

Bases de Datos

Apuntes

21/02/2008

Contenido

Parte 1: Introducción.....	4
Tema 1: Introducción	4
1.1. Definiciones	4
1.2. Sistemas de BBDD frente a Sistemas de Archivos.....	5
1.3. Visión de los datos.....	6
1.4. Modelos de datos	7
1.5. Lenguajes de Bases de Datos	9
1.6. Arquitectura de un SGBD	10
Parte 2: Ciclo de vida y modelos de datos	14
Tema 2: Modelos de datos	14
2.1. Definiciones	14
2.2. Aspectos de un modelo de datos	14
Tema 3: Ciclo de vida de las Bases de Datos	16
3.1. Diseño dirigido por procesos y diseño dirigido por datos.....	16
3.2. Ciclo de vida en cascada	16
3.3. Aplicación del ciclo de vida	16
Parte 3: Diseño conceptual: Modelo Entidad – Relación.....	21
Tema 4: Modelo Entidad - Relación	21
4.1. Introducción y elementos fundamentales	21
4.2. Restricciones de integridad	22
4.3. Reducción de diagramas E-R en tablas	25
4.4. Construcciones avanzadas.....	26
Tema 5: Modelo Entidad- Relación extendido	30
5.1. Relaciones ternarias	30
5.2. Características extendidas	31
5.3. Restricciones que afectan a las relaciones	32
Parte 4: Diseño lógico: Modelo relacional.....	34
Tema 6: Modelo relacional.....	34
6.1. Introducción	34
6.2. Estructura de las BBDD relacionales	34
6.3. Lenguajes de consulta formales: álgebra relacional	36
6.4. Cálculo relacional	39
6.5. Las 12 reglas de codd.....	40


Ejercicios (E-R y paso a tablas)	45
1. Ejercicio 1	45

Parte 1: Introducción

Tema 1: Introducción

1.1. Definiciones

- **Banco de Datos:** Conjunto de datos relacionados entre sí. Ejemplo: ficheros con la información académica.
- **Base de Datos:** Cuando el banco de datos en vez de almacenarse en soportes tradicionales se guardan en medios informáticos.

A veces lo dibujaremos así: . Es necesaria una forma de acceder a los datos y gestionarlos. En el primer caso (banco de datos) podría ser una persona, pero usando programas; en el caso del sistema informático (B.D.) se necesitan programas: SGBD.

- **SGBD:** Dada una o varias BBDD es el conjunto de programas que permite tener acceso a los datos almacenados.

El acceso a los datos implica consulta y modificación de los datos. “Acceder” también se conoce con el nombre de “gestionar”. También debe proporcionar:

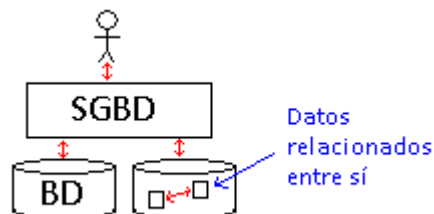
- Comodidad (conveniencia): que sea cómodo de acceder a los datos, fácil de utilizar.

Ejemplo:

```
SELECT nota
FROM alumno
WHERE nombre = 'Borja Flecha'
AND asignatura = 'BBDD'
```

- Eficiencia: Que de los resultados en un tiempo razonable. Hay que buscar equilibrio entre ellas: se intenta ir aumentando la comodidad y mejorando, si es posible, la eficiencia; o que al menos no se resienta.

Hay que gestionar grandes cantidades de información por lo que las estructuras de almacenamiento tienen que ser específicas y los algoritmos asociados hay que diseñarlos adecuadamente.



- Seguridad/control de acceso: No todo el mundo puede hacer cualquier cosa, hay que restringir el acceso. También hay que tener en cuenta otras cosas derivadas de la utilización.
- Concurrencia: ¿qué pasa si hay 10 o 100 o X usuarios que quieren modificar algo en la BD? Hay que considerar los casos en que haya más de un usuario accediendo simultáneamente.

Los SGBD más conocidos:

- Oracle: tiene una cuota de mercado muy grande.
- IBM DB2: en sistemas profesionales sobre todo Unix.

- Informix: comprado por IBM hace unos años.
- Sybase
- Ingress: de código abierto.
- MS SQL server
- MS Access
- MySQL: el más conocido y también el peor.
- Postgres

1.2. Sistemas de BBDD frente a Sistemas de Archivos

Para ver los objetivos de las BBDD vamos a ver primero los problemas de los sistemas de procesamiento de fichero.

Los datos se almacenan en un fichero formado por diversos registros con la información para acceder al fichero. Para acceder al fichero se realiza un programa específico para la función deseada. Si se necesita acceder a otra cosa debe hacerse otro programa o modificar uno ya existente. Y si necesitan incluirse nuevos datos se utiliza otro fichero.

Todo esto evoluciona a lo largo del tiempo aumentando en complejidad y tamaño. Y produce:

- Redundancia en los datos: Al haber varios ficheros puede haber datos repetidos (por ejemplo: nombre de cliente. La **redundancia** es la inconsistencia debido a que al estar los datos repetidos es necesario cambiar dichos datos en todos los ficheros; y para ello, deben modificarse todos los programas que modifican datos.
- Dificultad de acceso: no es cómodo acceder a los datos. Generalmente implica crear un programa específico para dicho acceso.
- Aislamiento de datos: Con el paso del tiempo se cambian los SSOO, los lenguajes de programación, etc. Y todos esos cambios hacen que coexistan ficheros con distintos formatos. Si pasado el tiempo se necesita acceder a un fichero antiguo es mucho más difícil además de no encontrar el necesario.
- Concurrencia: Como los SSOO no tienen demasiada protección de datos habría que rehacer os programas para que no hubiera problemas. También podría ponerse un “monitor de transacciones” entre los programas y los ficheros de transacciones.
- Seguridad: Con un SO puedes controlar que usuario lea o no un fichero completo, pero no puedes hacer cosas más complejas como acceder a unos campos sí y a otros no. Así que hay que programar con muchas condiciones; aunque eso no proporciona la suficiente seguridad ya que el fichero, al ser accesible por el usuario, puede ser consultado totalmente.
- Integridad: Además de almacenar los datos los valores tienen que respetar unas restricciones e integridad (por ejemplo, no vale que la edad sea -5). De nuevo hay que programarlo, aunque se realiza mediante bibliotecas, si hay cambios, además de la complejidad hay que cambiarlo todo.

La idea de las BBDD es meter inteligencia a las propias BBDD, no sólo datos. Con los datos meteremos información adicional a la que llamaremos **metadatos**. Antes la inteligencia estaba en los programas ahora se la meteremos al sistema.

La única forma de acceder a la BD es mediante los SGBD. Por ello, como hay un punto centralizado se puede controlar la concurrencia, la integridad, la seguridad...

Lo ideal es que el programa tenga la lógica del programa y todo lo demás para las BBDD. Pero no es así ya que algunas cosas habrá que hacerlo en las aplicaciones y se hacen cambios habrá que tocar código de las aplicaciones y recompilar pero ahora es más sencillo que antes.

1.3. Visión de los datos

1.3.1. Niveles de abstracción

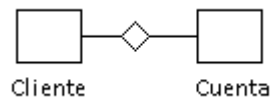
En el mundo de las BBDD se establecen 3 niveles de abstracción:

- Físico: indica cómo se van a guardar los datos. No llega al disco duro, sino que se refiere a árboles, tablas hash...
- Conceptual: Qué datos hay que cómo están relacionados entre sí. Se indica con estructuras de alto nivel.
- De visión o vistas: en lugar de enseñar todo el nivel conceptual, enseñamos subconjuntos de este adaptadas a cada usuario según las restricciones.

¿Qué estructuras vamos a tener en cada nivel? Las que se definen en cada nivel se llaman **esquemas**; en una BD tendremos un esquema conceptual, un esquema físico y unos esquemas de vistas o subesquemas.

1.3.2. Instancias y esquemas

- Esquemas: Representación de los datos que necesitamos para cada capa. Ejemplo: nivel conceptual.



- Instancia: Datos que hay almacenados en un momento dado. Si se añade, elimina o modificar algo de la BD la instancia cambia. Manejaremos las instancias cuando hablemos de **consultas**. Nosotros trataremos más con el esquema.

1.3.3. Independencia de datos

En dos niveles:

- Física: Es independiente. Podemos cambiar la representación en el esquema físico sin tocar nada en el nivel conceptual.
- Lógica: igual que la física podemos cambiar el esquema conceptual sin tocar las aplicaciones.

Con el nivel conceptual es más difícil puesto que las aplicaciones dependen o se basan en el esquema conceptual. Para ello utilizamos el nivel de visión.

En definitiva, podemos hacer cambios en un nivel sin afectar a otros y por ello a las aplicaciones.

1.4. Modelos de datos

Hay que decidir, en el nivel conceptual por ejemplo, qué tipo de estructuras se va a utilizar para mostrar a los usuarios.

Cada nivel está relacionado y el sistema debe conocer la correspondencia entre niveles. De igual modo que el sistema operativo abstrae al disco duro del usuario mediante ficheros, de forma que no tiene que preocuparse de escribir bloques ni nada similar; en los SGBD se abstrae de la BD mediante tablas.

Modelo de datos es la herramienta que se utiliza para describir lo que hay en cada nivel además de describir información adicional. Los hay de dos tipos: lógicos y físicos. Estos últimos en la práctica no están desarrollados.

Dependiendo de la estructura usada en cada nivel se usa uno u otro modelo. Generalmente el modelo conceptual (lógico) es muy importante ya que es el nivel que usa el usuario. En función de nuestra aplicación hay que elegir la organización física que mejor nos venga.

1.4.1. Modelos lógicos

Los hay de dos tipos:

- Basados en objetos
 - o Entidad – relación (el que más se usa)
 - o Orientado a objetos
- Basados en registros
 - o Relacional (usado por el 99% de las BBDD)
 - o Red (en desuso)
 - o Jerárquico (en desuso)

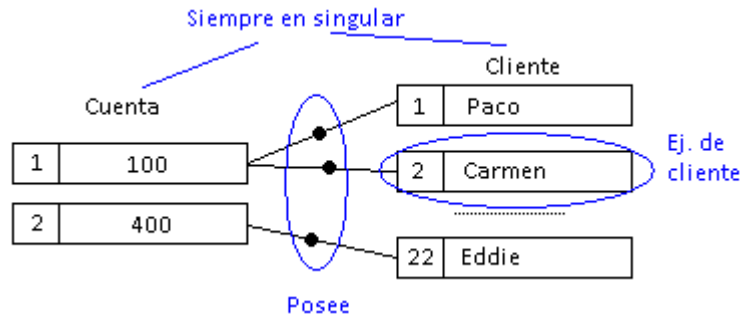
Primero se trabaja con un modelo basado en objetos y luego se pasa a relacional, que es lo comercial. Nosotros realizaremos los mismos pasos con el objetivo de que funcione en Oracle.

