4. DISEÑO DE BASES DE DATOS RELACIONALES

Introducción

4.1 Definición del problema

4.2 Solución de problemas

4.3 Normalización: 1NF, 2NF, 3FN

4.4 Criterios para normalizar

Introducción

Como ya hemos visto en los Subtemas Nos. 2.4 y 3.4 los Modelos Relacionales son de los utilizados muy ampliamente y recordando que el modelo es la base (core) para los DBMS es importante refrendar los conceptos básicos y de donde vienen.

Muchas disciplinas (y sus metodologías de diseño asociadas) tienen algún tipo de base teórica. Los ingenieros industriales diseñan estructuras utilizando teorías de la física. Los compositores crean sinfonías utilizando conceptos de teoría de la música. La industria del automóvil utiliza teorías de la aerodinámica para diseñar automóviles con menor consumo. La industria aeronáutica utiliza las mismas teorías para diseñar alas de aviones que reduzcan la resistencia al viento.

Estos ejemplos demuestran que la teoría es muy importante. La ventaja principal de la teoría es que hace que las cosas sean predecibles: nos permite predecir qué ocurrirá si realizamos una determinada acción. Por ejemplo, sabemos que si soltamos una piedra, caerá al suelo. Si somos rápidos, podemos apartar nuestros pies del camino de la teoría de la gravedad de Newton. Lo importante es que siempre funciona. Si ponemos una piedra plana encima de otra piedra plana, podemos predecir que se quedarán tal y como las hemos puesto. Esta teoría permite diseñar pirámides, catedrales y casas de ladrillos. Consideremos ahora el ejemplo de una base de datos relacional. Sabemos que si un par de tablas están relacionadas, podemos extraer datos de las dos a la vez, simplemente por el modo en que funciona la teoría de las }bases de datos relacionales. Los datos que se saquen de las dos tablas se basarán en los valores coincidentes del campo que ambas tienen en común. Una vez más, nuestras acciones tienen un resultado predecible.

El modelo relacional se basa en dos ramas de las matemáticas: la teoría de conjuntos y la lógica de predicados de primer orden. El hecho de que el modelo relacional esté basado en la teoría de las matemáticas es lo que lo hace tan seguro y robusto. Al mismo tiempo, estas ramas de las

matemáticas proporcionan los elementos básicos necesarios para crear una base de datos relacional con una buena estructura, y proporcionan las líneas que se utilizan para formular buenas metodologías de diseño.

Hay quien ofrece una cierta resistencia a estudiar complicados conceptos matemáticos para tan sólo llevar a cabo una tarea bastante concreta. Es habitual escuchar quejas sobre que las teorías matemáticas en las que se basa el modelo relacional y sus metodologías de diseño, no tienen relevancia en el mundo real o que no son prácticas. No es cierto: las matemáticas son básicas en el modelo relacional. Pero, por fortuna, no hay que aprender teoría de conjuntos o lógica de predicados de primer orden para utilizar el modelo relacional. Sería como decir que hay que saber todos los detalles de la aerodinámica para poder conducir un automóvil. Las teorías de la aerodinámica ayudan a entender cómo un automóvil puede ahorrar combustible, pero desde luego no son necesarias para manejarlo.

La teoría matemática proporciona la base para el modelo relacional y, por lo tanto, hace que el modelo sea predecible, fiable y seguro. La teoría describe los elementos básicos que se utilizan para crear una base de datos relacional y proporciona las líneas a seguir para construirla. El organizar estos elementos para conseguir el resultado deseado es lo que se denomina diseño.

En 1970, el modo en que se veían las bases de datos cambió por completo cuando E. F. Codd introdujo el modelo relacional. En aquellos momentos, el enfoque existente para la estructura de las bases de datos utilizaba punteros físicos (direcciones de disco) para relacionar registros de distintos ficheros. Si, por ejemplo, se quería relacionar un registro con un registro , se debía añadir al registro un campo conteniendo la dirección en disco del registro . Este campo añadido, un puntero físico, siempre señalaría desde el registro al registro . Codd demostró que estas bases de datos limitaban en gran medida los tipos de operaciones que los usuarios podían realizar sobre los datos. Además, estas bases de datos eran muy vulnerables a cambios en el entorno físico. Si se añadían los controladores de un nuevo disco al sistema y los datos se movían de una localización física a otra, se requería una conversión de los ficheros de datos. Estos sistemas se basaban en el modelo de red y el modelo jerárquico, los dos modelos lógicos que constituyeron la primera generación de los DBMS.

El modelo relacional representa la segunda generación de los DBMS. En él, todos los datos están estructurados a nivel lógico como tablas formadas por filas y columnas, aunque a nivel físico pueden tener una estructura completamente distinta. Un punto fuerte del modelo relacional es la sencillez de su estructura lógica. Pero detrás de esa simple estructura hay un fundamento teórico importante del que carecen los DBMS de la primera generación, lo que constituye otro punto a su favor.

