casos de uso (representados en forma de elipses), actores (en forma de personajes) y relaciones (en forma de líneas y/o flechas). UML define cuatro tipos de relaciones en los diagramas de casos de uso:

- Comunicación: Relación (asociación) entre un actor y un caso de uso. El estereotipo de la relación de comunicación es: **communicate** aunque generalmente no se estipula ningún nombre, como podemos apreciar en el siguiente ejemplo de comunicación, ver figura 8.13.
- Inclusión: Un caso de uso base incorpora explícitamente el comportamiento de otro caso de uso en algún lugar de su secuencia. La relación

Figura 8.13: Acercamiento a la forma natural de pensar.

de inclusión sirve para enriquecer un caso de uso con otro y compartir una funcionalidad común entre varios casos de uso, también puede utilizarse para estructurar un caso de uso describiendo sus subfunciones. El caso de uso incluido existe únicamente con ese propósito, ya que no responde a un objetivo de un actor. Estas relaciones se representan mediante una flecha discontinua con el estereotipo **include**. Algunos casos de uso típicos de inclusión son: comprobar, verificar, buscar, validar, autentificar o login... En principio, no deberíamos abusar de este tipo de relación, para no hacer una descomposición funcional del sistema. A partir de UML 1.3 la relación **include** reemplazó al denominado **uses**. Veamos un ejemplo de inclusión entre casos de uso, ver la figura 8.14:

Figura 8.14: Acercamiento a la forma natural de pensar.

- Extensión: Un caso de uso base incorpora implícitamente el comportamiento de otro caso de uso en el lugar especificado indirectamente por este otro caso de uso. En el caso de uso base, la extensión se hace en una serie de puntos concretos y previstos en el momento del diseño, llamados puntos de extensión, los cuáles no son parte del flujo principal. La relación de extensión sirve para modelar: la parte opcional del sistema, un subflujo que sólo se ejecuta bajo ciertas condiciones o varios flujos que se pueden insertar en un punto determinado. Este tipo

de relación produce confusión y no debería utilizarse en exceso. Conviene su uso sólo para insertar un nuevo comportamiento no previsto en un caso de uso existente. Estas relaciones se representan mediante una flecha discontinua con el estereotipo **extend**. Veamos la figura 8.15 : Aquí usamos la relación de extensión entre los casos de uso Abrir, ac-

Figura 8.15: Acercamiento a la forma natural de pensar.

ción de mejora, y Resolver, consulta. En este caso tenemos el punto de extensión "resolución retrasada" (en el caso de uso Resolver consulta) debido a que cuando haya pasado un tiempo estipulado por la organización (como 3 días laborales) se abre una acción de mejora para dejar constancia del retraso y, posteriormente, realizar las acciones pertinentes, de ahí que digamos que el caso de uso Abrir, acción de mejora, es una subfunción de uso que puede extender al caso de uso Resolver, consulta.

- Especialización y generalización de los casos de uso: Un caso de uso (subcaso) hereda el comportamiento y significado de otro, es decir las relaciones de comunicación, inclusión y extensión del super-caso de uso. En muchas ocasiones este super-caso de uso es abstracto y corresponde a un comportamiento parcial completado en el subcaso de uso. O dicho de otra manera, Los casos de uso 'hijo'son una especialización del caso de uso 'padre'. En la medida de lo posible debe evitarse puesto que produce cierta confusión en algunas ocasiones. Veamos un ejemplo de generalización en la figura 8.16:

Como vemos pueden existir vínculos de generalización o herencia entre actores. Los actores especializados (Abogado y Psicólogo) heredan los casos de uso del actor general (Profesional). La punta de flecha apunta al actor más general. Hay que reseñar que los actores especializados pueden tener otros casos de uso propios que no estarán disponibles para los demás actores. Este

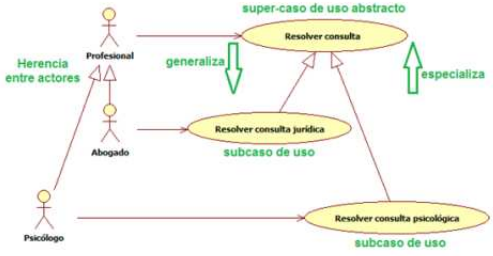


Figura 8.16: Acercamiento a la forma natural de pensar.

tipo de herencia entre actores si que se usa frecuentemente puesto que nos simplifica considerablemente el diagrama, nos ahorra un número importante de relaciones de comunicación entre actores y casos de uso y nos sirve para esclarecer visualmente la jerarquía entre actores del sistema.