Veremos un ejemplo con cada modelo aunque hay que tener en cuenta que los modelos de datos no son totalmente expresivos y puede haber cosas que no se puedan expresar, no como en los lenguajes de programación.

E-R

Es un modelo **estricto** de datos. Se basa en que las estructuras de las que dispone para representar los datos son entidades y relaciones. Entidad es un objeto que existe por sí mismo, y las relaciones asocian entidades.

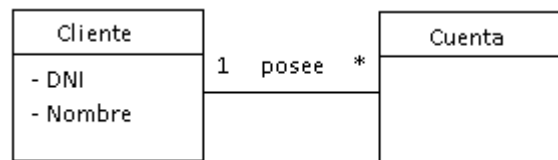
Ejemplo:



Orientado a objetos

En este modelo también se explica el comportamiento. Cada elemento del conjunto cliente se denomina ejemplar u ocurrencia. Mediante las categorías podemos asignar responsabilidades y capacidades a sus elementos. Además pueden establecerse relaciones entre dos categorías. Las relaciones se pueden establecer entre elementos individuales de las categorías y las relaciones pueden agruparse.

Ejemplo:



La principal diferencia entre OO y E-R es que en E-R sólo se definen los datos mientras que en O también se define el comportamiento.

Relacional

Organizamos los datos en forma de "tablas", con columnas a las que llamamos atributos.

DNI y Núm. Cuenta están subrayados ya que los usaremos como clave.

Cliente	<u>DNI</u>	Nombre	Cuenta	<u>Núm. Cuenta</u>	Saldo
	1	Paco		1	100
	2	Carmen		2	200
	222	Eddie			

Lo que nos falta ahora es la relación, es decir, quién es el dueño de cada cuenta. En el modelo relacional no hay relaciones ni asociaciones por lo que hay que relacionarlos mediante tablas.

Posee	<u>DNI</u>	<u>Núm. Cuenta</u>
	1	1
	2	1
	222	2

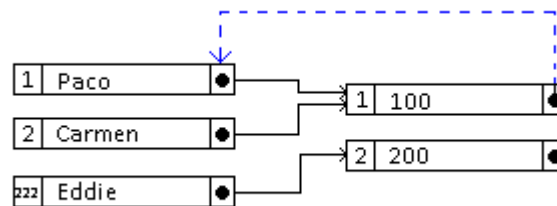
En este caso utilizamos claves, id que no se pueden repetir: DNI y núm. Cuenta. Para expresar que un cliente es propietario de más de una cuenta sería necesario introducir otra fila. Si cada cliente sólo puede tener una cuenta podríamos ahorrarnos esta tabla y poner donde DNI y Nombre como otra columna la cuenta asociada.

Este modelo no es perfecto, funciona muy bien con números y letras y con pocos tipos.

En red

Tenemos registros y enlaces, punteros entre los registros. La diferencia principal con E-R es que en E-R son punteros abstractos, mientras que en el modelo de datos en red son a bajo nivel (se ven, se puede quitar la referencia...).

Ejemplo:

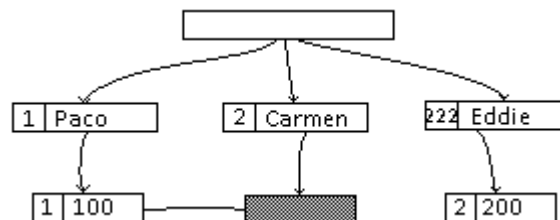


Para poder ir de uno a otro hay que tener un puntero, ya que no se puede utilizar el mismo que nos llevó al último registro. Es una forma de programar a muy bajo nivel. Se utilizaba en los años 70 antes de la existencia del modelo relacional.

Jerárquico

Es el predecesor del modelo en red, surgido en torno a los años 50.

Ejemplo:



Si se ponen 2 punteros a punto deja de ser un árbol por lo que se utilizaron una especie de registros fantasma, que lo que hacen realmente es redireccionarlo. Este modelo de datos es malo para cosas que se compartieran; aquí se prima un sentido de navegación (en este ejemplo, de clientes a cuentas).

1.5. Lenguajes de Bases de Datos

Necesitamos lenguajes de bases de datos para comunicarnos con el SGBD.

1.5.1. Lenguaje de definición de datos (DDL)

Nos permite definir el esquema a nivel conceptual y, en ocasiones, físico. Genera los metadatos del sistema.

Ejemplo:

```
CREATE TABLE Cliente
(
  DNI INTEGER,
  nombre CHAR(20)
)
```

Es importante tener en cuenta el diccionario de datos el cual tiene metadatos. Se modifica al compilar el código como el ejemplo. Añade la inteligencia de la BD que nosotros introduzcamos.

1.5.2. Lenguaje de Manejo de datos (DML)

De la misma forma que podemos definir las estructuras, tenemos que poder trabajar a nivel de instancia. El manejo de las instancias nos va a permitir dos cosas:

- Consulta
- Modificar / insertar / eliminar / actualizar.

Es importante que sea en un lenguaje cómodo, aunque hay muchos tipos de LDM. Básicamente hay dos tipos:

- Declarativos: se indica la forma que tiene el resultado, declaramos la definición, no se indica cómo obtenerlo.

Ejemplo:

$$\{ \langle DNI, nombre \rangle \mid \exists (ciudad \in cliente, DNI, nombre \rangle \in cliente \wedge \exists saldo (\langle -saldo, DNI \rangle \in cuenta) \}$$

Notación de lógica de 1^{er} orden

- Procedimentales: Hay que indicar los pasos que hay que dar, cómo manipular la información para llegar al resultado que queremos.

Ejemplo:

$$\Pi_{DNI, nombre}(\sigma_{ciudad = 'OVD' \wedge saldo > 100}(cliente \bowtie cuenta))$$

También podemos dividir los lenguajes por el “aspecto externo”:

- Comerciales: Pensados para el usuario final, con una sintaxis más atractiva y fácil. Ejemplo: SQL (sabiendo éste podemos manejar todas las BBDD ya que todas traen una interfaz SQL.

```
SELECT DNI, nombre
FROM cliente, cuenta
WHERE cliente.DNI = cuenta.DNI
      AND ciudad = 'OVD'
      AND saldo > 100
```

- Formales: Los lenguajes puros, como los que vimos antes.

1.6. Arquitectura de un SGBD

En este punto veremos diferentes elementos que intervienen en un SGBD, tanto humanos como de otro tipos.

GESTOR BBDD

Parte del sistema que se encarga de hacer de interfaz entre consultas de alto nivel y datos de bajo nivel. Centraliza los accesos. Tiene una serie de funciones:

- Interactuar con el HW (o con SO o sistema de ficheros...) → GESTOR DE ALMACENAMIENTO.
- Gestionar o implantar la integridad de los datos → GESTOR DE INTEGRIDAD.
- Controlar la seguridad → GESTOR DE AUTORIZACIONES.
- Ordenar el acceso concurrente a los datos → **GESTOR DE TRANSACCIONES**.
- Utilidades de respaldo y recuperación... Suelen tener una forma de volcar todo esto ya que tienen a un soporte de seguridad y luego poder recuperarlo.

→ GESTOR DE TRANSACCIONES

Una **transacción** es una unidad de trabajo con el sistema. Son un conjunto de operaciones que ejecutan de forma atómica. Son lógicas y forman una unidad. Deben tener:

- **Atomidad**: O se efectúan todas las operaciones de transacciones o ninguna.
- **Consistencia**: No se pueden dejar las BBDD en estado inconsistente.
- **Durabilidad**: Una vez finaliza una transacción, sus efectos son permanentemente persistentes y accesibles.
- **Aislamiento**: Cuando una transacción se está efectuando es como si estuviera ella sola.

Una transacción se puede abortar no sólo por razones de consistencia sino también de concurrencia además también hay que dar soporte para la atomicidad.

→ GESTOR DE AUTORIZACIONES E INTEGRIDAD

Controla los permisos de cada usuario. Por él pasan **todos** los accesos a datos. Para su funcionamiento también necesita el gestor de almacenamiento, el gestor de archivos y el gestor de memoria intermedia.

Hay 2 tipos de permisos de forma general:

- **Administrador de la BD**: Es la persona que tiene el control central sobre el sistema, tanto sobre los datos como sobre los programas que acceden a esos datos. Sus funciones son las siguientes:
 - Definiciones de los esquemas, tanto a nivel conceptual como a nivel físico (que, gracias a la independencia de datos, no afectan a las aplicaciones).
 - Modificaciones oportunas al esquema.
 - Concesiones de permisos, autorizaciones. En este caso el trabajo con los permisos no es como los ficheros de tipo "rwx" sino del tiempo: consultar, actualizar, etc. En SQL la forma de realizarlo está estandarizada.
 - Definición de restricciones de integridad. También estandarizado en SQL.
 - Monitorización del sistema: en general, vigilar cómo va el sistema. Para monitorizarlo necesita herramientas que le permitan ver determinados

parámetros, para ello o las realiza el administrador o las proporciona los SGBD.

- **Usuarios:** Hay 3 tipos:

- **Programadores/Desarrolladores de aplicaciones:** Son los que desarrollan las aplicaciones, normalmente usando lenguajes de manejo de datos. Para ello, incrustan sentencias SQL en el código de un lenguaje de los habituales (Java, C++...).

Ejemplo:

```
---  
int _saldo, _ncuenta;  
---  
  
$UPDATE cuenta  
SET saldo = saldo + _saldo  
WHERE ncuenta =: _ncuenta
```

Tal y como está, en un compilador normal no compilaría, normalmente se usa un preprocesador que transforma el código en SQL en algo que el compilador comprenda, para ello se utiliza la directiva "\$" antes del código para indicar al procesador que trabajamos con SQL. Este tipo de lenguajes se denominan **lenguajes de programación inmersa**. El uso de estos lenguajes es una manera de trabajar con ellos en el desarrollo ya que son lenguajes de manejo de datos pero no son de propósito general.

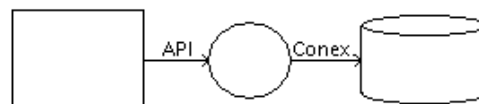
Otra técnica para su uso, que es la que utilizaremos nosotros, es con bibliotecas de acceso, es decir, bibliotecas del lenguaje para acceder a la Base de Datos. Es a lo que recurren los preprocesadores.

Ejemplo de bibliotecas: Ambos son independientes del sistema de gestión.