Dada la popularidad del modelo relacional, muchos sistemas de la primera generación se han modificado para proporcionar una interfaz de usuario relacional, con independencia del modelo lógico que soportan (de red o jerárquico). Por ejemplo, el sistema de red IDMS ha evolucionado a IDMS/R e IDMS/SQL, ofreciendo una visión relacional de los datos.

En los últimos años, se han propuesto algunas extensiones al modelo relacional para capturar mejor el significado de los datos, para disponer de los conceptos de la orientación a objetos y para disponer de capacidad deductiva.

El modelo relacional, como todo modelo de datos, tiene que ver con tres aspectos de los datos:

- Estructura de datos.
- Integridad de datos.
- Manejo de datos.

4.1 Definición del problema

La definición del problema es el proceso por el que se determina la organización de una base de datos, incluidos su estructura, contenido y las aplicaciones que se han de desarrollar. Durante mucho tiempo, el diseño de bases de datos fue considerado una tarea para expertos: más un arte que una ciencia. Sin embargo, se ha progresado mucho en el diseño de bases de datos y éste se considera ahora una disciplina estable, con métodos y técnicas propios. Debido a la creciente aceptación de las bases de datos por parte de la industria y el gobierno en el plano comercial, y a una variedad de aplicaciones científicas y técnicas, el diseño de bases de datos desempeña un papel central en el empleo de los recursos de información en la mayoría de las organizaciones. El diseño de bases de datos ha pasado a constituir parte de la formación general de los informáticos, en el mismo nivel que la capacidad de construir algoritmos usando un lenguaje de programación convencional

Para definir correctamente al Problema lo primero es realizar diseño conceptual, que parte de las especificaciones de los requisitos del usuario y su resultado es el esquema conceptual de la base de datos que corresponderá a un Modelo Entidad – Relación (E / R). Un *esquema conceptual* es una descripción de alto nivel de la estructura de la base de datos, independientemente del DBMS que se vaya a utilizar para manipularla. Un *modelo conceptual* es un lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es describir el contenido de los Datos de la base de datos (DB) y no las

estructuras de almacenamiento que se necesitarán para manejar esta información.

El modelo relacional representa un sistema de bases de datos en un nivel de abstracción un tanto alejado de los detalles de la máquina subyacente, de la misma manera como, por ejemplo, un lenguaje del tipo de PL/1 representa un sistema de programación con un nivel de abstracción un tanto alejado de los detalles de la máquina subyacente. De hecho, el modelo relacional puede considerarse como un lenguaje de programación mas bien abstracto, orientado de manera específica hacia las aplicaciones de bases de datos [Date, 1993]

En términos tradicionales una relación se asemeja a un archivo, una tupla a un registro, y un atributo a un campo. Pero estas correspondencias son aproximadas, en el mejor de los casos. Una relación no debe considerase como ``solo un archivo'', sino mas bien como un archivo disciplinado, siendo el resultado de esta disciplina una simplificación considerable de las estructuras de datos con las cuales debe interactuar el usuario, lo cual a su vez simplifica los operadores requeridos para manejar esas estructuras.

Características principales de los ``archivos'' relacionales:

- Cada ``archivo'' contiene solo un tipo de registros
- Los campos no tienen un orden específico, de izquierda a derecha
- Los registros no tienen un orden específico, de arriba hacia abajo
- Cada campo tiene un solo valor
- Los registros poseen un campo identificador único (o combinación de campos) llamado clave primaria

Así, todos los datos en una base de datos relacional se representan de una y solo una manera, a saber, por su valor explícito (esta se denomina en ocasiones ``principio básico del modelo relacional''). En particular, las conexiones lógicas dentro de una relación y entre las relaciones se representan mediante esos valores; no existen ``ligas'' o apuntadores visibles para el usuario, ni ordenamientos visibles para el usuario, ni grupos repetitivos visibles para el usuario, etc.

Actualmente algunos de los manejadores de bases de datos, utilizan un sistema de búsqueda con algoritmos de árboles b. Pero las búsquedas que se pueden realizar con estos algoritmos son sólo para memoria principal.

Los algoritmos implementados para realizar búsquedas con listas salteadas o por bloques (*skip lists*) son eficientes para realizar búsquedas en memoria secundaria. Como tienen varios niveles en cada

nodo de la lista, nos permite dar saltos mas largos al realizar las búsquedas, esto provoca que las sean mas rápidas.

El primer paso para la definición del Problema l diseño de una base de datos es la producción del esquema conceptual. Normalmente, se construyen varios esquemas conceptuales, cada uno para representar las distintas visiones que los usuarios tienen de la información. Cada una de estas visiones suelen corresponder a las diferentes áreas funcionales de la empresa como, por ejemplo, producción, ventas, recursos humanos, etc.

A los esquemas conceptuales correspondientes a cada vista de usuario se les denomina *esquemas conceptuales locales*. Cada uno de estos esquemas se compone de entidades, relaciones, atributos, dominios de atributos e identificadores. El esquema conceptual también tendrá una documentación, que se irá produciendo durante su desarrollo. Las tareas a realizar en el diseño conceptual son las siguientes:

1. Identificar las entidades.
2. Identificar las relaciones.
3. Identificar los atributos y asociarlos a entidades y relaciones.
4. Determinar los dominios de los atributos.
5. Determinar los identificadores.
6. Determinar las jerarquías de generalización (si las hay).
7. Dibujar el diagrama entidad-relación.
8. Revisar el esquema conceptual local con el usuario

El modelo Entidad-Relación (E/R) se basa en una representación del mundo real en que los datos se describen como entidades, relaciones y atributos.