Apéndice A

El modelo L5

Es un modelo orientado al desarrollo de aplicaciones para Internet. No es una derivación del modelo de desarrollo **MVC**.

Es una división completa de las partes que integran el proceso de realización de cualquier proyecto orientado a Internet en el cual, la división la constituye las ramas de tecnologías que intervienen en una aplicación orientada a Internet. Así, el modelo se divide en cinco grandes áreas:

- Modelado de Datos.
- Lógica de Programación.
- Estructura de Contenido.
- Diseño de la Interfaz.
- Maquetación e Implementación.

Estos cinco puntos son el sustento del modelo L5. Adicionalmente, cada punto tiene un grado de división a ser propuesto por el equipo de desarrollo, dependiendo de la complejidad del proyecto.

Por motivos del alcance de este libro no está incluidas la fase de análisis y la definición de metas necesarias para que estos puntos se lleven a cabo; son la base sobre la que descansa cualquier proyecto de desarrollo.

La justificación de separar el modelo en cinco puntos, es bastante simple. Consideremos el desarrollo tradicional de un proyecto sencillo como un sitio de una pequeña empresa con un administrador de contenidos.

A.1. Trabajando con un CMS

Lo común es que usemos una implementación de base de datos (MySQL, Oracle, SQL Server), relacionada con el contenido y accedida a través del lenguaje de scripting implementado (PHP, ASP.Net, Java, Ruby).

Debemos contar con un diseño de interfaz y el producto final es la página vista en el explorador (HTML, CSS, JavaScript).

Existen varias tecnologías involucradas en el producto final, todas con una tarea específica. Este modelo es aplicable en cualquier nivel de definición y complejidad de un proyecto.

Nuestro punto de inicio se centra en la definición del modelo de la base de datos y del modelo de relaciones de los datos. Esta filosofía se enfoca en el hecho de que separar cada uno de los puntos mencionados proporciona la dinámica que debe tener una aplicación.

Si en el apartado de la definición de la base de datos, relegamos por completo el manejo del almacenamiento y las funciones de gestión de datos al SGBD, sea cual sea, nos garantizamos un rendimiento absoluto y un máximo aprovechamiento.

También, el punto a destacar es que cada uno de los elementos y tecnologías que integran el modelo puede ser reemplazado sin que esto afecte a los procesos subsiguientes.

A.2. La programación adaptada a módulos

Para la programación seleccionamos tecnologías que se adapten a cada módulo a desarrollar. Con esto aprovechamos al máximo el uso de una tecnología en una parte del esquema y utilizamos diferentes tecnologías dentro del proyecto, con lo cual, cada módulo se hace de una manera óptima y sencilla por que estamos utilizando lo mejor para su desarrollo.

A.3. La estructura de contenido

Es el punto de inflexión dentro del desarrollo. En esta fase generamos un esquema de modelado de contenido. Es decir, en el desarrollo de la base de datos, optimizamos el manejo de la información, en el área de programación optimizamos los procesos y construimos el punto de unión entre el contenido y el almacenamiento.

A.4. En la programación

No debemos generar consultas completas a la base de datos (sino llamadas a funciones y vistas almacenados en el SGBD). Las salidas deben estar en un formato estándar entre tecnologías como XML.

El contenido debe estar en un formato estándar. Esto facilita su manipulación en cualquier dispositivo.

A.5. El diseño gráfico de la interfaz

Es la definición de plantillas para el diseño del sitio y el último punto es la maquetación general del sitio con la integración y la interpretación del contenido generado. Este aspecto también es reemplazable. En si, todos los puntos son reemplazables y escalables.

A.6. Diseño inicial

- MySQL
- PHP
- XML

Diseño a Tres Columnas

- XHTML 1

Reemplazar por:

- SQL Server
- PHP
- XML

Diseño a Tres Columnas

- XHTML 1

Actualización:
Sin cambiar toda la estructura en una nueva, quedamos como:

- SQL Server
- ASP.Net
- XML
- HTML 5

Es un cambio radical, pero el modelo sigue bastante intacto, como al principio.

