Diseño de Bases de Datos

Introducción

Una base de datos es un sistema que consta de una **colección de datos** *grande*, *persistente*, *integrada* y *dinámica* y que además proporciona determinadas operaciones para describir, establecer, manipular y acceder a esos datos.

- **Colección de datos**: conjunto de valores de interés junto con sus relaciones.
- En cuanto a los adjetivos:
  - *Grande*: El almacenamiento primario de un único ordenador no es suficiente para mantener todos los datos.
  - *Persistente*: Los datos "sobreviven" a las aplicaciones que operan sobre ellos.
  - *Integrada*: Los datos son de interés para diferentes usuarios. Es necesario establecer mecanismos de acceso simultáneo.
  - *Dinámica*: Los datos cambian con el tiempo: las operaciones de usuario típicas son la inicialización, actualización y recuperación de datos. Resulta complejo mantener la integridad de los datos por:
    * Introducción de información incorrecta.
    * Violación de las restricciones impuestas en el diseño de la base de datos.

Sistemas de gestión de bases de datos

Un sistema de gestión de bases de datos (*Data Base Management System*, DBMS) es la entidad que da soporte a una base de datos.

Proporciona:

- **Lenguajes:**
  - Lenguaje de definición de datos (*Data-Definition Language*, DDL): permiten a los administradores definir los datos y metadatos.
  - Lenguajes de manipulación: permiten a los usuarios introducir, acceder y manipular los datos.
- **Servicios:** gestionan los problemas causados por el volumen de datos y su naturaleza dinámica, integrada y persistente.
  - Las estructuras de almacenamiento y los optimizadores que proporcionan acceso eficiente a grandes volúmenes de datos.
  - Mecanismos de recuperación frente a fallos que protegen los datos cuando se producen fallos del *hardware* o del *software*.
  - Mecanismos de control de acceso concurrente que permiten a varios usuarios acceder y actualizar la base de datos simultáneamente.
  - Sistemas de control de las restricciones que mantienen la integridad de los datos.

Aplicaciones de bases de datos

Una *aplicación de base de datos* consiste en una base de datos junto con la aplicación o el conjunto de aplicaciones coordinadas que se ejecutan en un sistema de gestión de bases de datos (DBMS). Una aplicación de base de datos almacena sus datos en una base de datos y utiliza un sistema de base de datos para recuperar, actualizar, proteger y mantener la integridad de los datos.

¿Qué tipo de aplicaciones son las que nos interesan?

- Durante mucho tiempo: aplicaciones fundamentalmente orientadas al proceso de gestión.
- La tendencia en la actualidad es: extender las aplicaciones de bases de datos para que puedan formar parte del proceso de toma de decisiones, prospección de mercados, planificación e incluso de actividades de control...

¿Cual es el ámbito de aplicación de las bases de datos?

- Procesos de diseño – El diseño de objetos físicos (edificios, aviones, etc...) y objetos no físicos (*software*, por ejemplo).
- Sistemas de conocimiento.
- Aplicaciones multimedia.
- Aplicaciones *World Wide Web* – Motores de búsqueda, aplicaciones de *data mining*.
- Aplicaciones evolutivas – Ingeniería inversa y gestión de cambios en sistemas con reglas variables (tales como regulaciones gubernamentales, impuestos, etc...).

Tecnología de objetos

- Los lenguajes de programación de propósito general y los de sistemas de bases de datos son inadecuados para satisfacer las necesidades de las cada día más sofisticadas aplicaciones.
- Los lenguajes de programación tradicionales (tales como C y C++) carecen de *persistencia* y en general, de los servicios proporcionados por un DBMS.
- Los DBMSs tradicionales han sido desarrollados para el entorno de gestión: registros simples, la atención a un gran número de usuarios y las transacciones de corta duración.

  Necesitan las características de un lenguaje de programación para manejar estructuras de datos complejas. Además, requieren nuevas funcionalidades, como *triggers*, reglas, transacciones de larga duración, control de versiones, multimedia, etc...

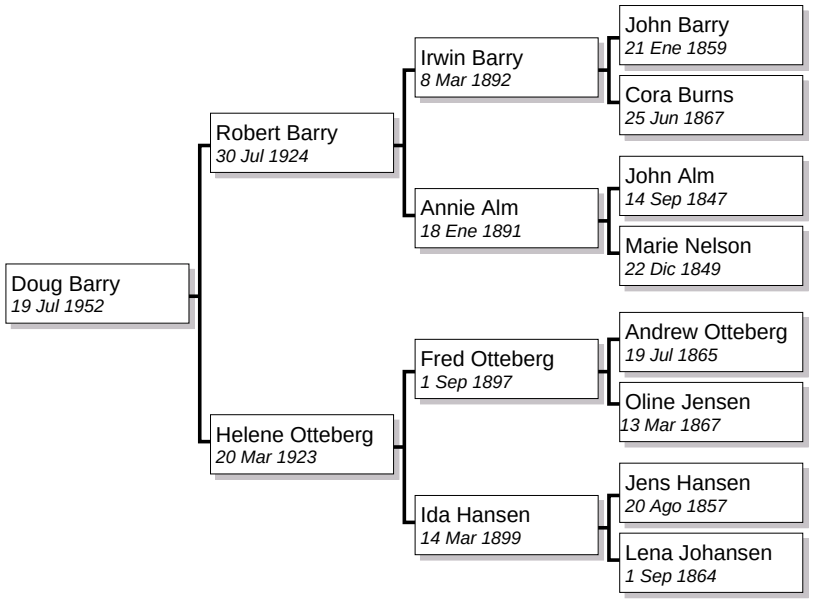
La tecnología orientada a objetos emerge como el paradigma unificador. Podemos diseñar:

- Objetos pasivos simples que sólo representan un valor.
- Objetos pasivos complejos que dan soporte a estructuras de datos complejas.
- Objetos multimedia que representan imágenes, vídeo o sonido.
- Servicios, que responden a peticiones de algún tipo.
- Objetos activos que llevan a cabo algún tipo de actividad de forma independiente.
- Objetos reactivos que responden a *eventos* del entorno.
- Objetos inteligentes que exhiben capacidad de razonamiento.
- y finalmente, objetos capaces de evolucionar modificando su comportamiento.