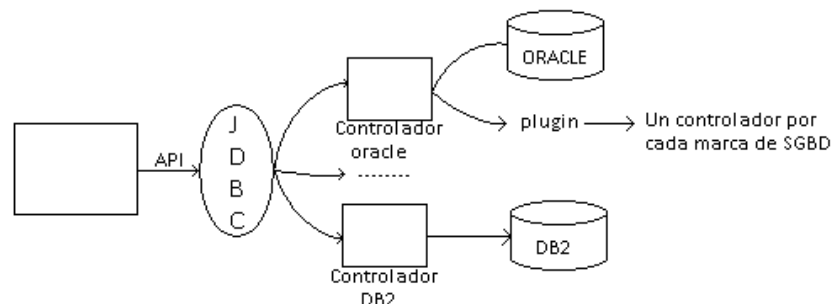
Java → JDBC

Windows → ODBC

Para explicar cómo funcionan antes de la introducción de bibliotecas:



Partiendo de esto, lo que hacen JDBC y ODBC es introducir un nivel de dirección en medio:



Al cargar le decimos al controlador que cargar con qué BBDD. Es importante hacer las cosas lo más independiente posible de las BBDD, porque si no dependeríamos de una marca concreta. Para ello están los std

en las API, la cual contiene toda la información que tiene que ser común a todos los sistemas.

La tercera de las técnicas es la utilización de un lenguaje de 4ª generación (L4G) aunque ahora está en desuso. Lo que hace es coger un lenguaje de marcado de datos y lo extiende para que sea un lenguaje completo, es decir, en un lenguaje de programación que pueda hacer de forma nativa el acceso a las BBDD para que fuese todo más rápido.

- Usuarios avanzados: Los que saben manejar las consultas.
- Usuarios normales: Los que manejan los programaciones de aplicaciones (que están usando una Base de Datos pero no ven cómo).

Parte 2: Ciclo de vida y modelos de datos

Tema 2: Modelos de datos

2.1. Definiciones

2.1.1. Evolución y definición

Al principio no existían los modelos de datos, por ello, la información se agrupaba en ficheros dependientes de la aplicación. Posteriormente, intentaron hacerse más genéricos; y con la aparición de las BBDD ya se comenzó a trabajar con modelos, es decir, representaciones de la realidad.

La clave está en la abstracción: modelador consiste en abstraer la realidad y obtener una representación. El **modelo** es la herramienta que nos permite obtener la representación; el **esquema** es el resultado de aplicar el modelo a un caso concreto, es decir, la representación.

2.1.2. Universo del discurso/mundo real

Habitualmente al problema se le llama “**universo del discurso**”. Es, del mundo real, lo que el diseñador considera relevante para un problema en un caso concreto. Como dijimos antes, el modelo es una herramienta de representación. El esquema son los elementos relevantes obtenidos del “universo del discurso” mediante el modelo. Dependiendo del “universo del discurso” habrá unos u otros elementos relevantes; por ello, puede haber varios esquemas con el mismo “universo del discurso a partir de varios modelos.

2.1.3. Objetivos de un modelo de datos

Básicamente hay dos objetivos:

- Formalizar (razonar).
- Diseñar (construir).

2.2. Aspectos de un modelo de datos

El modelo de datos tiene una serie de propiedades:

- Estáticas: elementos que siempre pertenecen, como las estructuras de datos. Representados por el lenguaje de definición de datos.
- Dinámicas: representan la evolución de los datos, es decir conjunto de instrucciones para manejar las estructuras.

2.2.1. Estática

Dentro de las propiedades estáticas nos encontramos con:

- Objetos permitidos: los elementos representados por el modelo (dependen de él).
- Objetos prohibidos: Hay elementos que no se pueden representar.
Ejemplo: En el modelo relacional los elementos deben tener un único valor. No sería válido por ejemplo:

	afición
	{fútbol,cine,teatro}

2.2.2. Dinámica:

- Selección
 - Conjunto
 - Ocurrencia individual
- Acción
 - Consulta
 - Modificar
 - Altas = Inserciones
 - Bajas = Borrados
 - Actualizaciones

Necesitamos lenguajes para niveles:

- Lógico / Conceptual:
 - Conceptuales: mayor nivel de abstracción (más potente)
 - Convencionales: implantación (SGBD)
- Físico

2.2.3. Restricciones de integridad

Hay dos tipos de restricciones:

- Restricciones inherentes al modelo.
- Restricciones de usuario: hay casos que no pueden ocurrir en el mismo “universo del discurso”.
 - Reconocidas: el sistema permite definir esa restricción.
 - No reconocidas: el sistema no puede definir algunas cosas, por lo que hay que programar las restricciones a mano.

Tema 3: Ciclo de vida de las Bases de Datos

3.1. Diseño dirigido por procesos y diseño dirigido por datos

A la hora de representar los datos del sistema hay dos diseños:

- Dirigido por procesos: El esquema de la BD se deriva directamente de las necesidades que tengan los procesos.
- Dirigido por datos: Los datos se organizan a partir de los propios datos que existen y se necesitan, independientemente de cómo debe trabajarse con ellos. Se deriva la estructura de la Base de Datos de la semántica de los datos.

Los problemas de uno son las ventajas del otro: si la estructura de datos está organizada por los procesos que tenemos es muy eficiente, si bien, si cambian los procesos, la estructura ya no es válida.

Generalmente se utiliza el diseño dirigido por datos (primero miras qué hay y luego qué hay que hacer), ya que los procesos es “normal” que cambien. El diseño dirigido por procesos se utiliza en lugares en los que la eficiencia es esencial, como en los sistemas en tiempo real o en aquellos en que los procesos no suelen cambiar.

3.2. Ciclo de vida en cascada

Se compone de las siguientes fases:

1. Análisis de requisitos.
2. Diseño (estructura) conceptual.
3. Diseño lógico.
4. Refinamiento por el uso: se adapta la estructura de la Base de Datos de forma limitada para mejorar el rendimiento.
5. Diseño físico: conocida la estructura de la Base de Datos hay que diseñar la estructura física que tendrá (árboles, índices...)
6. Implementación: Usando el lenguaje de definición de datos se definen los datos; luego se definen los procesos.
7. Prueba: Nada garantiza que la implementación está bien hecha, así que hay que realizar pruebas para comprobar que se cumplen los requisitos (por ahora no hay forma de que esto se haga de forma automática). Se realizan dos tareas:
 - a. Monitorización: Comprobar qué está pasando para tomar las decisiones adecuadas.
 - b. Mantenimiento: Una vez terminado e instalado, puede necesitarse cambiar procesos o mismamente datos.

3.3. Aplicación del ciclo de vida

Realizaremos un ejemplo con todas las fases para entenderlas mejor. El **universo del discurso** es una empresa con clientes, vendedores y pedidos. Los clientes hacen pedidos a los vendedores.

3.3.1. Análisis de requisitos

Mediante entrevistas con el cliente tenemos que averiguar los requisitos. Es una labor muy difícil, a partir de la cual se obtendrá el “documento de requisitos”.

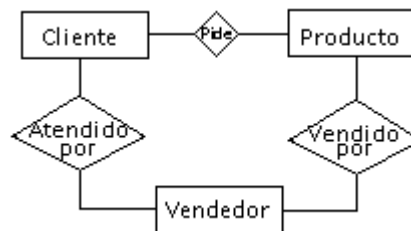
Habrà un diccionario de datos en el que lo nombres de las cosas estaràn descritos para evitar ambigüedad de la siguiente forma:



3.3.2. Diseño conceptual

Debemos dar una especificación en el modelo conceptual del universo del discurso. Generalmente no se puede abordar todo el sistema de un golpe por ser demasiado grande, sino que se divide en subsistemas para tratarlo más fácil; de esta forma se obtienen diferentes vistas que deberán ser integradas:

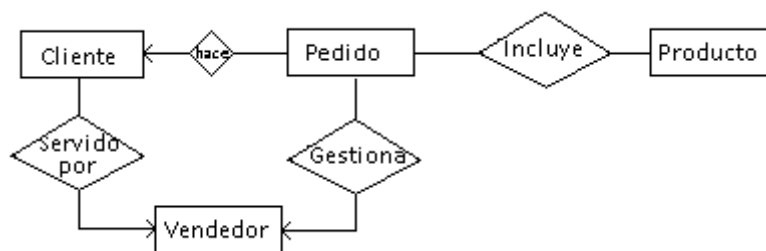
- Vista Cliente:



- Vista producto:



- Integración de las vistas:



3.3.3. Diseño lógico

Transformamos lo anterior en modelo relacional (dentro de lo posible, ya que lo anterior tiene más semántica).

Cliente	<u>DNI</u>	nombre_cliente	...	n-vend
	1	Paco	...	777

Vendedor	<u>n-vend</u>	nombre-vend	...
	666	Eddie	...
	777	Meléndez	...

Pedido	n-ped	fecha-ped	total-ped	...	DNI	n-vend
	1	10/10/1980	1000	...	1	666

Producto	n-prod	des-prod	...

¿Cómo asociamos cada pedido con un cliente? Introduciendo en la columna Pedido la clave primaria del cliente, en este caso DNI; y de igual manera con el n-vend. De igual forma para saber qué vendedor gestionó el pedido.

Otro aspecto que nos surge es decir qué productos tiene un pedido; pero para esto no sirve añadir columnas para cada producto de forma que n_prod, n_pro2... Para las relaciones entre pedidos y productos: 1 fila por cada producto de la siguiente forma:

Incluye	n-ped	n-prod
	1	A
	1	B

Son las claves externas. La clave es la **unión** de las dos claves externas.

En esta fase también hay una etapa de normalización, para lo que por ahora nos vale con eliminar la redundancia del modelo. Tal y como está ahora el ejemplo no tenemos redundancia así que añadiremos atributos para explicar lo que es la redundancia y cómo se elimina.

Vendedor	n-vend	...	Categoría	dias_vac
	666	...	C1	30
	222	...	C1	30

Hay repetición de innecesaria de información, lo que implica un desperdicio de espacio. Para eliminarlo el proceso de normalización crea lo siguiente:

Vendedor	n-vend	...	Categoría
	666	...	C1
	222	...	C1

Convencio vacaciones	Categoría	dias_vac
	C1	30
	C2	45

3.3.4. Refinamiento por el uso

En esta etapa nos centraremos en los procesos, adaptándolo de forma limitada teniendo en cuenta las necesidades de los procesos para mejorar el rendimiento. A la hora de realizar la adaptación habrá que favorecer unos procesos sobre otros, y para ello evaluamos la

prioridad de dichos procesos. Para obtener esa prioridad debemos realizar la evaluación teniendo en cuenta los siguientes criterios:

- Volumen de datos: A mayor volumen, mayor prioridad.
- Frecuencia del proceso: A mayor frecuencia, mayor prioridad.
- Complejidad del proceso, aunque tiene más importancia el volumen.
- Importancia para el negocio

Ejemplo: el proceso que controla la temperatura del reactor: poco volumen, pocos datos, frecuencia nula muy importante.