A.7. Ventajas

Son muchas las ventajas de realizar un proyecto basado en esta metodología. El modelo es utilizable en proyectos sencillos y complejos. Simplemente bastará con saber identificar de manera clara a los actores dentro del proceso y definir las funciones.

Muchas soluciones de terceros pueden ser agregadas al proyectos en el nivel correspondiente sin afectar el desarrollo de los demás niveles. La combinación de tecnologías es una gran ventaja.

Algunas de las soluciones óptimas para nuestro proyecto, si son de terceros, podrían no estar disponibles en nuestro lenguaje de programación principal, por lo cual su utilización sería imposible.

Pero, si el proyecto no lo requiere o contamos con todos los elementos disponibles, no debemos utilizar múltiples tecnologías dentro del proyecto.

Lo importante del desarrollo es que el producto final cumpla con las necesidades requeridas. Sabemos que la optimización es importante. El modelo L5, hace énfasis en ello. Al final de cada desarrollo, se pretende que construimos una estructura de programación estable y sobre todo reutilizable.

Podemos dividir los desarrollos de sub-módulos en clases, módulos y paquetes completos con características propias. Siempre, la estructura es interdependiente. El mayor reto es crear una base sólida multinivel.

A.8. Desventajas

Las desventajas aparentes están dentro del proyecto en si. El utilizar múltiples tecnologías, demanda un mayor conocimiento de ellas por parte del grupo de desarrolladores, lo que útil en aplicaciones escalables.

La metodología demanda que el equipo esté conformado por profesionales con amplio conocimiento en esas tecnologías: desde la implementación de la base de datos hasta la maquetación.

El otro punto consiste en que, posiblemente, necesitemos mayores recursos para diseñar un proyecto. Podríamos tener un proyecto publicado en algún servidor compartido, y esto sería desventajoso, debido el grado de personalización y requerimientos que podríamos alcanzar dentro de la aplicación.

A.9. Oportunidades

Todas las metodologías, ofrecen un sin número de métodos. Algunas son únicas para cada método.

Entre ellos destacan un mejor control dentro del proyecto, sencillez de actualización y del proceso de depuración y mantenimiento, y que, preferiblemente, al final de cada desarrollo, contaremos con mejores herramientas, reutilizables en posteriores proyectos o en la escalabilidad de antiguos desarrollos.

A.10. Retos

Al ser profesionales en algún método de desarrollo, siempre el cambio de perspectiva, causa cierto grado de incertidumbre y malestar. Es normal. Uno

de los retos para nuevos emprendedores, será la tentación de volver a las viejas prácticas. El sistema demandará cierto grado de disciplina.

Si no nos guiamos dentro de los parámetros de desarrollo, podríamos dañar el proceso. Tal vez, no lo veamos dentro del producto final, pero en un futuro, tendremos serios problemas dentro de la estructura de la aplicación. La optimización es un proceso laborioso, pero debemos recordar que, al final, este punto es vital para cualquier aspecto.

Recordemos que las tecnologías en el Internet cambia constantemente.

Realizar una aplicación que sea óptima para el presente y que sujeta de grandes cambios, dentro del ciclo de funcionamiento, puede ser un gran reto, pero ciertamente, como equipo o como profesional independiente, nos brindará más de una satisfacción.

Apéndice B

Diagramas ER

B.1. Componentes y características

Los diagramas ER se componen de entidades, relaciones y atributos. También, representan la cardinalidad que define las relaciones en términos de números. A continuación, un glosario:

B.2. Entidad

Se define como una persona, objeto, concepto u evento, que teene datos almacenados acerca de este. Piensa en las entidades como sustantivos. Como un cliente, estudiante, auto o producto. Por lo general, se muestran como un rectángulo.

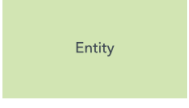


Figura B.1

Tipo de entidad Un grupo de cosas que se pueden definir, como estudiantes o atletas, mientras que la entidad sería el estudiante o atleta específico. Otros ejemplos son clientes, autos o productos.