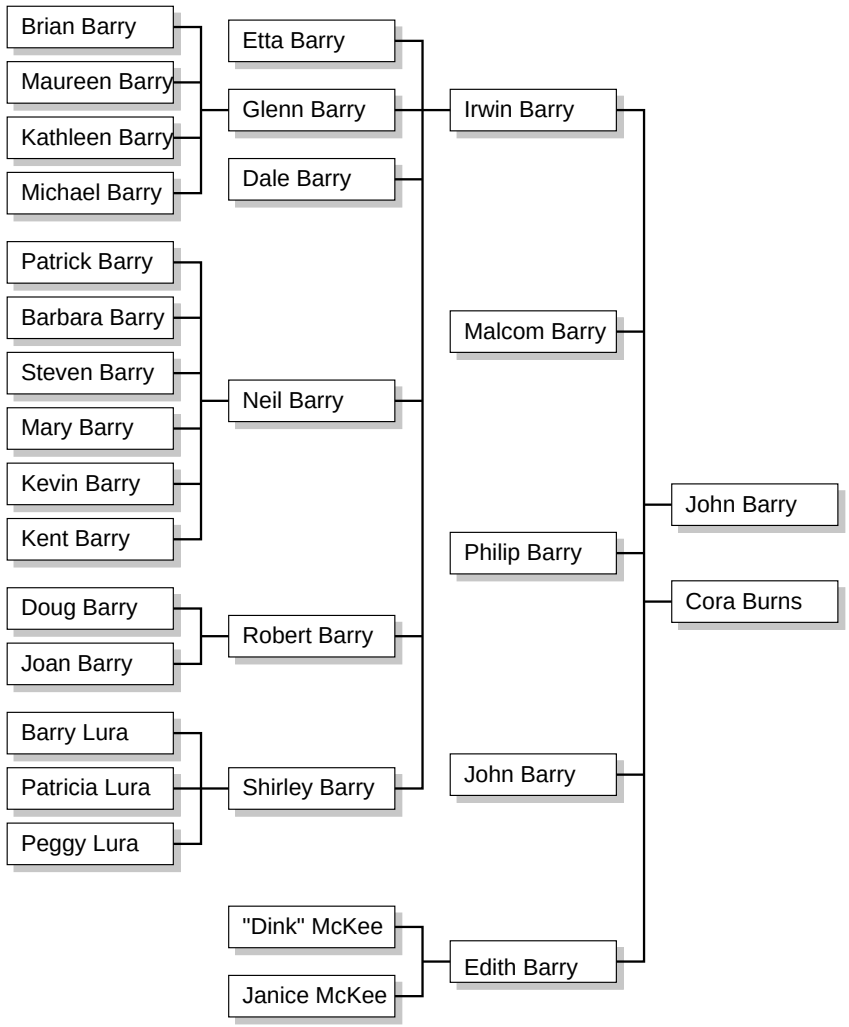
Información compleja

Un DBMS relacional puede manejar con éxito los datos complejos hasta un cierto límite, pero en algún momento la información puede alcanzar un grado de complejidad demasiado grande.

Los DBMS orientados a objetos pueden ser la clave ya que proporcionan excelentes rendimientos cuando es necesario almacenar y recuperar grandes cantidades de información compleja.



Información compleja (cont)...



Esta figura y la anterior representan el mismo árbol de familia desde dos puntos de vista. Intentemos ahora imaginar un esquema completo del árbol familiar...

Información compleja (cont)...

Otros ejemplos de datos complejos incluyen modelos de datos en sistemas multimedia, banca, seguros, inversiones bursátiles, telecomunicaciones, manufacturas, etc...Es fácil reconocer la presencia de datos complejos analizando algunos factores:

- Ausencia de un identificador único.
- Relaciones muchos-a-muchos.
- Acceso a los datos mediante recorridos.
- Uso frecuente de códigos de tipo de dato.

Ausencia de un identificador único

La mayoría de las cosas de este mundo carecen de un identificador unívoco propio.

- Un ejemplo de algo tangible que carece de un identificador único lo constituye una dirección postal. No existe ningún identificador natural aparte de la misma dirección postal completa.
- Algunas cosas menos tangibles, como sentimientos, sonidos o un movimiento, también carecen de identificador unívoco.

Los sistemas orientados a objetos generan OIDs (*Object IDentifiers*) para identificar a nivel de sistema aquellas cosas que tienen un significado.

- Para los datos que carecen de identificador inherente, el OID proporciona una forma única e inmutable de referenciar los datos.
- En aquellos casos en los que los datos si disponen de un identificador unívoco, el OID proporciona una forma alternativa e inmutable de hacer referencia al objeto aún cuando el identificador unívoco cambie. Un ejemplo de esto último lo constituye el caso de una fabrica que decide cambiar los números de *Part ID* de sus productos: la utilización de referencias basadas en OIDs permitiría modificar la estructura de *Part IDs* sin afectar a nada que haga referencia a dichos elementos.

Una característica común a todo tipo de información compleja es que se puede representar, dentro del esquema orientado a objetos, mediante un grafo. El mecanismo de OIDs proporciona un mecanismo excelente para representar la información descrita por este tipo de grafos.

Relaciones muchos-a-muchos

Otro signo de la presencia de información compleja lo constituye la existencia de relaciones muchos-a-muchos.