3.3.5. *Diseño físico*

Compone dos tareas principales: la definición de índices y el agrupamiento (clustering).

Los índices son algo que tienen todos los productos por lo que hay que realizarlo para todos. Son por donde buscamos: las claves y las claves externas. Aunque también se pueden definir índices por otros campos pero ocupan espacio y ralentizan las modificaciones por lo que hay que mirar si compensa. Esta tarea también está estandarizada en SQL.

El agrupamiento (clustering) consiste en agrupar los datos que muy probablemente se pretende acceder juntos. Ejemplo: dado un cliente, es posible que quieras saber qué cuentas tiene por lo que agruparemos en la misma zona de disco el cliente y sus cuentas.

3.3.6. *Implementación*

La implementación mediante el lenguaje de definición de datos creamos el esquema.

Ejemplo:

```
CREATE TABLE Cliente
(
    DNI INTEGER,
    nombre_cuenta CHAR(30),
    ...
    n-vend,
    ...
    FOREIGN KEY n-vend
        REFERENCES vendedor
);
```

Con FOREIGN KEY podemos hacer restricciones de integridad al añadir una clave externa. Ya que con la implementación también podemos implementar las restricciones. Por ejemplo: al intentar añadir un vendedor inexistente a un cliente, el motor de evaluación consultará el diccionario de datos, verá que n-vend es una clave externa, irá a la tabla de vendedor, buscará, no lo encontrará y por ello no lo dejará añadir.

3.3.7. *Pruebas*

Lo probaremos con las dos técnicas que nombramos anteriormente:

- Monitorización: realizar cambios a nivel físico de forma que no haya que tocar las aplicaciones para mejorar el funcionamiento.

21 de febrero de 2008

- Mantenimiento: si lo hemos hecho todo bien el mantenimiento será mucho más barato.

Parte 3: Diseño conceptual: Modelo Entidad - Relación

Tema 4: Modelo Entidad - Relación

4.1. Introducción y elementos fundamentales

Tiene sus orígenes en los años 70 con Peter Chen. Es un modelo **lógico** basado **en objetos**, destinado a la fase de diseño lógico, pero en general es un modelo conceptual para realizar diseño conceptual. Dispone de notación gráfica; si bien no está estandarizada, de alto nivel, de forma que se puede entender los gráficos sin necesidad de demasiados conocimientos.




4.1.1. Entidades y conjuntos de entidades

Una **entidad** es una cosa que existe por sí misma y se puede distinguir de otras. Pueden ser concretas o abstractas. Las entidades deben pertenecer a un conjunto de entidades (clases).

Las entidades del universo de discurso tienen atributos que las describen, aquellos que tienen sentido en el universo del discurso al que pertenecen. Todas las entidades pertenecientes a un conjunto de entidades tienen los mismos atributos. Estos atributos tienen un dominio de valores. Ejemplo: conjunto: {{{(DNI, 1), (nombre-cli, "Paco")}, {(DNI, 2), (nombre-cli, "Pepe")}}}. Aunque el orden de los atributos no importa se designa un orden por omisión definido en algún sitio.

Aunque los términos correctos son "conjunto de entidades" y "entidades", en ocasiones se usa "entidades" y "ocurrencias". Y a veces se mezclan varias. También es importante tener en cuenta que puede haber conjuntos varios de un mismo esquema

La notación es la siguiente:

- Conjunto de entidades: 
- Atributos: 
- Atributos identificativos (claves): 

Ejemplo:



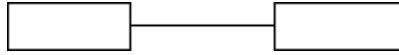
4.1.2. Relaciones y conjunto de relaciones

Las entidades pueden asociarse mediante una **relación**. Varias relaciones con el mismo significado forman **un conjunto de relaciones**.

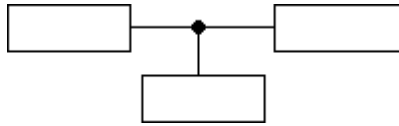
Entre dos conjuntos de entidades no tiene por qué haber un único conjunto de relaciones, sino tantas como asociaciones de distinto tipo existan. El máximo de relaciones

entre dos conjuntos de entidades es el producto cartesiano de las entidades. El grado es el número de conjuntos que asocia una relación. Según el grado, las relaciones pueden ser:

- Binarias: relaciones entre dos entidades (lo más usado):

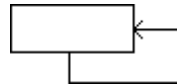


- Ternarias: relaciones entre tres entidades (poco frecuente):



- N-arias: relaciones entre n entidades (muy infrecuente).

- Unarias/reflexivas: asocian a la misma entidad:



Del mismo modo que las entidades tienen atributos, las relaciones también pueden tenerlo. Pero el valor de cada atributo dependerá de la relación concreta. Por ejemplo: en la relación “es titular de”, el atributo “última fecha de acceso”: Si Juan y Pepe son titulares de la misma cuenta cada uno tendrá su propia última fecha de acceso; si se quisiera disponer únicamente de la última fecha de acceso general se quitaría de la relación y se pondría en cuenta.

4.2. Restricciones de integridad

4.2.1. Restricciones de cardinalidad: uno / muchos

Indica, dentro de un mismo conjunto de relaciones con cuántas entidades se relaciona del conjunto destino. Generalmente la cardinalidad es de 1 a n. La representación:

- N: Se representa con: —. No hay restricción
- 1: Se representa con: —>. Hay restricción, máximo 1.



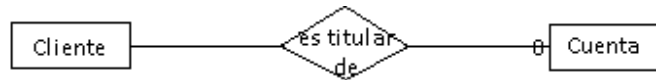
Sólo un cliente cada cuenta, cardinalidad 1.

Los diferentes tipos de relaciones son las siguientes:

- Relación muchos-a-muchos: Diagrama que muestra dos rectángulos (entidades) conectados por una línea horizontal (relación) que tiene un punto central, con una línea vertical descendente que conecta con un cuarto rectángulo (entidad).
- Relación muchos-a-uno: Diagrama que muestra dos rectángulos (entidades) conectados por una línea horizontal (relación) que tiene un punto central, con una línea vertical descendente que conecta con un cuarto rectángulo (entidad).
- Relación uno-a-uno: Diagrama que muestra dos rectángulos (entidades) conectados por una línea horizontal (relación) que tiene un punto central, con una línea vertical descendente que conecta con un cuarto rectángulo (entidad).
- Relaciones uno-a-muchos: Diagrama que muestra dos rectángulos (entidades) conectados por una línea horizontal (relación) que tiene un punto central, con una línea vertical descendente que conecta con un cuarto rectángulo (entidad).

La cardinalidad depende también según el universo del discurso. Cuando no se dice nada, interpretamos la cardinalidad como máxima. También hay cardinalidad mínima:

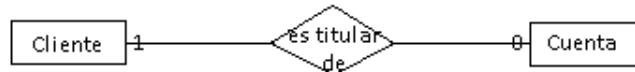
- 0: Ninguna, no hay restricción. En el ejemplo, un cliente no tiene porqué tener cuenta:



- 1: Tiene que haber al menos una, restricción. En el ejemplo, toda cuenta debe tener al menos un cliente (titular).



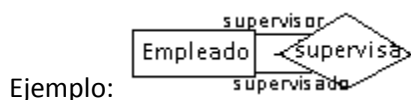
- Combinando los dos obtenemos la relación en que cada cliente podrá ser titular de entre 0 y 1 cuenta. A su vez, las cuentas pertenecerán a un único cliente, y toda cuenta pertenecerá a un cliente, al menos.



- Rangos: En el caso de querer poner valores diferentes se utilizan rangos (m,n):



También hay que tener en cuenta que pueden tener papel o rol que es la etiqueta que se puede poner en un extremo de una relación para describirla mejor. Se utiliza para aclarar bien una relación que podría no ser del todo clara sin ella, como por ejemplo, en relaciones reflexivas (sobre la propia entidad).



4.2.2. Superclaves, claves candidato y clave primaria

Una clave es el conjunto de atributos que permiten identificar de forma única una entidad en un conjunto de entidades.

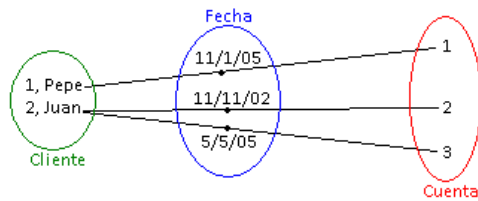
- **Superclave:** No se puede repetir. Ejemplo: {DNI} o {DNI, ciudad}.
- **Clave candidato:** Superclave en la que no hay atributos superfluos. Ejemplo: {DNI} lo es. {DNI, ciudad} no lo es.
- **Clave primaria:** De todas las claves candidato se escoge una para hacer de forma primaria o principal de identificación. En este caso DNI.

4.2.3. Clave primaria de un conjunto de relaciones

Será igual la relación que sin ellas, pero dependiendo del universo del discurso el atributo puede formar parte de la clave. Si perteneciese, permitiría asociar dos entidades con la misma clave.

El atributo de una relación es un dato que describe la relación. Ejemplo: si la relación entre cuenta y cliente tuviese un atributo "fecha_acceso" considerado como la última fecha en la que se accedió a la cuenta no contaría (se sobrescribiría). En cambio, si se registrasen todos los accesos, sí pertenecería a la clave.

$[CIP(E_1) \ CIP(E_2) \ \{A_1, A_2, \dots, A_n\}] \rightarrow$ es una Superclave]



DNI	Nº cuenta	fecha
1	1	10/01/2005
2	2	11/11/2002
2	3	05/05/2005

Puede haber diferentes relaciones:

- N-n: La clave candidato es DNI · Nº cuenta. No puede haber repetidos $\rightarrow CIP(E_1) \cdot CIP(E_2)$.
- 1-n: La clave candidato es nº cuenta. Ya que sólo puede tener un cliente cada cuenta, por tanto, cada cuenta sólo puede salir una vez: clave. Cada vez que haya 1-n el n es la clave candidato.
- 1-1: Salen dos claves candidato: cliente y nº cuenta. Las dos unidas son Superclave: un clave candidato por entidad.

COMO MÍNIMO SIEMPRE HAY UNA CLAVE PRIMARIA.

4.2.4. Dependencia por existencia.

Sucede cuando la existencia de una entidad depende de la existencia de otra entidad.

Por ejemplo:

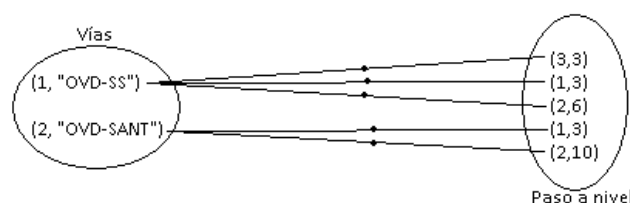


La existencia del paso a nivel depende de la existencia de la vía. Parecido a la herencia. Cuando se borre la vía se debería borrar el paso a nivel. Gráficamente se indica con la línea que une las una más gruesa de lo normal.