Conjunto de entidades Es igual que un tipo de entidad, pero se define en un momento determinado, como por ejemplo estudiantes que se inscribieron en una clase el primer día. Otros ejemplos son clientes que realizaron una compra en el último mes o autos registrados actualmente en Florida. Un término relacionado es una instancia, en la que una persona determinada o un auto específico podría ser una instancia del conjunto de entidades.

Categorías de entidades Las entidades se clasifican en fuertes, débiles o asociativas. Una **entidad fuerte** se define únicamente por sus propios atributos, en cambio, una **entidad débil** no. Una **entidad asociativa** es aquella que relaciona entidades (o elementos) dentro de un conjunto de entidades.

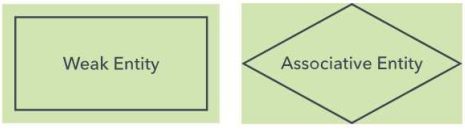


Figura B.2

Claves de entidad Se refiere a un atributo que únicamente define una entidad en un conjunto de entidades. Las claves de entidad se dividen en superclave, clave candidata o clave primaria. **Superclave** un conjunto de atributos (uno o más) que juntos definen una entidad en un conjunto de entidades. **Clave candidata** es una superclave mínima, es decir, contiene el menor número posible de atributos para seguir siendo una superclave. Un conjunto de entidades puede tener más de una clave candidata. **Clave primaria** es una clave candidata seleccionada por el diseñador de la base de datos para identificar únicamente al conjunto de entidades. **Clave foránea** identifica la relación entre ellas.

B.3. Relación

Cómo las entidades interactúan o se asocian entre sí. Piensa en las relaciones como verbos. El estudiante mencionado podría inscribirse en un curso. Las dos entidades son el estudiante y el curso, y la relación representada es el acto de inscribirse, que conecta ambas entidades de ese modo. Las relaciones se muestran, por lo general, como diamantes o etiquetas directamente en las

líneas de conexión.

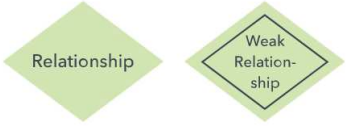


Figura B.3

Relación recursiva la misma entidad participa más de una vez en la relación.

B.4. Atributo

Una propiedad o característica de una entidad. A menudo se muestra como un óvalo o círculo.

Figura B.4

Atributo descriptivo una propiedad o característica de una relación (frente a una entidad).

Categorías de los atributos los atributos se clasifican en simples, compuestos y derivados, así como de valor único o de valores múltiples. **Simples** significa que el valor del atributo es mínimo y no se divide, como un número de teléfono. **Compuestos** los subatributos surgen de un atributo. **Derivados** los atributos se calculan o derivan de otro, como la edad que se calcula a partir de la fecha de nacimiento.

Figura B.5

Valores múltiples denotan más de un valor del atributo, como varios números de teléfono para una persona.

Figura B.6

Valor único contienen sólo un valor de atributo. Los tipos se pueden combinar, puede haber atributos de valor único simples o atributos de múltiples valores compuestos.

B.5. Cardinalidad

Define los atributos numéricos de la relación entre dos entidades o conjuntos de entidades. Las tres relaciones cardinales principales son uno a uno, uno a muchos y muchos a muchos. Un ejemplo de uno a uno es un estudiante asociado a una dirección de correo electrónico. Un ejemplo de uno a muchos (o muchos a uno, en función de la dirección de la relación) es un estudiante que se inscribe en muchos cursos, y todos esos cursos se asocian a ese estudiante en particular. Un ejemplo de muchos a muchos es que los estudiantes en grupo están asociados a múltiples miembros de la facultad y, a su vez, los miembros de la facultad están asociados a múltiples estudiantes.

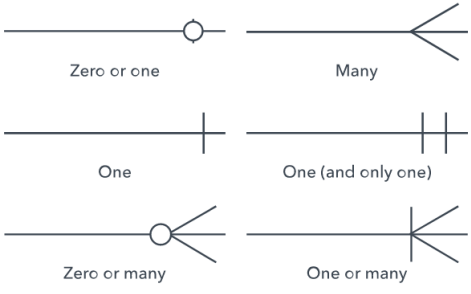


Figura B.7

Vistas de cardinalidad: la cardinalidad puede estar del lado opuesto o del mismo, en función de dónde se muestran los símbolos.