Por ejemplo, un estudiante puede matricularse en varias asignaturas y en una asignatura pueden estar matriculados varios estudiantes. Cada instancia en la relación estudiante-asignatura tiene varias opciones:

Cada estudiante puede matricularse en varias asignaturas

Cada asignatura cuenta con varios estudiantes

Estudiante

Nombre	Nacimiento	Asignatura
Juan	14 Ene 1978	MN−6505 A−6004 CDI−6092
Teresa	13 Jun 1977	BD−6434 DBD−6160
Susana	8 Ago 1978	CDI−6092 BD−6434 A−6004
Marc	14 Dic 1999	DBD−6160 BD−6434 MN−6505

Asignatura

Número	Departamento	Estudiante
A−6004	Algebra	Juan Susana
CDI−6092	Análisis Mat.	Juan Susana
MN−6505	Informática	Juan Marc
BD−6434	Informática	Teresa Susana Marc
DBD−6160	Informática	Teresa Marc

- Un DBMS orientado a objetos no tiene problemas para gestionar este tipo de relaciones: la información es almacenada de forma muy similar a la representada en la figura.
- El modelo relacional no permite la repetición de grupos. En la figura, repetición de grupos son las múltiples asignaturas que un estudiante puede cursar o los varios estudiantes que se matriculan en una misma asignatura.

Relaciones muchos-a-muchos (cont.)

Para obviar este problema, es necesario duplicar la información:

Se repiten nombre y fecha de nacimiento

Se repiten número de curso y departamento

Estudiante

Nombre	Nacimiento	Asignatura
Juan	14 Ene 1978	MN−6505
Juan	14 Ene 1978	A−6004
Juan	14 Ene 1978	CDI−6092
Teresa	13 Jun 1977	BD−6434
Teresa	13 Jun 1977	DBD−6160
Susana	8 Ago 1978	CDI−6092
Susana	8 Ago 1978	BD−6434
Susana	8 Ago 1978	A−6004
Marc	14 Dic 1999	DBD−6160
Marc	14 Dic 1999	BD−6434
Marc	14 Dic 1999	MN−6505

Asignatura

Número	Departamento	Estudiante
A−6004	Algebra	Juan
A−6004	Algebra	Susana
CDI−6092	Análisis Mat.	Juan
CDI−6092	Análisis Mat.	Susana
MN−6505	Informática	Juan
MN−6505	Informática	Marc
BD−6434	Informática	Teresa
BD−6434	Informática	Susana
BD−6434	Informática	Marc
DBD−6160	Informática	Teresa
DBD−6160	Informática	Marc

La información redundante es causa de multitud de problemas. Por ejemplo:

- Sólo modificar la fecha de nacimiento de `Juan` supone modificar tres registros en la misma tabla `Estudiantes`.
- Si `Juan` anula su matricula en la asignatura `CDI-6092` es necesario borrar dos registros en dos tablas diferentes, `Estudiante` y `Asignatura`

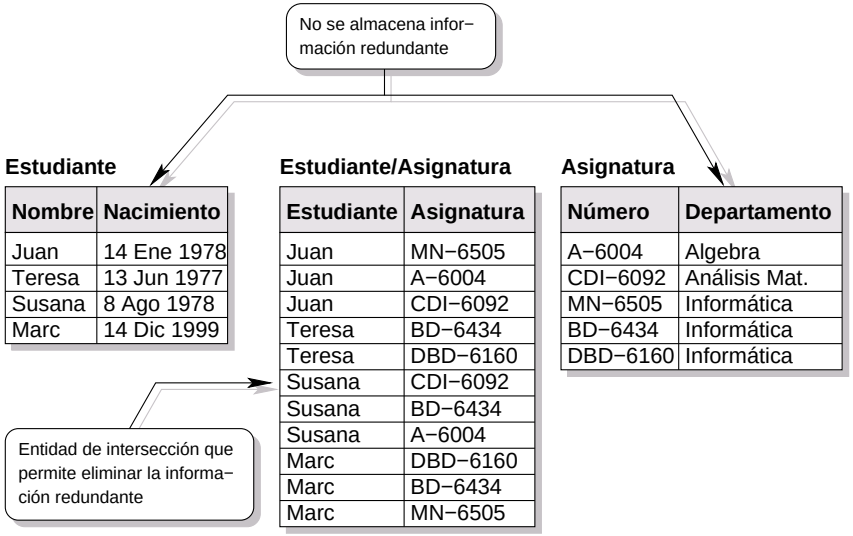
Actualizaciones anómalas. Un ejemplo:

- Borrar el registro `Juan` de la asignatura `CDI-6092` sólo en la tabla `Estudiante`.
- Si consultamos la tabla `Estudiante`, `Juan` no cursa `CDI-6092`, pero si consultamos la tabla `Asignatura` `Juan` está todavía matriculado.

Relaciones muchos-a-muchos (cont.)

Para resolver este problema:

- Los desarrolladores de esquemas relacionales `normalizan` o simplifican los datos.
- Esta técnica consiste en introducir una entidad (tabla) que es la intersección de las tablas `Estudiante` y `Asignatura`.
- Empleando este esquema no es necesario duplicar más información que la necesaria para hacer referencia a los datos almacenados en la otra tabla.
- La normalización de datos permite resolver el problema de las relaciones muchos-a-muchos en el modelo relacional.



Relaciones muchos-a-muchos (cont.)