4.2.5. Entidades fuertes y entidades débiles

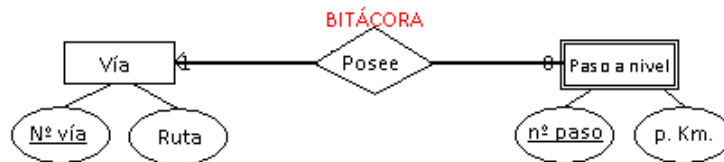
Las **entidades débiles** son entidades que por sí mismas no tienen atributos suficientes para formar claves. Para distinguirlas se utiliza un discriminador, que es la clave que permite distinguir entre entidades de un conjunto de entidades débiles. La clave primaria de un conjunto de entidades débiles está formada por la clave primaria de la entidad de la entidad de la que depende y su discriminador.

Los conjuntos de entidades débiles sí tienen clave primaria por, si bien se construye de forma diferente que las entidades fuertes. Ejemplo:



Para distinguir los pasos a nivel (ya que como vemos hay dos (1,3)) se necesita acudir a entidades externas, ya que aunque se forme la clave primaria con todos los atributos no se podría distinguir entre algunas debido a que Paso a Nivel es una entidad débil.

Ahora bien, dentro del conjunto de pasos a nivel de una vía pueden distinguirse por ser discriminador el número de paso a nivel



De esta manera observamos: nº vía es la clave primaria de la entidad a la que pertenece. Nº paso es el discriminador. A la relación se llama bitácora, es un nombre genérico para las relaciones entre entidades débiles y fuertes.

Hay que tener en cuenta que las entidades débiles dependen de otras, que a su vez puede ser débil o fuerte.

Pasando a tablas el último ejemplo quedaría de la siguiente forma:

Paso a Nivel	<u>nº vía</u>	<u>nº paso</u>	p. Km.
	1	1	3
	1	2	6
	1	3	3

Por otra parte, las **entidades fuertes** no dependen de otra entidad para tener clave. Partiendo del ejemplo anterior tendríamos:

Vía	<u>nº vía</u>	ruta
	1	"OVD-SS"
		"OVD-SANT"

La bitácora no hace falta ponerla, la información ya está en la tabla de la entidad débil.

4.3. Reducción de diagramas E-R en tablas

Cada conjunto del diagrama generará una tabla con igual nombre. A continuación del nombre la clave primaria y el resto de atributos.



Partiendo de este ejemplo lo pasaremos a tablas.

Ciente	DNI	nombre
	1	Paco
	2	Carmen

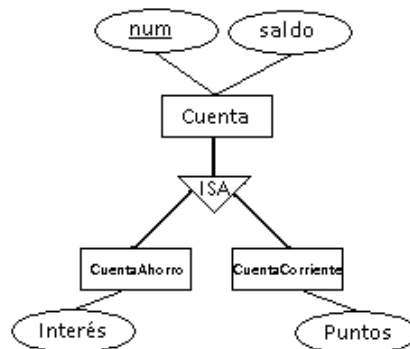
Cuenta	nº cuenta	Saldo	DNI	Fecha
	1	100	1	10/10/2004
	2	200	1	11/11/2005
	3	300	2	11/10/2002
	4	400	-	-

Si no fuera obligatorio que una cuenta tuviera un titular (relación 0-N) se dejan nulos los campos del dueño de la cuenta. Añadiendo “fecha” en la cuenta, no es necesario poner la tabla de titular. Esto sucede cuando se da relación con final en: →.

4.4. Construcciones avanzadas

4.4.1. Generalización

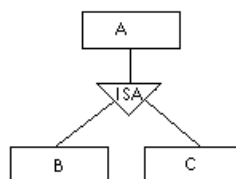
Es como la herencia de Java. Cada entidad de un nivel más alto debe pertenecer a un conjunto de entidades más bajo; en el ejemplo: puede existir una Cuenta Corriente o una Cuenta de Ahorro pero no una “Cuenta”, sólo sus hijos.




La relación con Cuenta e ISA tiene que ser con un trazo grueso. Es de tipo TOTAL.

4.4.2. Especialización

Es parecido a la generalización pero en este caso es de tipo PARCIAL; eso significa que algunas entidades de nivel más alto pueden no pertenecer a algún conjunto de entidades de nivel más bajo. Para representarlo la línea se dibuja más fina.



Los conjuntos de entidades de nivel más bajo pueden ser:

- Disjuntos: una entidad no puede pertenecer a más de un conjunto de entidades de nivel más bajo. Es decir o es B o es C. (Como el ejemplo)
- Solapados: La misma entidad puede pertenecer a más de un conjunto de entidades de nivel más bajo en una generalización simple. Se representa: .

4.4.3. Paso a tablas de especialización y generalización

A tablas sería algo como:

- Cuenta(nº cuenta, saldo)
- CuentaAhorro(nº cuenta, interés)
- CuentaCorriente(nº cuenta, puntos)

Vale tanto para generalización como para especificación pero hace falta explicar cómo interpretarlo ya que se pierde información al pasar a tablas.

Si sólo hay CuentaAhorro y CuentaCorriente y no hay cuentas a secas (generalización) podría hacerse lo siguiente:

- CuentaAhorro (nº cuenta, saldo, interés)
- CuentaCorriente(nº cuenta, saldo, puntos)

Otra forma sería hacer una sola tabla con todo y valores nulos para indicar qué tipo de cuenta es con algo así:

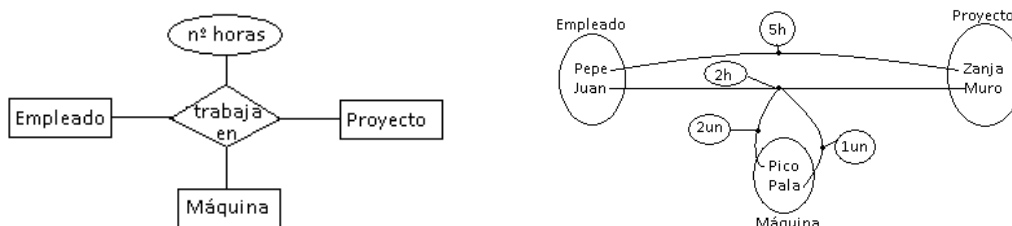
- Cuenta (nº cuenta, interés, puntos). Si puntos es nulo, no es una CuentaCorriente. Si interés es nulo, no es una CuentaAhorro y si ambos son nulos es una Cuenta.

Esta última opción tiene la ventaja de que no hay que buscar en varias tablas si no todo en la misma pero también es un gran inconveniente ya que si hay muchos valores nulos hay una mala gestión del espacio.

Si usamos tres tablas, para saber qué tipo de cuenta es habría que buscar por número de cuenta específicos para ver dónde están. Una solución a este problema podría ser introducir un atributo discriminador "tipoCuenta" en cuenta para que nos diga de qué tipo es; el problema es que hay que mantener la coherencia con el resto de la clase.

4.4.4. Agregación

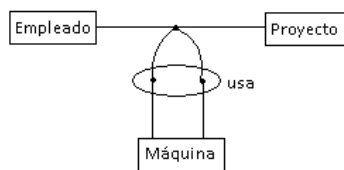
El modelo E-R prohíbe relaciones entre relaciones pero puede haber casos en los que convenga. Para ello surge la agregación. Para entenderlo, un ejemplo:



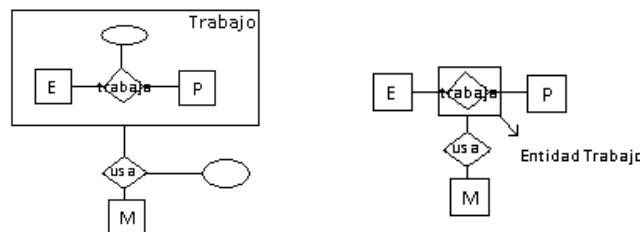
Es una relación de grado variable, según el número de máquinas que se use; y eso no es válido.

En la situación actual, ¿Cómo podemos asociar más de una máquina? ¿Asociaríamos una nueva máquina a la relación? Dejaría de ser genérico y necesitamos tener una relación ternaria para evitar, otra alternativa sería una relación ternaria pero ¿Cómo representaríamos si no se usa una máquina? No se le pueden quitar enlaces a la relación ternaria siempre tiene que haber. Podría ponerse con una máquina cualquiera y valor 0 o nulo, pero uno es malo diseño. Pero ninguna de las anteriores posibles soluciones nos vale por lo que es necesario la agregación.

La agregación consiste en convertir un conjunto de relaciones en un conjunto de entidades. Así:



Para ello, tendríamos dos opciones:



Pasando a tablas el ejemplo:

```

empleado (dni);
proyecto(c_proyecto,... );
máquina(c_maquina,...);
trabaja(dni, c-proyecto, n_h);
usa(dni,c-proyecto,c-maquina, n_unidades);

```

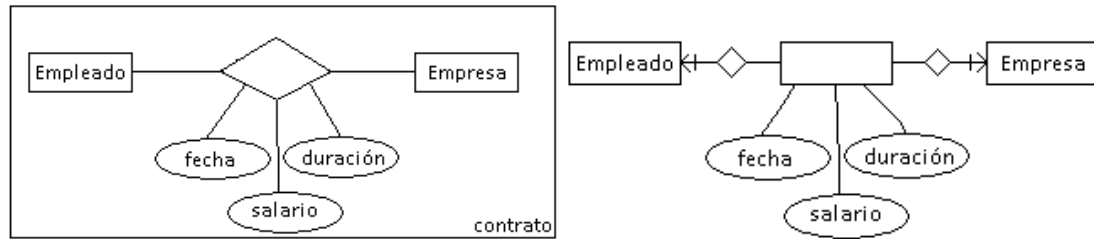
En caso de que un empleado sólo pudiera trabajar en un único trabajo:

```

trabaja(dni, c_proyecto, n_h);
usa(dni,c_maquina, n_unidades);

```

Vamos a ver un ejemplo de cómo representar lo mismo con agregación y con entidades con relaciones:



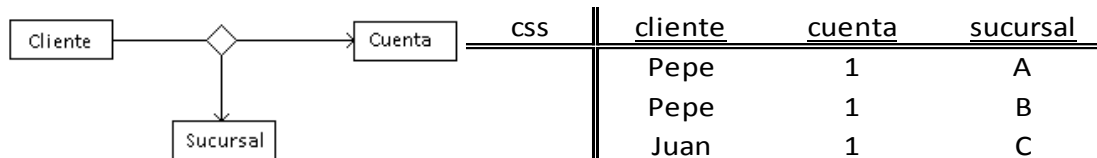
Ambas opciones son válidas aunque algunas metodologías recomiendan alguna más que otra.