Restricciones de cardinalidad: Los números máximos o mínimos que se aplican a una relación.

B.6. Creación de mapas de lenguaje natural

Los componentes ER reflejan las categorías gramaticales, eso fue lo que hizo Peter Chen. Esto muestra cómo un diagrama ER se compara con un diagrama gramatical:

- Sustantivo común: tipo de entidad. Ejemplo: estudiante.
- Sustantivo propio: entidad. Ejemplo: Sally Smith.
- Verbo: tipo de relación. Ejemplo: se inscribe (por ej. en un curso, que podría ser otro tipo de entidad).
- Adjetivo: atributo de una entidad. Ejemplo: principiante.
- Adverbio: atributo de una relación. Ejemplo: digitalmente.

B.7. Símbolos y notaciones de diagramas ER

Hay numerosos sistemas de notación que son similares, pero que se diferencian en algunos aspectos específicos.

B.7.1. Estilo de la notación de Chen

Hay quien los llama esquemas entidad/relación relacionales por su similitud con los esquemas entidad/relación.

Lo cierto es que intentan representar todo lo que los esquemas conceptuales son capaces de representar y adaptarlo a las premisas del Modelo Relacional.

Sin embargo, la mayoría no son capaces de representar lo mismo que el modelo entidad/relación de Chen, la razón estriba en que el modelo relacional no tiene tantas formas de relación. Por ello el modelo entidad/relación debe de seguir siendo una referencia que implica crear restricciones apropiadas para reflejar la profundidad de las relaciones del modelo conceptual

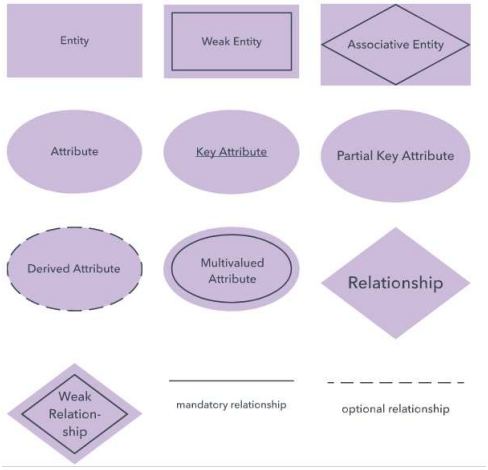


Figura B.8

B.7.2. Estilo de la ingeniería de la información, notación de Martin y notación patas de gallo

Quizá la forma más popular en las herramientas CASE para representar esquemas relacionales es la notación de patas de gallo (crow's foot es el término con que se conoce en inglés) utilizado en diversas metodologías y herramientas de trabajo como la notación Barker (utilizada en gran medida por la propia empresa Oracle), en la metodología SSADM[1], en la metodología Information Engineering (Ingeniería de la Información) y en otras metodologías y notaciones formales. Además está presente en la mayoría de herramientas CASE.

B.7.3. Estilo de la notación de Bachman

En este esquema las flechas representan cardinalidades n y los círculos cardinalidades de tipo cero. Las de tipo uno no tienen ningún símbolo asignado. A este tipo de diagramas (los de flechas y ceros) se les llama diagramas en notación Bachman.

[1] Structured systems analysis and design method, Análisis de sistemas estructurado y método de diseño

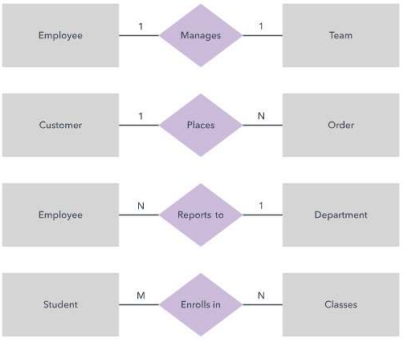


Figura B.9

B.7.4. Estilo de la notación de IDEF1X

Es una metodología desarrollada originalmente para el Bank of America y ha sido utilizada para modelar datos en la Fuerza Aérea Norteamericana, desde 1985. En el lenguaje de definición de la BD, se especifican tanto la parte gráfica como la sintáctica de la estructura de la BD. Originalmente esta metodología se comenzó a usar en ambientes mainframe, pero evolucionó para soportar BD en plataformas cliente-servidor. Muchas herramientas CASE utilizan su nomenclatura para representar las entidades, relaciones y atributos en el modelo Entidad-Relación.