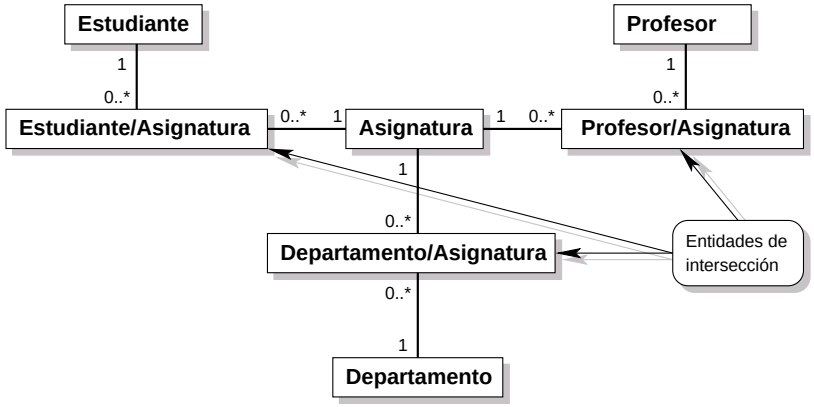
Problema adicional: un DBMS relacional debe combinar los identificadores. A medida que aumenta el número de tablas que es necesario combinar para obtener la información el rendimiento del DBMS relacional decrece:

- Es necesario acceder a múltiples tablas con información clave para combinar los datos de todas las tablas que contienen dicha clave.
- Cuanto mayor es el número de tablas necesarias, más operaciones de combinación es necesario realizar y más se relentiza el proceso.

Ampliemos un poco el problema anterior y analicemos como afecta al rendimiento:

- Una asignatura puede tener más de un profesor.
- Un estudiante puede cursar más de una asignatura.
- Una asignatura cuenta con varios estudiantes.
- Un departamento tiene adscritas varias asignaturas.
- Una asignatura puede estar adscrita a más de un departamento.

Relaciones muchos-a-muchos (cont.)



En este modelo hay tres entidades de intersección:

- `Estudiante/Asignatura`.
- `Profesor/Asignatura`.
- `Departamento/Asignatura`.

El esquema relacional completo cuenta con siete entidades: las tres entidades de intersección más las cuatro tablas `Profesor`, `Estudiante`, `Asignatura` y `Departamento`.

Obtener un listado con la información sobre qué asignaturas (incluyendo departamento y profesor) cursa cada estudiante exige combinar las siete entidades. En una situación real este proceso puede llegar a ser bastante lento en muchos DBMS relacionales cuando la base de datos alcanza una cierta magnitud.

Relaciones muchos-a-muchos (cont.)

El empleo de entidades de intersección para resolver las relaciones muchos-a-muchos en un esquema relacional es un indicador de la existencia de información compleja.

Si en una base de datos se requiere un número relativamente alto de entidades de intersección, es probable que un DBMS orientado a objetos sea más adecuado para su implementación por dos motivos fundamentales:

- El alto coste en rendimiento que puede tener la utilización de un gran número de entidades de intersección. Este coste tiene su origen en la lentitud del proceso de combinación. Una solución frecuentemente adoptada en los DBMSs relacionales consiste en evitar en la medida de lo posible las entidades de intersección (este proceso se conoce como *desnormalización*) y suele conducir a modelos de datos poco adecuados.
- El coste de mantenimiento de todas las entidades implicadas cuando se utiliza un DBMS relacional. Este coste de mantenimiento tiene su origen en el pobre modelo de datos al que se puede llegar cuando se recurre al mecanismo de desnormalización.

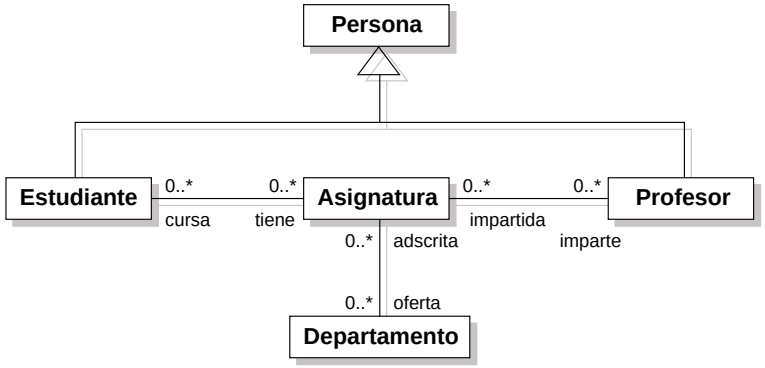
Relaciones muchos-a-muchos (cont.)

El modelo orientado a objetos gestiona las relaciones muchos-a-muchos de forma fácil. Como se muestra en la figura, empleando el modelo orientado a objetos sólo son necesarias cuatro entidades:

- `Profesor`.
- `Estudiante`.
- `Asignatura`.
- `Departamento`.

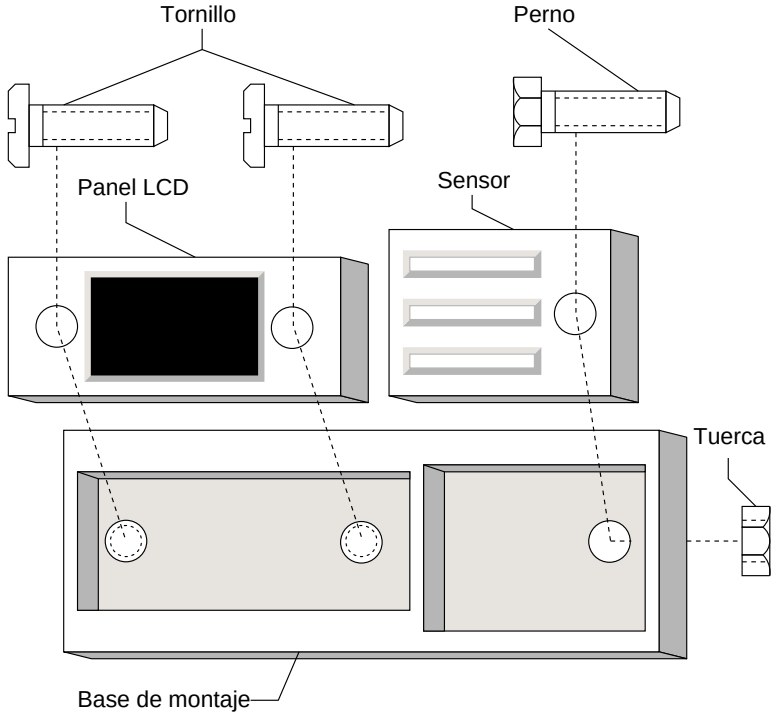
No es necesario combinar entidades:

- Basta con recorrer todas las relaciones.
- Un recorrido sigue los enlaces entre objetos empleando el OID de los objetos ligados como punteros.