Tema 5. Modelo Entidad- Relación extendido

5.1. Relaciones ternarias

Asocian tres entidades. Se utiliza cuando las tres relaciones son necesarias. Si una de las “patas” de la relación sólo aparece a veces no es una relación ternaria. Se podría usar una agregación en la dirección principal (las dos entidades siempre relacionadas).

En ocasiones tenemos cosas como las siguientes (ejemplo):



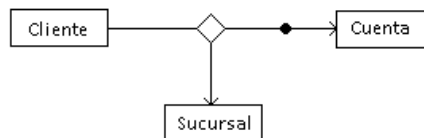
¿Cómo sería la cardinalidad?

- Una pareja cliente-cuenta sólo se puede asociar a una única sucursal.
- Una pareja cliente-sucursal sólo se puede asociar con una única cuenta.

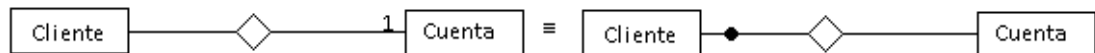
¿Habría cardinalidades mínimas?

- No tiene sentido indicarlo en las relaciones ternarias ya que son casos tan degenerados que casi nunca se darán.

¿Se podría obligar a unas entidades a salir en una relación?



- El punto indica que toda entidad cuenta debe participar al menos una vez en la relación, es decir, estar asociada a un cliente y a una sucursal. Como consecuencia, en relaciones binarias tenemos la siguiente equivalencia:



Como consecuencia de ello, en las tablas no se permitiría ninguna de las dos opciones:

<u>nombreCliente</u>	<u>n cuenta</u>	<u>n suc</u>
Pepe	1	A
Pepe	1	B
Pepe	2	A

Todo esto, debido a que Pepe sólo puede tener una cuenta en una sucursal. Por lo que no puede repetirse el número de cuenta ni el nombre de sucursal con el nombre de cliente.

5.2. Características extendidas

5.2.1. Atributos multivaluados

Los atributos en un diagrama E-R generalmente se asocian directamente en columnas para las tablas apropiadas. Sin embargo, los atributos multivaluados son una excepción ya que para estos atributos se crean tablas nuevas.

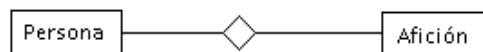
Ejemplo: afición = {fútbol, cine, teatro} a tablas:

Persona	DNI	Nombre	...	Afición
	1	Pepe	...	fútbol
	1	Pepe	...	cine

No está permitido ya que repetimos datos como por ejemplo el DNI 1 y el nombre Pepe para las dos aficiones. Por lo que debería ser así:

pers/afici	DNI	afición
	1	fútbol
	1	cine

En entidad-relación, una forma de representarlo menos compacta y que se lee peor es la siguiente:



La forma en la que lo representaremos nosotros es la siguiente:



Así damos la idea de que la "afición" sólo interesa con respecto a las personas. De la otra forma, también interesaría por sí sólo

5.2.2. Atributos compuestos

Un atributo compuesto sería por ejemplo fecha que le componen: día, mes y año. En las BBDD relacionales no existen los atributos compuestos. Aunque sí hay por compatibilidad hacia atrás al no poder hacer "fecha.día" se pierde mucho la relación, por lo que no se utilizan.

5.2.3. Atributos derivados

Se trata de atributos que se calculan a partir de otros. También se puede documentar la fórmula usada. Ejemplo de las formas de representarlo:

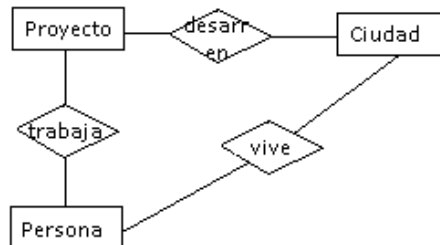


Estrictamente no hace falta guardar el valor ya que se calcula sobre la marcha. En caso de que la fórmula sea muy compleja (si hay que mirar muchas tablas para ello) se pre calcula, se guarda en la BD y, si cambian los datos, se recalcula; si se solicita se da la de la BD, que

actúa como una memoria caché. Hacerlo de una u otra forma dependerá de la tasa de actualización del atributo, de la complejidad de la fórmula, etc.

El problema de pre calcular es la inconsistencia, porque el atributo que usaste para calcular puede no estar actualizado correctamente o cualquier otro problema. De la otra forma, siempre es correcto.

También podría haber relaciones derivadas e incluso hasta entidades. Ejemplo:

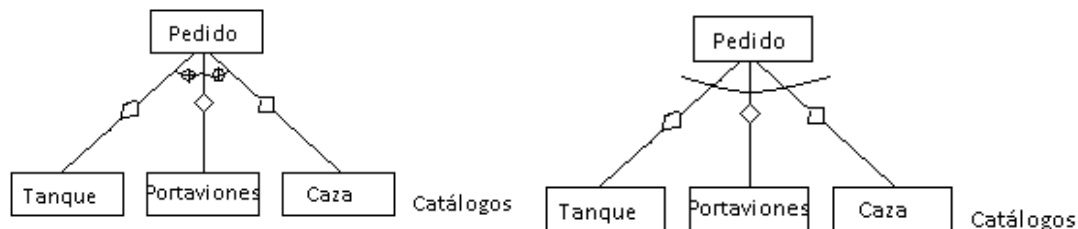


Por ejemplo: si por las características del proyecto tienes que vivir en la ciudad, la relación “vive” se puede calcular indirectamente por otro lado. Puede ser interesante reflejarlo para que se vea que existe la relación. También es interesante representar en el diagrama los atributos y demás derivados para que se sepa que existen, que tienen un nombre, que están en el diccionario de datos... y que sea derivado o no también depende de la semántica del atributo (por ejemplo: si no tiene nada que ver que trabajar en un proyecto y vivir en una ciudad no puede ser una relación derivada).

5.3. Restricciones que afectan a las relaciones

Veremos restricciones que se establecen entre un conjunto de entidades denominado raíz y un conjunto de entidades denominado hoja y con relaciones entre ellos.

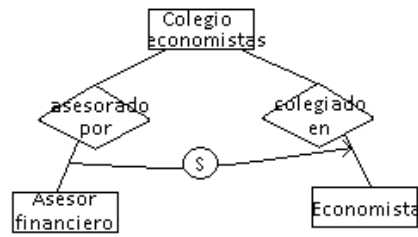
5.3.1. Exclusión



Con el ejemplo anterior no se pueden mezclar entre ellos elementos de un mismo pedido, es decir, si pides tanques, sólo puede haber tanques en el pedido. Por ello, si sale una relación por tanques no puede por las otras. Se representa con el círculo entre las relaciones involucradas. Sólo uno de los conjuntos puede tener elementos.

5.3.2. Subconjunto

Ejemplo: en los colegios de economistas hay colegiados economistas, y además son asesorados por asesores financieros. Los asesores van a ser economistas y queremos poner como restricción que han de estar colegiados. Por tanto, el conjunto de asesores ha de ser un subconjunto de los economistas colegiados.



Se fija una entidad raíz fija arriba y se calculan las relaciones con los de abajo.

Parte 4: Diseño lógico: Modelo relacional

Tema 6. Modelo relacional

6.1. Introducción

E.F. Codd el año 1969 inventó el modelo relacional.

El modelo relacional es un modelo lógico / conceptual orientado a registros, de más bajo nivel que el E-R. Desbancó a los anteriores modelos (jerárquico y en red) gracias a:

- Comodidad de uso: para los usuarios es fácil entender el concepto de tabla.
- Eficiencia: Llevan 30 años de optimización. Aunque al principio no hubo demasiada con el tiempo se ha logrado mucha.
- Fundamento matemático muy sólido. Está mucho más formalizado que otros modelos.

El modelo relacional suele coger los datos y organizarlos mediante tablas que nosotros llamaremos relaciones; los atributos, columnas; y las filas, tuplas.

6.2. Estructura de las BBDD relacionales

Primero haremos la relación:

- Tablas = relaciones
- Atributos = columnas
- Filas = tuplas

Ejemplo:

Cliente	DNI	nom_cli	...
	1	Paco	...
	2	Carmen	...

Las tablas se llaman relaciones porque lo que hacen es asociar, relacionan los atributos; esto proviene del concepto matemático de relación.

Las tuplas no tienen un orden, al igual que los atributos.

Hay que distinguir entre el esquema y las relaciones individuales: de un mismo esquema pueden declararse varias relaciones individuales. Ejemplo:

```
E_cliente=(DNI,nom_cli)
cliente(E_cliente)
cliente_moroso(E_cliente)
```

También debería especificarse el dominio de un atributo (aunque aquí podríamos deducirlo ya que está dentro del dominio de los dnis en general: un número de 10 cifras). En caso de que no lo conociéramos:

```
Dom_DNI = N
E_cliente=(DNI:Dom_DNI, nom_cli)
```

En las últimas versiones de SQL se pueden definir dominios, pero la mayoría lo sigue definiendo manualmente.

Las columnas deben tener un dominio, que debería especificarse. Aunque lo ideal es que el dominio sea gestionado por el sistema en ocasiones no se puede. Por ejemplo: los nombres de clientes podemos decir que son cadenas pero no podemos rechazar nombre como "vx36". Esto es labor del operario humano controlarlo. Los dominios restringen los valores posibles, es una restricción de integridad.

Las columnas de las tuplas no son multivaluadas.

6.2.1. Restricciones de las relaciones

1. Integridad de entidad: Garantiza que no hay elementos repetidos en un conjunto, es decir, que no existen tuplas repetidas en una relación, para ello no hay que meter claves repetidas.
2. Integridad referencial: Las claves externas sólo pueden tomar valores que estén en la tabla de la que son claves primarias. Siempre se cumple que el sistema rechaza las claves externas inexistentes.

Si estas dos relaciones NO se cumplen NO es una base de datos relacional.

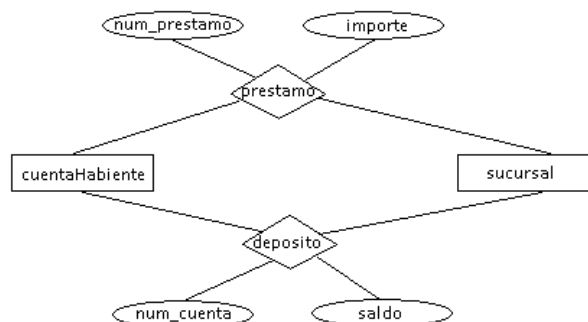
Ejemplo:

cuenta (nc, saldo, DNI)
cliente (DNI, nombre_cli, ...)