Los componentes de la metodología IDEF1X son:

1) Entidades	2) Relaciones	3) Atributos/Claves
a) Independiente	a) Identificadora	a) Atributo
b) Dependiente	b) No-identificadora	b) Clave Primaria
	c) De Categoría	c) Clave Alterna
	d) No-específica	d) Clave Foránea

En la notación IDEF1X, un círculo sólido indica la clave externa de la relación, la tabla secundaria. Si no se necesita esta última para tener una principal, se muestra un rombo vacío en la parte principal de la relación.

Las relaciones no identificadoras se dibujan mediante una línea con guiones; de esta forma, se indica las que tienen una línea sólida.

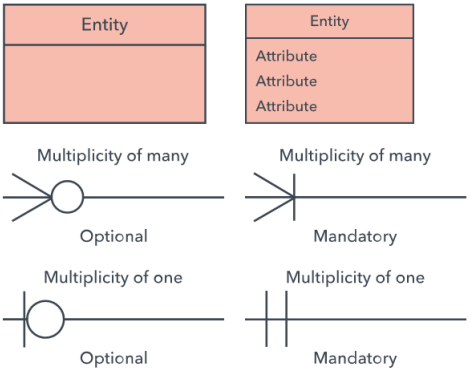


Figura B.10

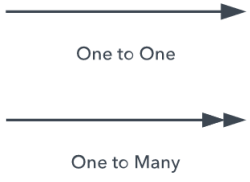


Figura B.11

B.7.5. Estilo de la notación de Barker

En el diagrama B.12 se puede examinar un modelo sencillo estilo pata de gallo. En estos diagramas la cardinalidad máxima n se dibuja con las famosas patas de gallo, la cardinalidad mínima de tipo cero con un círculo y la cardinalidad de tipo uno con una barra vertical. El hecho de que suministros y existencias tengan las esquinas redondeadas es para remarcar que representan relaciones entre entidades (no siempre se remarca).

En cualquier caso tampoco hay un estándar unánimemente aceptado para este tipo de notación.

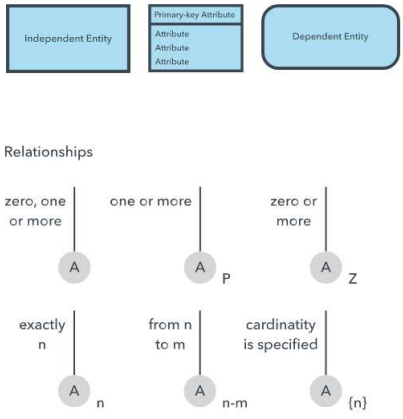


Figura B.12

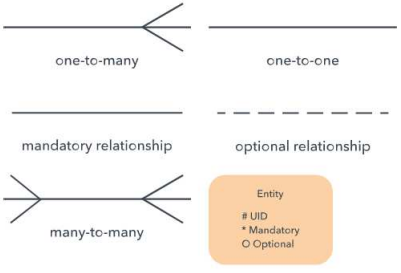


Figura B.13

B.8. Relaciones ISA

En el caso de las relaciones ISA, se siguen estas normas:

1. Cada entidad de las relaciones ISA (sin importar si es super o subentidad) se convierte en una tabla que contendrá cada uno de los atributos de la entidad (en el caso de que la ISA sea de tipo total, se podría incluso no hacer una tabla para la superentidad y pasar todos sus atributos a sus subentidades, pero no es recomendable porque dificulta el trabajo en la base de datos y además refleja peor este tipo de relación).

2. Si las subentidades no tienen clave propia, se colocará como clave, la clave de su superentidad. Esta clave heredada será clave externa (foreign key), además de clave principal.

3. En el caso de que las subentidades tengan clave principal propia. Se colocará en las subentidades el identificador de la superentidad como clave externa que además será clave alternativa.