Acceso mediante recorridos

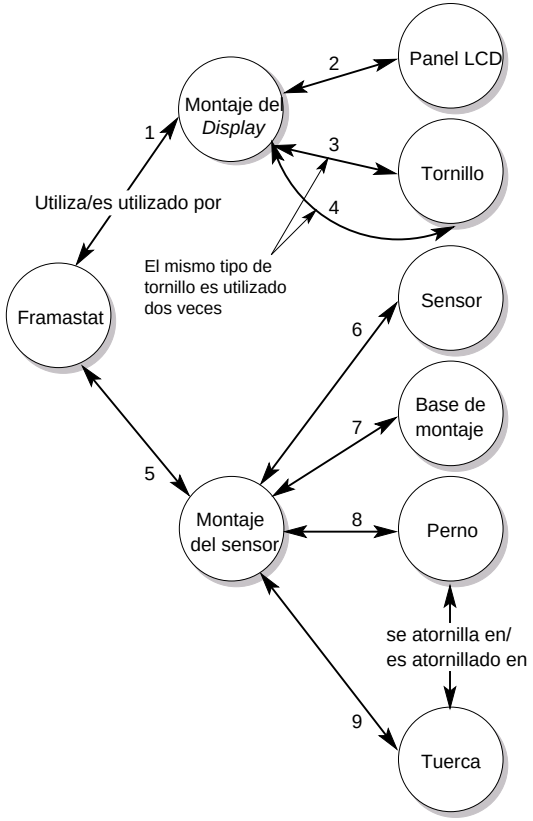
Un recorrido se da cuando hay que desplazarse desde una instancia de un objeto hacia otras instancias conectadas a ésta. Un despiece de los componentes de un determinado diseño es un ejemplo del empleo de conexiones entre instancias.



Un despiece de este tipo se representa en un lenguaje orientado a objetos mediante una estructura de tipo grafo.

Acceso mediante recorridos

Los lenguajes de programación orientados a objetos acceden a la información compleja recorriendo las relaciones mediante flechas.



Los grafos son, en general, indicadores de que la información que estamos tratando es compleja. En un DBMS relacional el recorrido de un grafo se realiza mediante sucesivas búsquedas de índices y combinaciones de tablas.

Un DBMS relacional es más lento que un DBMS orientado a objetos para este tipo de aplicaciones.

Uso frecuente de códigos de tipo

El empleo de códigos de tipo es:

1. Indicador de que la información debe procesarse de forma específica (mediante estructuras del tipo `if-then-else-if`)
2. Indicativo de la presencia de una jerarquía de tipos.
3. El procesado especial necesario para cada tipo requiere que el código del programa lleve a cabo algún tipo de comprobación de códigos.

- Los DBMSs relacionales comerciales no gestionan bien los tipos: no *entienden* que el tipo de código va asociado a una forma diferente de manipular el objeto.
- En contraposición, los tipos de código son una parte inherente de la jerarquía de clases de un DBMS orientado a objetos. En un ODBMS cada clase tiene asociado un conjunto de métodos haciendo innecesaria la utilización de estructuras del tipo `if-then-else-if`.

Los modelos relacional y orientado a objetos: comparación

¿Qué es mejor, un RDBMS o un ODBMS?

La respuesta es que no existe una respuesta. Qué modelo de datos es mejor depende de un gran número de factores.

- La tecnología relacional es bien conocida y cuenta con una tradición de varias décadas.
- La tecnología orientada a objetos es más reciente y tiene su origen en los departamentos de CAD (*Computer Aided Design*). Para describir sus modelos necesitaban mayor potencia de proceso que la que el modelo relacional podía proporcionarles.

  La solución:

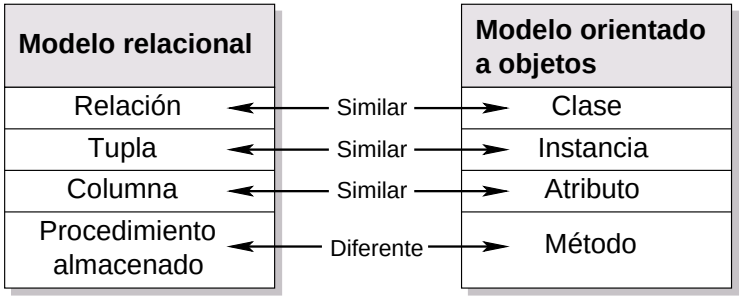
  Unificar la tecnología orientada a objetos con los sistemas de gestión de bases de datos.

  Curiosamente, esta misma demanda de rendimiento se está dando en la actualidad en industrias tan alejadas del CAD como las telecomunicaciones, seguros y mercados financieros.

Los puntos básicos que analizaremos son:

- Las similitudes y diferencias conceptuales entre un RDBMS y un ODBMS.
- La utilidad de los conceptos de abstracción y herencia en la manipulación de datos complejos.
- La utilidad de las técnicas de encapsulación en el desarrollo de sistemas grandes y complejos.

Terminología relacional y orientada a objetos



- El término *relación* empleado en el modelo relacional es similar al término *clase*. Ambos términos definen la estructura de un grupo de instancias similares.
- El concepto de *tupla* del modelo relacional es similar al concepto de *instancia de objeto* ya que ambos contienen los datos.
- El término *columna* en el modelo relacional es el equivalente a *atributo* en el modelo orientado a objetos en el sentido de que ambos contienen un dato simple.