DNI relaciona las dos entidades.

Ahora veremos un ejemplo de modelo relacional con una base de datos:

sucursal (num_suc, nombre_suc, activo, ciudad);
deposito(num_suc, nombreCH, num_cuenta, saldo);
prestamo(num_prestamo, num_suc, nombreCH, importe);
cuentaHabiente(nombreCH, calle, ciudadCH);



Este ejemplo es el que usaremos en todos los ejemplos a partir de ahora.

- LMD:

- Consulta
- Modificación
- Lenguajes formales
 - Procedimentales
 - Declarativos

6.3. Lenguajes de consulta formales: álgebra relacional

Es un lenguaje algebraico: operadores y operandos. Aquí los operandos serán las relaciones.

6.3.1. Operaciones fundamentales

Las operaciones básicas son:

- Selección
- Proyección
- Producto cartesiano
- Unión
- Diferencia
- Renombramiento

Iremos describiendo una a una cada operación:

6.3.1.1. Selección (σ)

Devuelve un subconjunto de una relación. Es un filtro.

Ejemplo: $\sigma_{(ciudad="Oviedo" \wedge DNI > 9.200.000)}(cuentaHabiente) \rightarrow$ Como subíndice, la condición que se aplica a cada tupla.

6.3.1.2. Proyección (Π)

Para quedarte con las columnas, los atributos que se quieran.

Ejemplo: $\Pi_{nombreCH, calle}(cuentaHabiente)$ obtenemos:

resultado	nombreCH	calle
	Pepe	Uría
	Juan	Valdes Salás

6.3.1.3. Producto cartesiano (\times)

Haciendo el producto cartesiano de dos conjuntos se obtiene otro con todas las posibles combinaciones de los elementos de los dos conjuntos concatenados.

Ejemplo. Dadas estas 2 tablas:

CH	nombreCH	calle	ciudad
	Pepe	Uría	Oviedo
	Juan	Valdés Salás	Gijón

depósito	num_suc	nombreCH	num_cuenta	saldo
	A	Pepe	1	100
	B	Juan	2	200
	A	Juan	1	100

El resultado del proyecto cartesiano:

Chxdepósito	CH.nombreCH	calle	ciudad	num_cuenta	n_suc	dep.nombreCH	saldo
	Pepe	Uría	Oviedo	1	A	Pepe	100
	Pepe	Uría	Oviedo	2	B	Juan	200
	Pepe	Uría	Oviedo	1	A	Juan	100
	Juan	VS	Gijón	1	A	Pepe	100

Los atributos son la concatenación de los dos.

Es la operación más importante porque es la que nos permite relacionar dos tablas. Cuando hay ambigüedad en los nombres de las columnas se prefija: nombreTabla.atributo.

Es imposible hacer una selección de un producto cartesiano.

6.3.1.4. Renombramiento (ρ)

Se utiliza para dar nombre a los resultados del álgebra relacional y así referirse a ellos. Como las relaciones se consideran expresiones se pueden usar para darles un nuevo nombre.

Ejemplo: Sacar los nombre de los que viven donde los que se llaman pepe.

$CH(\underline{nCH}, \text{calle}, \text{ciudadCH})$

$\sigma_{CH.nCH="Pepe"}(CH \times CH)$. Se hace producto de una tabla consigo misma, así que para distinguir entre una tabla y la segunda tabla por lo que hay que renombrar además de repetir datos. La consulta debería ser:

$\sigma_{CH.calle=CHPepe.calle \wedge CH.ciudadCH=CHPepe.ciudadCH}(\sigma_{CHPepe.nCH="Pepe"}(CH \times \rho_{CHPepe}(CH)))$

Ahora sacamos los nombres de las personas que viven donde pepe. Si no queremos sacar los nombres de Pepes debemos añadir al primer σ " $\wedge CH.nCH \neq CHPepe.nCH \neq "Pepe"$ ".

6.3.1.5. Unión (\cup)

Las dos tablas a unir deben tener mismo número y mismo tipo de atributos; tienen que ser relaciones compatibles.

Ejemplo: sacar las personas que tienen cuenta, préstamo o ambos en una sucursal llamada Perryridge.

$\Pi_{InCH}(\sigma_{nsuc="Perryridge"}(\text{préstamo})) \cup \Pi_{InCH}(\sigma_{nsuc="Perryridge"}(\text{depósito}))$

6.3.1.6. Diferencia (−)

Permite buscar las tuplas que están en una relación pero no en la otra.

Ejemplo: sacar la gente que tiene cuenta pero no préstamo

$$\Pi_{\text{InCH}} (\sigma_{\text{nsuc}=\text{"Perryridge"}} (\text{préstamo})) - \Pi_{\text{InCH}} (\sigma_{\text{nsuc}=\text{"Perryridge"}} (\text{depósito}))$$

6.3.2. Operaciones adicionales

Son aquellas que se pueden expresar en base a operaciones fundamentales:

- Intersección
- Producto natural
- División
- Asignación

6.3.2.1. Intersección (\cap)

$A - (A - B)$. Al igual que en la unión, las relaciones tienen que ser compatibles.

Ejemplo: sacar nombres de clientes de Oviedo con saldo > 100

$$\Pi_{\text{CH.nCH}} (\sigma_{\text{CH.ciudad}=\text{"OVD"}} \text{deposito.saldo} > 100 (\sigma_{\text{CH.nCH}=\text{deposito.nCH}} (\text{CH} \times \text{deposito})))$$

6.3.2.2. Producto natural ($|><|$).

$A |><| B \equiv \Pi_{A \cup B} (\sigma_{A.a=B.a \wedge A.b=B.b \wedge \dots} (A \times B))$. Hace el producto cartesiano, se queda con las filas cuyos atributos comunes son iguales (generalmente, y si la BD está bien diseñada, según la clave primaria y la externa) y elimina las columnas repetidas).

Ejemplo: sacar nombres de clientes de Oviedo con saldo > 1000 (última consulta).

$$\Pi_{\text{CH.nCH}} (\sigma_{\text{CH.ciudad}=\text{"OVD"}} \wedge \text{deposito.saldo} > 1000 (\text{CH} |><| \text{deposito}))$$

Ejemplo: encontrar clientes con cunetas con saldo > 1000 en sucursales de Pekín y que el cliente sea de Oviedo.

$$\Pi_{\text{CH.nCH}} (\sigma_{\text{CH.ciudad}=\text{"OVD"}} \wedge \text{deposito.saldo} > 1000 \wedge \text{sucursal.ciudad}=\text{"Pekin"}} (\text{CH} |><| \text{deposito} |><| \text{sucursal}))$$

6.3.2.3. División (\div)

Permite conocer, por ejemplo, los clientes que tienen cuentas en todas las sucursales de una ciudad.

Ejemplo: Encontrar clientes que tienen cuenta en al menos una sucursal de Oviedo.

$$\Pi_{\text{InCH}} (\sigma_{\text{sucursal.ciudad}=\text{"OVD"}} (\text{deposito} |><| \text{sucursal}))$$

Ejemplo 2: Encontrar clientes que tienen cuenta en todas las sucursales de Brooklyn.

$$\Pi_{nCH, nsuc}(\text{deposito}) \div \Pi_{nsuc}(\sigma_{sucursal.ciudad = \text{"Brooklyn"}}(sucursal))$$

6.3.2.4. Asignación (\leftarrow)

Hace lo mismo que la asignación en los lenguajes de programación. Su evaluación no muestra ningún resultado

Ejemplo:

$$r \leftarrow \Pi_{nCH, nsuc}(\text{deposito})$$

$$s \leftarrow \Pi_{nsuc}(\sigma_{sucursal.ciudad = \text{"Brooklyn"}}(sucursal))$$

$$r \div s$$

6.3.3. Conclusión

Hay consultas en AR que no se pueden hacer, como por ejemplo: consultas recursivas, despieces (sacar todos los hijos de un nodo de un árbol, etc.). En estos casos, el SQL suele complementarse con un programa escrito en C, por ejemplo.

6.4. Cálculo relacional

Se da la definición o declaración del resultado. En este lenguaje tenemos:

- Tuplas: t. Las variables toman valores de tuplas.
- Dominios: x. Toman valores en dominios: nombre, calle...

6.4.1. Cálculo relacional de tuplas

Ejemplo: = es de COMPARACIÓN, no de asignación

$$\begin{aligned} t &\in CH \\ t[nCH] &= \text{"Pepe"} \\ t[nCH] &= c[nCH] \end{aligned}$$

Las consultas, en el cálculo relacional de tuplas se expresan como: $\{t / P(t)\}$, es decir, son el conjunto de todas las tuplas para que el predicado P es cierto.

Ejemplo: encontrar los clientes que viven en Oviedo:

$$\{t_{(nCH)} / t \in CH \wedge t[ciudadCH] = \text{"OVD"}\}$$

Si sólo quisiéramos el nombre:

$$\{t_{(nCH)} / \exists c \in CH (c[nCH] = t[nCH] \wedge c[ciudadCH] = \text{"OVD"})\}$$

Ejemplo: encontrar los clientes que viven en Oviedo y tienen saldo > 1000)

$$\{t_{(nCH)} / \exists c \in CH (c[nCH] = t[nCH] \wedge c[ciudadCH] = \text{"OVD"}) \wedge \exists d \in \text{deposito} (d[nCH] = c[nCH] \wedge d[saldo] > 1000)\}$$

Este último ejemplo en álgebra relacional sería:

$$\prod_{nCH} (\sigma_{CH.ciudadCH = "OVD" \wedge deposito.saldo > 1000} (CH \bowtie deposito))$$

En general encadenar dos existen (\exists) suele desembocar en producto cartesiano (\bowtie).

Casi lo mismo que se puede hacer con álgebra relacional se puede hacer en cálculo relacional, y viceversa. Seguimos con ejemplos:

Ejemplo: Clientes con cuenta o préstamo en Perryridge o con las dos cosas:

$$\{ t_{(nCH)} / \exists d \in \text{deposito} (t[nCH] = d[nCH] \wedge d[nSuc] = \text{"Perryridge"}) \vee \exists p \in \text{prestamo} (t[nCH] = p[nCH] \wedge p[nSuc] = \text{"Perryridge"}) \}$$

Ejemplo: Clientes con cuenta y préstamo. Igual que el ejemplo anterior pero con " \vee " en vez de " \wedge ".