4. Que la relación ISA sea exclusiva o no, no cambia el esquema relacional, pero sí habrá que tenerlo en cuenta para las restricciones futuras en el esquema interno (casi siempre se realizan mediante triggers), ya que en las exclusivas no puede haber repetición de la clave de la superentidad en ninguna subentidad (por ello si es posible se sigue dibujando el arco en este esquema).

5. No varía el resultado porque la relación ISA sea total o parcial; el modelo relacional no tiene capacidad para marcar esa posibilidad. Pero es interesante tenerlo en cuenta (se suele añadir un comentario al diseño relacional para posibles restricciones, nuevamente mediante triggers, a crear)

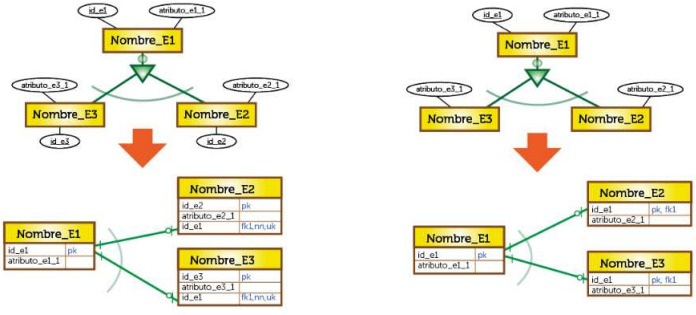


Figura B.14

La figura B.14. Entidad débil cuyo identificador contiene el de la entidad fuerte. En este caso se usa esa columna como clave externa (no se requiere añadir de nuevo el identificador de la tabla principal)

B.9. Ejemplos

A continuación unos ejemplos de diagramas ER en cada sistema.

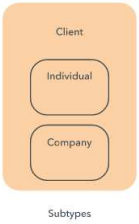


Figura B.15

B.10. Cómo dibujar un diagrama ER básico

Propósito y alcance: definen el propósito y el alcance de lo que estás analizando o modelando.

Entidades identifican las entidades involucradas. Cuando estés listo, comienza a dibujarlas en rectángulos (o en la figura que selecciones en tu sistema) y etiquétalas como sustantivos.

Relaciones determinan cómo se relacionan todas las entidades. Dibuja líneas entre ellas para indicar las relaciones y etiquétalas. Algunas entidades pueden no estar relacionadas, y eso está bien. En diferentes sistemas de notación, la relación se puede etiquetar en un diamante, otro rectángulo o directamente sobre la línea de conexión.

Atributos brindan más detalles mediante la adición de atributos clave de las entidades. Los atributos a menudo se muestran como óvalos.

Cardinalidad muestra si la relación es 1-1, 1-muchos o muchos a muchos.

B.11. Consejos sobre diagramas ER

1. Muestra el nivel de detalle necesario para tu propósito. Tal vez requieras dibujar un modelo físico, lógico o conceptual, en función de los detalles necesarios. (Consulta las descripciones de esos niveles).

2. Presta atención a las relaciones o entidades redundantes.
3. Si estás solucionando un problema de una base de datos, presta atención a los vacíos en las relaciones o los atributos o entidades que faltan.
4. Asegúrate que todas tus entidades y relaciones estén etiquetadas.
5. Puedes convertir tablas relacionales a diagramas ER, y viceversa, si eso te ayuda a alcanzar tu objetivo.
6. Asegúrate de que el diagrama ER admita todos los datos que necesitas guardar.
7. Puede haber diferentes enfoques válidos para un diagrama ER. Mientras brinde la información necesaria para su alcance y propósito, es apropiado.

Bibliografía

Bibliografía

[1] ATKINSON, M. P. (1989) *Manifiesto de los Sistemas de Bases de Datos al Objeto puras.*

[2] BAKER. *Véase http://www.entitymodelling.org/*

[3] BURCH, J., STRATER, F. R. (1981) *Sistemas de información : teoría y práctica, Editorial Limusa , México,*

[4] CHAMBERLAIN, H., BOICE., (1974) *Language SEQUEL 2. IBM.*

[5] CHEN, P. (1976) *Modelo entidad-relación: hacia una visión unificada de los datos*

[6] CHU, S., (2006) *Introducing databases by in Conrick, M. Health informatices: transformista cacheteare with technology, Thomson.*