La diferencia más significativa se encuentra en la forma en que se asocian los datos con el código ejecutable:

- El modelo relacional emplea *procedimientos* (almacenados)
- El modelo orientado a objetos utiliza *métodos*.

Similitudes y diferencias conceptuales

Tres son los conceptos básicos del modelo orientado a objetos:

- Abstracción.
- Herencia.
- Encapsulado.

Utilizaremos estos tres conceptos para comparar y contrastar los modelos relacional y orientado a objetos empleando un ejemplo simplificado en el que interviene información compleja.

Datos abstractos

El concepto de dato abstracto nos permite simplificar y modelar de forma adecuada los datos que encontramos en el mundo real.

En la siguiente figura se muestra una primera aproximación a la descripción del sensómetro digital avanzado mediante el modelo relacional.

Parte

ID Parte	Tipo	Peso
P1	Panel LCD	0.5
P2	Tornillo	2.0
P3	Sensor	1.2
P4	Base de montaje	14.0
P5	Perno	0.9
P6	Tuerca	6.0
P7	Montaje*Display*	
P8	Montaje sensor	
P9	Sensómetro	

Usa

ID Parte	Utiliza parte
P7	P1
P7	P2
P7	P2
P8	P3
P8	P4
P8	P5
P8	P6
P9	P7
P9	P8

Valores que es necesario calcular

Aunque de forma simplificada, el esquema ilustra algunos conceptos importantes:

- En el modelo relaciona, cada instancia debe tener un identificador que es proporcionado en los mismos datos. En el ejemplo concreto, este identificador es `ID Parte`.
- Las referencias entre elementos requieren el uso de identificadores `ID Parte`, que se implementan mediante la relación (tabla) `Usa`. Por ejemplo, en el Montaje del sensor (`P8`) utiliza una base de montaje (`P4`); ambas claves son extrañas al modelo y constituyen la forma en que el modelo relacional implementa las *flechas*.

Datos abstractos

Un modelo algo más realista que muestra los diferentes tipos de datos es el mostrado en la siguiente figura y refleja el hecho de que diferentes partes y montajes requieren almacenar información específica. Se ha añadido una columna `Tipo` en la relación `Parte` de modo que sea posible determinar el tipo pieza de cada parte.

Parte

ID Parte	Tipo
P1	Panel LCD
P2	Tornillo
P3	Sensor
P4	Base de montaje
P5	Perno
P6	Tuerca
P7	Montaje
P8	Montaje
P9	Montaje

LCD

ID Parte	Nombre	Peso	Voltaje
P1	Panel LCD	0.5	0.1
más...			

Tornillo

ID Parte	Nombre	Peso	Cabeza
P2	Tornillo	2	Philips
más...			

Sensor

ID Parte	Nombre	Peso	Umbral
P3	Sensor	1.2	95
más...			

Base de montaje

ID Parte	Nombre	Peso	Altura
P4	Base	0.9	30
más...			

Usa

ID Parte	Utiliza parte
P7	P1
P7	P2
P7	P2
P8	P3
P8	P4
P8	P5
P8	P6
P9	P7
P9	P8

Perno

ID Parte	Nombre	Peso	Diámetro	Atornilla en
P5	Perno	14	7	P6
más...				

Tuerca

ID Parte	Nombre	Peso	Diámetro	Atornilla en
P6	Tuerca	6	7	P5
más...				

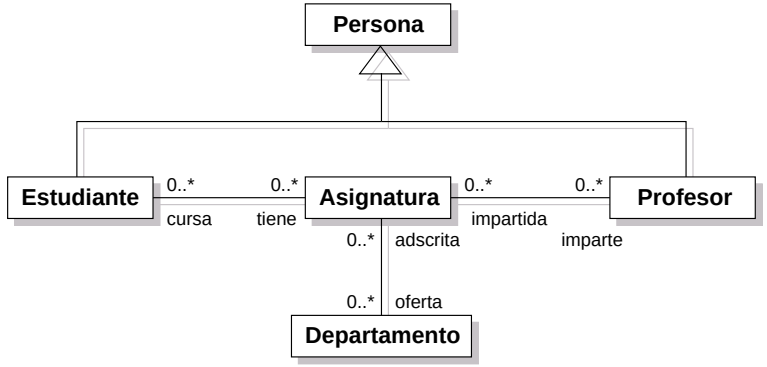
Montaje

ID Parte	Nombre
P7	Montaje del *Display*
P8	Marco de montaje
P9	Sensómetro Digital

En el esquema relacional, existe una relación independiente para cada tipo de parte ya que cada parte requiere almacenar una determinada información específica.

Herencia

La herencia es el mecanismo que nos permite definir una clase en términos de otra clase. La herencia en el modelo orientado a objetos toma el papel de los códigos de tipo en el modelo relacional.



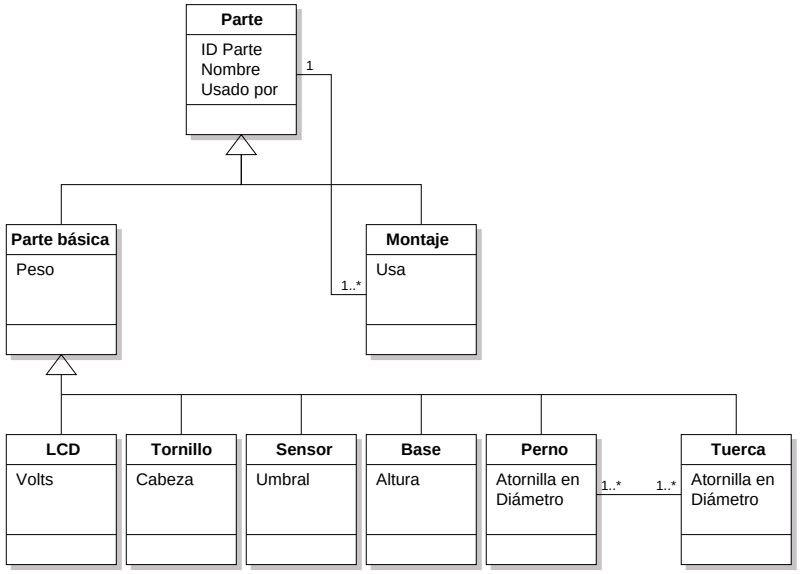
En el ejemplo de la univesidad, `Estudiante` y `Profesor` heredaban sus atributos y métodos de la clase `Persona`: la herencia elimina la necesidad de emplear los códigos de tipo. Puesto que:

- `Estudiante` es un *tipo* de `Persona`.
- `Profesor` es otro *tipo* de `Persona`.

desaparece la necesidad de utilizar códigos de tipo.

Herencia

Tampoco son necesarios los códigos de tipo en el ejemplo del sensómetro digital avanzado: los tipo son parte de al jerarquía de herencias.



El mecanismo de herencia en este ejemplo permite definir cada parte del sistema en términos de otras partes. Por ejemplo, una `Parte básica` y un `Montaje` son ambos tipos de `Parte` y bajando más aún en la jerarquía, un `LCD` es un tipo de `Parte básica`.

Uso de los tipos abstractos y la herencia

Objetivo:

- producir un listado de material del sensómetro digital avanzado.
- comprobar que influencia ejerce el uso de los datos abstractos y la herencia en la tarea.

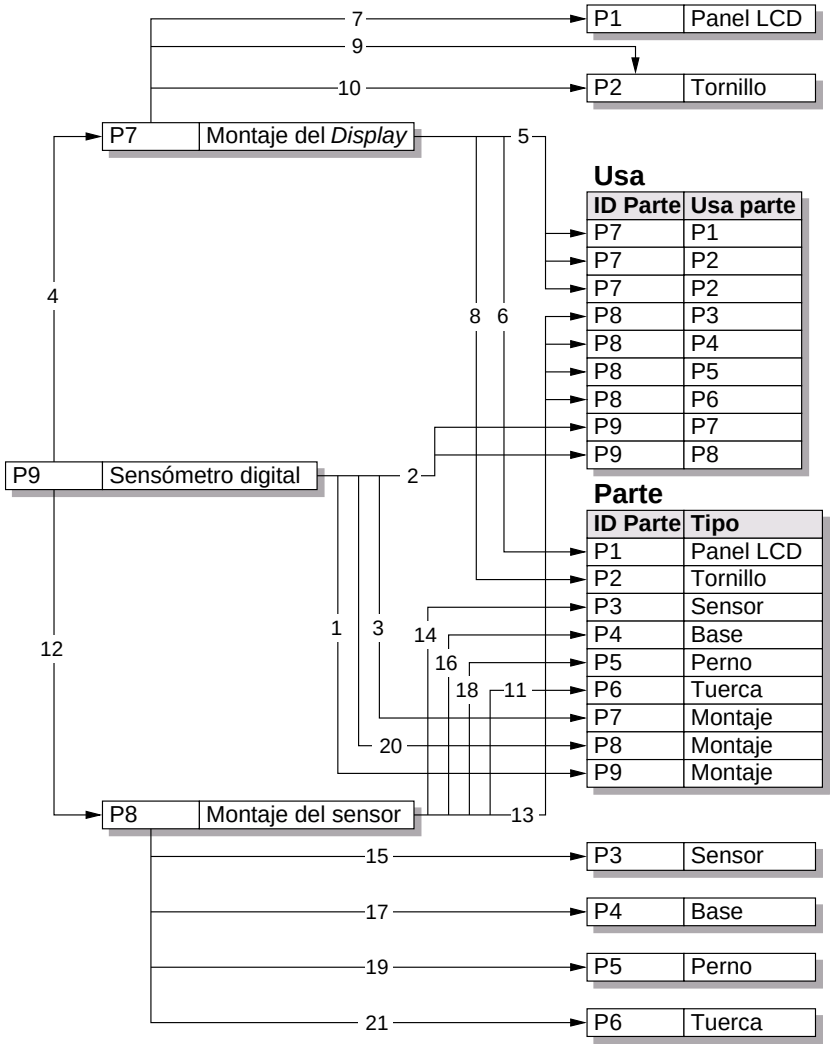
Instrumentos de medida, S.L.			Fecha: 24 Ene 2000
Sensómetro digital	P9	Peso: 98.3 g	
Montaje del *Display*	P7	Peso: 18.3 g	
Panel LCD	P1	Peso: 14.3 g	Volts: 0.1
Tornillo	P2	Peso: 2.0 g	Cabeza: Philips
Tornillo	P2	Peso: 2.0 g	Cabeza: Philips
Montaje del sensor	P8	Peso: 80.0 g	
Sensor	P3	Peso: 34.3 g	Umbral: 95
Base de montaje	P4	Peso: 25.7 g	Altura: 30 mm
Perno	P5	Peso: 14 g	Diámetro: 7 mm
Tuerca	P6	Peso: 6.0 g	Diámetro: 7 mm

Utilizando el modelo orientado a objetos basta recorrer la estructura de datos siguiendo la numeración de las flechas.

El recorrido tiene lugar utilizando los OIDs, pero desde la perspectiva del programa, este recorrido tiene lugar en la forma estándar en un lenguaje de programación orientado a objetos y la traducción de OIDs al lenguaje de programación utilizado la lleva a cabo el ODBMS.