Ejemplo: Clientes con cuenta pero sin préstamo

$$\{ t / \exists d \in \text{deposito} (...) \wedge \neg \exists p \in \text{préstamo} (...) \}$$

·Para todo (\forall) e implica \Rightarrow

Ejemplo: Clientes que tienen cuenta en todas las sucursales de Brooklyn.

$$\{ t_{(nCH)} / \forall s \in \text{sucursal} (s[ciudad_suc] = \text{"Brooklyn"} \Rightarrow (\exists d \in \text{deposito} (d[nsuc] = s[nsuc] \wedge d[nCH] = t[nCH]))) \}$$

Ejemplo 2: Clientes (y ciudad) que tienen cuenta en todas las sucursales de su ciudad.

$$\{ t_{(nCH)(ciudadCH)} / \exists c \in CH (c[nCH] = t[nCH] \wedge c[ciudadCH] = t[ciudadCH]) \wedge \forall s \in \text{sucursal} ((s[ciudad_suc] = c[ciudad_suc]) \Rightarrow \exists d \in \text{deposito} (d[nsuc] = s[nsuc] \wedge d[nCH] = t[nCH])) \}$$

Con el cálculo relacional podemos obtener infinitas tuplas si ponemos una condición que cumplan infinitas veces, como por ejemplo: cuentas que no estén en préstamo. Lo que vamos a hacer es restringir las expresiones que mira a expresiones SEGURAS: evaluables por el ordenador (si se pueden evaluar siendo valores que están en la BD solamente) y terminan siempre.

6.5. Las 12 reglas de codd

En la década de los 80 comenzaron a aparecer numerosos SGBD que se anunciaban como "relacionales". Sin embargo estos sistemas carecían de muchas características que se consideran importantes en un sistema relacional, perdiendo muchas ventajas del modelo relacional. Como ejemplo externo de esto "sistemas relacionales" eran simplemente sistemas que utilizaban tablas para almacenar la información, no disponiendo de elementos como claves primarias, etc.

En 1984 Codd publicó 12 reglas que un verdadero sistema relacional debería cumplir. En la práctica algunas de ellas son difíciles de realizar. Un sistema podría considerarse “más relacional” cuanto más siga esas reglas.

La regla 0 es aquella que dice: *“Para que un sistema se denomine sistema de gestión de bases de datos relacionales, este sistema debe usar (exclusivamente) sus capacidades relacionales para gestionar la base de datos”*.

6.5.1. Regla de la información

“Toda la información en una base de datos relacional se representa explícitamente en el nivel lógico exactamente de una manera: con valores en tablas”.

- Por tanto, los metadatos (diccionario, catálogo) se representan exactamente igual que los datos de usuario.
- Puede usarse el mismo lenguaje (como SQL) para acceder a los datos y a los metadatos (regla 4)
- Un valor posible es el valor nulo, con sus dos interpretaciones:
 - o Valor desconocido (ejemplo: dirección desconocida)
 - o Valor no aplicable (ejemplo: empleado soltero no tiene esposa)

6.5.2. Regla del acceso garantizado

“Para todos y cada uno de los datos (valores atómicos) de una Base de Datos Relacional se garantiza que son accesibles a nivel lógico utilizando una combinación de nombre de tabla, valor de clave primaria y nombre de columna”

- Cualquier dato almacenado en una Base de Datos Relacional tiene que poder ser direccionado unívocamente. Para ello hay que indicar: en qué tabla está, cuál es la columna y cuál es la fila (mediante la clave primaria).
- Por tanto se necesita el concepto de clave primaria, que no es soportado en muchas implementaciones. En estos casos, para lograr un efecto similar se puede hacer lo siguiente:
 - o Hacer que los atributos clave primaria no puedan ser nulos (NOT NULL)
 - o Crear un índice único sobre la clave primaria.
 - o No eliminar nunca el índice.

6.5.3. Tratamiento sistemático de valores nulos

“Los valores nulos (que son distintos a la cadena vacía, blancos, 0,...) se soportan en los SGBD totalmente relacionales para representar información desconocida o no aplicable de manera sistemática, independientemente del tipo de datos.”

- Se reconoce la necesidad de la existencia de valores nulos, para un tratamiento sistemático de los mismos.
- Hay problemas para soportar los valores nulos en las operaciones relacionales, especialmente en las operaciones lógicas.
- Lógica trivaluada: Es una posible solución. Existen tres (no dos) valores de verdad: Verdadero, Falso y Desconocido (null). Se crean tablas de verdad para las operaciones lógicas:

- Null Y null= null
- Verdadero Y null = null
- Falso Y null = Falso
- Verdadero O null = Verdadero
- Etc.
- Un inconveniente es que de cara al usuario el manejo de los lenguajes relacionales se complica pues es más difícil de entender.

6.5.4. Catálogo dinámico en línea basado en el modelo relacional

“La descripción de la base de datos se representa a nivel lógico a la misma manera que los datos normales, de modo que los usuarios autorizados pueden aplicar el mismo lenguaje relacional a su consulta, igual que lo aplican a los datos normales.”

- En consecuencia de la regla 1 que se destaca por su importancia. Los metadatos se almacenan usando el modelo relacional, con todas sus consecuencias.

6.5.5. Regla del sublenguaje de datos completo

“Un sistema relacional debe soportar varios lenguajes y varios modos de uso de terminal (ejemplo: rellenar formularios, etc.). Sin embargo, debe existir al menos un lenguaje cuyas sentencias sean expresables, mediante una sintaxis bien definida, con cadenas de caracteres y que sea completo, soportando:

- Definición de datos
- Definición de vistas
- Manipulación de datos (interactiva y por programa)
- Limitantes de integridad
- Limitantes de transacciones (iniciar, realizar, deshacer) (Begin, commit, rollback).”
- Además de poder tener interfaces más amigables para hacer consultas, etc. Siempre debe de haber una manera de hacerlo todo de manera textual, que es tanto como decir que pueda ser incorporado en un programa tradicional.
- Un lenguaje que cumple esto en gran medida es SQL.

6.5.6. Regla de actualización de vistas

“Todas las vistas que son teóricamente actualizables se pueden actualizar por el sistema.”

- El problema es determinar cuáles son las vistas teóricamente actualizables, ya que no está muy claro.
- Cada sistema puede hacer unas suposiciones particulares sobre las vistas que son actualizables.

6.5.7. *Inserción, actualización y borrado de alto nivel*

“La capacidad de manejar una relación base o derivada como un solo operando se aplica no sólo a la recuperación de datos (consultas), sino también a la inserción, actualización y borrado de datos.”

- Esto es, el lenguaje de manejo de datos también debe ser de alto nivel (de conjuntos). Algunas bases de datos inicialmente sólo podían modificar las tuplas de las bases de datos de una en una (un registro de cada vez).

6.5.8. *Independencia física de datos*

“Los programas de aplicación y actividades del terminal permanecen inalterados a nivel lógico cuandoquiera que se realice cambios en las representaciones de almacenamiento o métodos de acceso.”

- El modelo relacional es un modelo lógico de datos, y oculta las características de su representación física

6.5.9. *Independencia lógica de datos*

“Los programas de aplicación y actividades del terminal permanecen inalterados a nivel lógico cuandoquiera que se realicen cambios a las tablas base que preserven la información.”

- Cuando se modifica el esquema lógico preservando información (no valdría por ejemplo eliminar un atributo) no es necesario modificar nada en niveles superiores.
- Ejemplos de cambios que preservan la información:
 - Añadir un atributo a una tabla base.
 - Sustituir dos tablas base por la unión de las mismas. Usando vistas de la unión puedo recrear las tablas anteriores.

6.5.10. *Independencia de integridad*

“Los limitantes de integridad específicos para una determinada base de datos relacional deben poder ser definidos en el sublenguaje de datos relacional, y almacenables en el catálogo, no en los programas de aplicación.”

- El objetivo de las bases de datos no es sólo almacenar los datos, sino también sus relaciones y evitar que estas (limitantes) se codifiquen en los programas. Por tanto en una base de datos relacional se debe poder definir limitantes de integridad.
- Cada vez se van ampliando más los tipos de limitantes de integridad que se pueden utilizar en los SGBDR, aunque hace poco eran muy escasos.
- Como parte de los limitantes inherentes al modelo relacional (forman parte de su definición) están:
 - Una BDR tiene integridad de entidad. Es decir, toda tabla debe tener una tabla primaria.
 - Una BDR tiene integridad referencial. Es decir, toda clave externa no nula debe existir en la relación donde es primaria.

6.5.11. Independencia de distribución

“Una BDR tiene independencia de distribución.”

- Las mismas órdenes y programas se ejecutan igual en una BD centralizada que en una distribuida.
- Las BDR son fácilmente distribuibles:
 - o Se parten las tablas en fragmentos que se distribuyen.
 - o Cuando se necesitan las tablas completas se recombinan usando operaciones relacionales con los fragmentos.
 - o Sin embargo se complica más la gestión interna de la integridad, etc.
- Esta regla es responsable de tres tipos de transparencia de distribución:
 - o Transparencia de localización. El usuario tiene la impresión de que trabaja con una BD local (aspecto de la regla de independencia física).
 - o Transparencia de fragmentación. El usuario no se da cuenta de que la relación con qué trabaja está fragmentada (aspecto de la regla de independencia lógica de datos).
 - o Transparencia de replicación. El usuario no se da cuenta de que pueden existir copias (réplicas) de una misma relación en diferentes lugares.

6.5.12. Regla de la no subversión

“Si un sistema relacional tiene un lenguaje de bajo nivel (un registro de cada vez), ese bajo nivel no puede ser usado para saltarse (subvertir) las reglas de integridad y los limitantes expresados en los lenguajes relacionales de más alto nivel (una relación (conjunto de registros) de cada vez).”

- Algunos problemas no se pueden solucionar directamente con el lenguaje de alto nivel.
- Normalmente se usa SQL inmerso en un lenguaje anfitrión para solucionar estos problemas. Se utiliza el concepto de cursos para tratar individualmente las tuplas de una relación. En cualquier caso no debe ser posible saltarse los limitantes de integridad impuestos al tratar tuplas a ese nivel.

Ejercicios (E-R y paso a tablas)

1. Ejercicio 1

“La secretaria del centro quiere una BD de gestión académica con datos sobre alumnos, asignaturas, profesores y sesiones de clase.

- *Profesores: nº registro, DNI, nombre.*
- *Alumnos: DNI, nombre, nº matrícula (expediente)*
- *Asignatura: número de matriculados, código Asignatura, descripción.*

Ha que saber nota del alumno en cada asignatura y qué profesor da cada asignatura.

- *Sesiones: hora inicio, duración, lugar, día.”*

En primer lugar, cuando el enunciado dice cosas del estilo “la secretaria del centro” no hay que poner una entidad para secretaría puesto que no va a haber más de una por lo que todo lo que hagamos se va a referir a la misma.

Asignatura es una entidad ya que es interesante hace consultas sobre ella, hay información descriptiva lo que, por el momento, hace pensar que es una entidad.