[7] CODASYL, (1971) *Informe CODASYL DBTG (Data Base Task Group).*

[8] COOD, E.F. (1970) *A relational model of data for large shared data banks. In: Communications of the ACM archive. Vol 13. Issue 6(June 1970)*

[9] CODD E. F., (1971 *Normalized Data Structure: A Brief Tutorial. IBM Research Report RJ. San José. California.*

[10] CONNOLLY, T.; BEGG, C. (2005) *Sistemas de bases de datos : un enfoque práctico para diseño, implementación y gestión.*

[11] DATE, C. J. (1999) *When's an extension not an extension?. Intelligent Enterprise.*

[12] DAVENPORT, T., PRUSAK, L. (1999) *CONOCIMIENTO EN ACCIÓN. COMO LAS ORGANIZACIONES MANEJAN LO QUE SABEN, Editorial Fundamentos.*

[13] DE MIGUEL, A. PIATTINI, M. (1999) *Fundamentos y Modelos de Bases de Datos. Ra-Ma.*

[14] DE MIGUEL, A. PIATTINI, M. (1999) *Concepción y Diseño de Bases de Datos Relacionales. Ra-Ma.*

[15] DELOBEL, C. ADIBA, M., () *Bases de Datos y sistemas relacionales.*

[16] ELMASRI, R.; NAVATHE, S. (2002) *Fundamentos de Sistemas de bases de datos. Addison-Wesley.*

[17] LUCAS GÓMEZ, A. () *Diseño y Gestión de Sistemas de Bases de Datos. Paraninfo.*

[18] MURDICK, R, (1988) *Sistemas de Información Administrativa, Prentice-Hall, México.*

[19] OBRIEN, JAMES (2007) *Sistemas de Información Gerencial, McGraw-Hill / Interamericana de México.*

[20] SILBERSCHATZ, A. - KORTH, H. F. - SUDARSHAN, S. () *Fundamentos de Bases de Datos.*

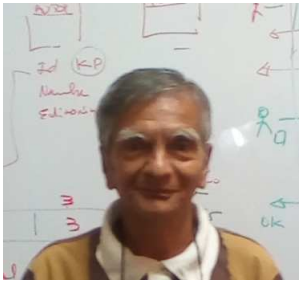
[21] STONEBRAKER, M. (1990) *Manifiesto de los SGBD de Tercera Generación.*

[22] TSICHRITZIS, D., KLUG, A C., (1978) *The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Dabatase Management Systems. Univ. de Toronto. Canadá.*

[23] ULLMAN, J.D. WIDOM, J. (1999) *Introducción a los Sistemas de Bases de Datos. Prentice Hall.*

[24] ZHUGE, H. (2008) *The Web Resource Space Model. Web Information Systems Engineering and Internet Technologies Book Series 4. Springer.*

Acerca del autor



Graduado en Física, Facultad de Ciencias, Universidad Nacional Autónoma de México (UNAM), Doctor en Ciencias de la Computación, Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas (IIMAS) UNAM. Cursante del Doctorado en Minería de Datos, Modelos y Sistemas Expertos por la Universidad de Illinois en Urbana-Champaings (USA).

Ha sido profesor en la Facultad de Química y en la Facultad de Ingeniería, UNAM. Fue profesor en el Departamento de Ciencias Básicas, Universidad de Las Américas en Cholula, Puebla (UDLAP) y, durante seis años, profesor visitante en la Universidade Federal de Rio Grande do Sul (Brasil), profesor y jefe de sistemas postgrados de agronomía y veterinaria, Universidad Central de Venezuela (UCV), actualmente es profesor del Departamento de informática y del Departamento de Postgrado, Universidad Politécnica Territorial de Aragua (UPT Aragua), Venezuela.

Expositor y conferencista a nivel nacional e internacional.

Es asesor en mejora de procesos, gestión de proyectos, desarrollo de software corporativo en los sectores de servicios, banca, industria y gobierno.

El Dr. Domínguez es un especialista reconocido en base de datos, desarrollo de software y servidores en el área del software libre, así como un experto en LINUX DEBIAN.

En la actualidad orienta su trabajo a la creación y desarrollo de equipos de software de alto desempeño. Autor de múltiples artículos y libros sobre la materia.