Uso de los tipos abstractos y la herencia

En el esquema relacional, el recorrido sería:



Uso de los tipos abstractos y la herencia

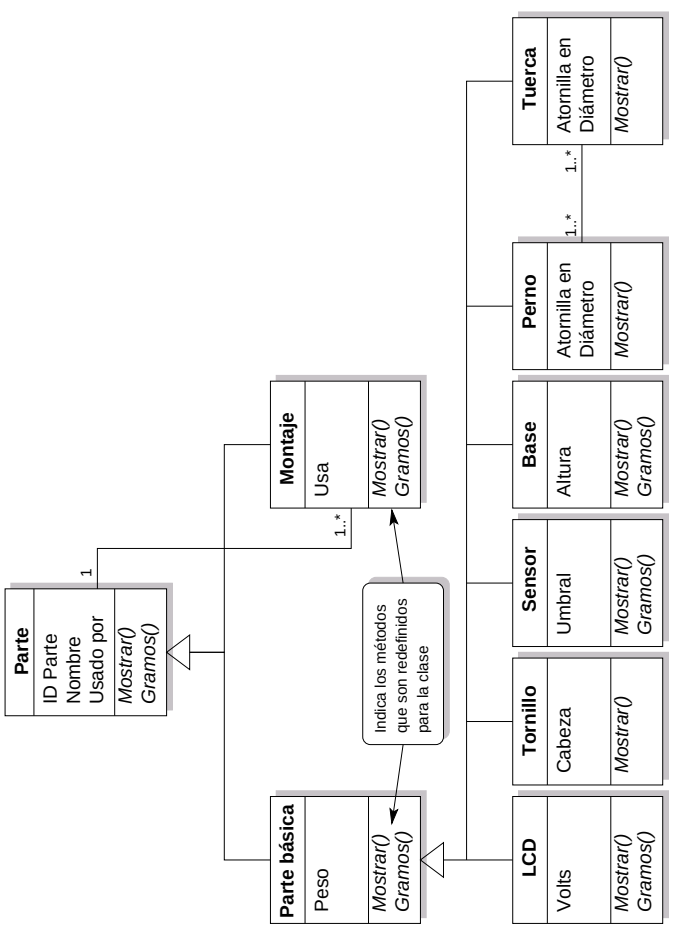
- Las relaciones `Usa` y `Parte` son consultadas continuamente.
- La relación `Usa` se emplea para determinar las relaciones que se describen de forma directa en el esquema orientado a objetos.
- La relación `Parte` se consulta para determinar el tipo de parte, que se requiere ya que es necesario ejecutar código diferente para cada tipo de parte.

Todo este trabajo adicional necesario para ensamblar la información almacenada en relaciones en una estructura de objetos (y al revés) se denomina en ocasiones *impedance mismatch*

- Un DBMS orientado a objetos no necesita la relación `Usa` ya que el camino recorrido se establece mediante las relaciones representadas por el OID.
- Tampoco necesita comprobar el tipo de parte para determinar el código de programa adecuado que tiene que ejecutar para la información asociada a un tipo particular de pieza.
- El código que se ejecuta es el adecuado porque un modelo orientado a objetos "sabe" de los diferentes tipos de piezas a través de la utilización de clases. Además, este código está *encapsulado* como parte de la definición de la clase.

Encapsulado

La técnica de encapsulado permite que el código se defina como parte de la definición de la clase. En el ejemplo que nos ocupa, el código específico para procesar cada tipo de parte puede encapsularse en la clase y no en el código de la aplicación.



Uso del encapsulado

Analicemos ahora el impacto que tiene la técnica de encapsulado en una operación como la de obtener una lista de piezas del modelo ejemplo que estamos tratando.

Código de la aplicación en el modelo orientado a objetos

```
Mostrar <ID del objeto sensómetro digital>
```

Código de la aplicación en el modelo relacional

```
if  type == Montaje then
      ... procesado específico para montaje ...
else if  type == LCD then
      ... procesado específico para LCD ...
else if  type == Tornillo then
      ... procesado específico para tornillo ...
else if  type == Sensor then
      ... procesado específico para sensor ...
else if  type == Perno then
      ... procesado específico para perno ...
... etc ...
```

- La aplicación desarrollada utilizando el modelo orientado a objetos no requiere información acerca de cómo debe mostrarse un objeto particular o cómo calcular su peso.
- El código de aplicación es mucho más corto que el utilizado en el esquema relacional, ya que este último requiere estructuras del tipo `if-then-else-if`.

Uso del encapsulado

Beneficio real muy importante de la técnica de encapsulado:

- La reducción del tamaño del código de la aplicación.
- Hay mucho más código encapsulado en las clases para implementar los métodos `Gramos()` y `Mostrar()`. Sin embargo, el programa de aplicación no necesita disponer de esa información.
- En términos del modelo orientado a objetos, la aplicación envía el mensaje `Mostrar()` al objeto "sensómetro digital avanzado".

La técnica de encapsulado permite desarrollar grandes sistemas de software (*programming in the large*):

- Este tipo de sistemas contienen un gran número de entidades de datos y muchas líneas de código.
- A medio y largo plazo, gestionar qué código actúa sobre qué datos da lugar a problemas graves que conducen a la duplicación de código y a ejecuciones incorrectas.
- El encapsulado permite asegurar que el código correcto se ejecuta sobre cada dato mediante la técnica de *overloading*.

Conclusiones

Concepto en el modelo orientado a objetos	Ténica en el modelo relacional	Técnica en el modelo orientado a objetos	Beneficio del modelo orientado a objetos
Datos abstractos	Entidades de intersección e índices que representan relaciones entre tuplas	OIDs que representan directamente referencias entre objetos	Esquema más simple para representar información compleja
Herencia	Códigos de tipo	Jerarquía de clases	Representación directa de las relaciones tipo/subtipo. Ejecución especializada
Código encapsulado	Estructuras if–then–else–if basadas en los códigos de tipo	Mecanismo intrínseco que garantiza que se ejecuta el código correcto	Reducción del código. Minimización de la posibilidad de error