El principal concepto del modelo ER es la *entidad*, que es una "cosa" en el mundo real con existencia independiente. Una entidad puede ser un objeto físico (una persona, un auto, una casa o un empleado) o un objeto conceptual (una compañía, un puesto de trabajo o un curso universitario).

En nuestro ejemplo de la sección anterior podemos definir dos entidades alumnos y cursos.

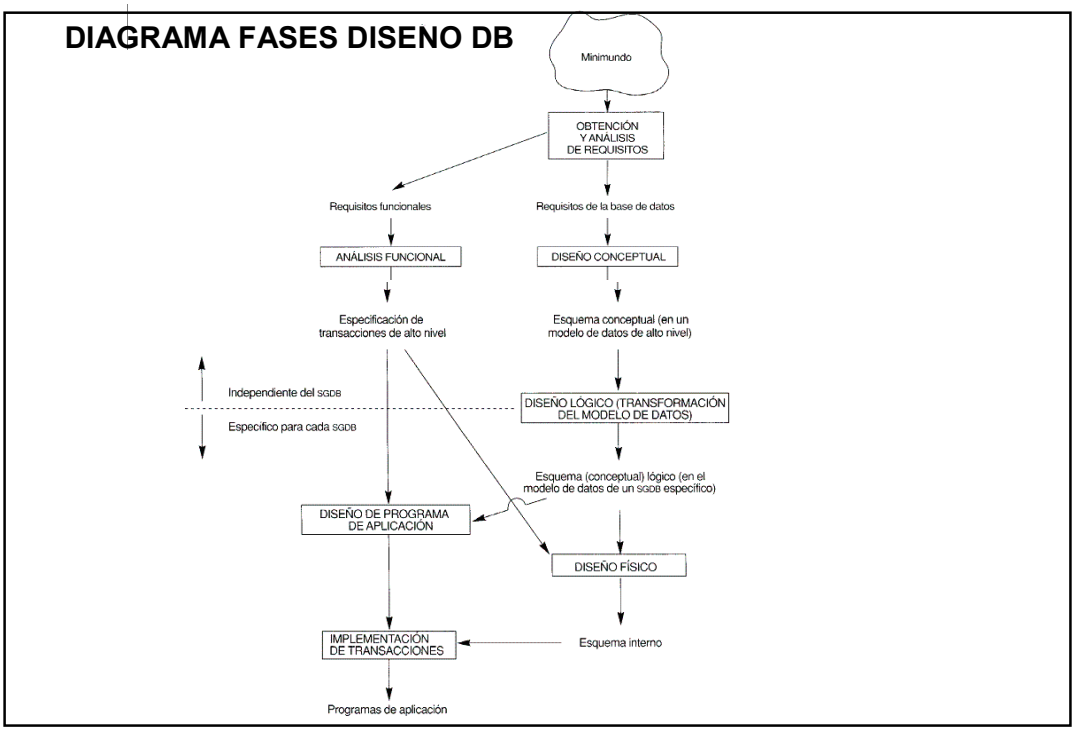
Cada entidad tiene propiedades específicas, llamadas *atributos*, que la describen. Por ejemplo, una sala de clases tiene un nombre, una ubicación, un cupo máximo, etc. En nuestro ejemplo, la entidad "alumno"

posee los atributos nombre y matrícula. Una entidad particular tiene un valor para cada uno de los atributos.

Cada uno de los atributos de una entidad posee un dominio, el que corresponde al tipo del atributo. Por ejemplo, "matrícula" tiene como dominio al conjunto de los enteros positivos y "nombre" tiene como dominio al conjunto de caracteres.

Para todo conjunto de valores de una entidad, debe existir un atributo o combinación de atributos, que identifique a cada entidad en forma única. Este atributo o combinación de atributos se denomina llave (primaria). Por ejemplo, el número de matricula es una buena llave para la entidad alumno, no así el nombre, porque pueden existir dos personas con el mismo nombre.

Una *relación* se puede definir como una asociación entre entidades. Por ejemplo, la entidad "libro" puede estar relacionada con la entidad "persona" por medio de la relación "está pedido". La entidad "alumno" puede estar relacionada con la entidad "curso" por la relación "está inscrito". Una relación también puede tener atributos. Por ejemplo, la relación "está inscrito" puede tener los atributos "semestre" y "nota de aprobación".



NOTA PARA VER BIEN EL DIAGRAMA PONER ZOOM A 150

Ejemplo:

Suponga que estamos modelando los datos de una COMPAÑIA. La base de datos COMPAÑIA debe mantener los Datos sobre los empleados de la compañía, los departamentos y los proyectos. La descripción del mini-mundo (la parte de la compañía a ser representada en la base de datos) es la siguiente:

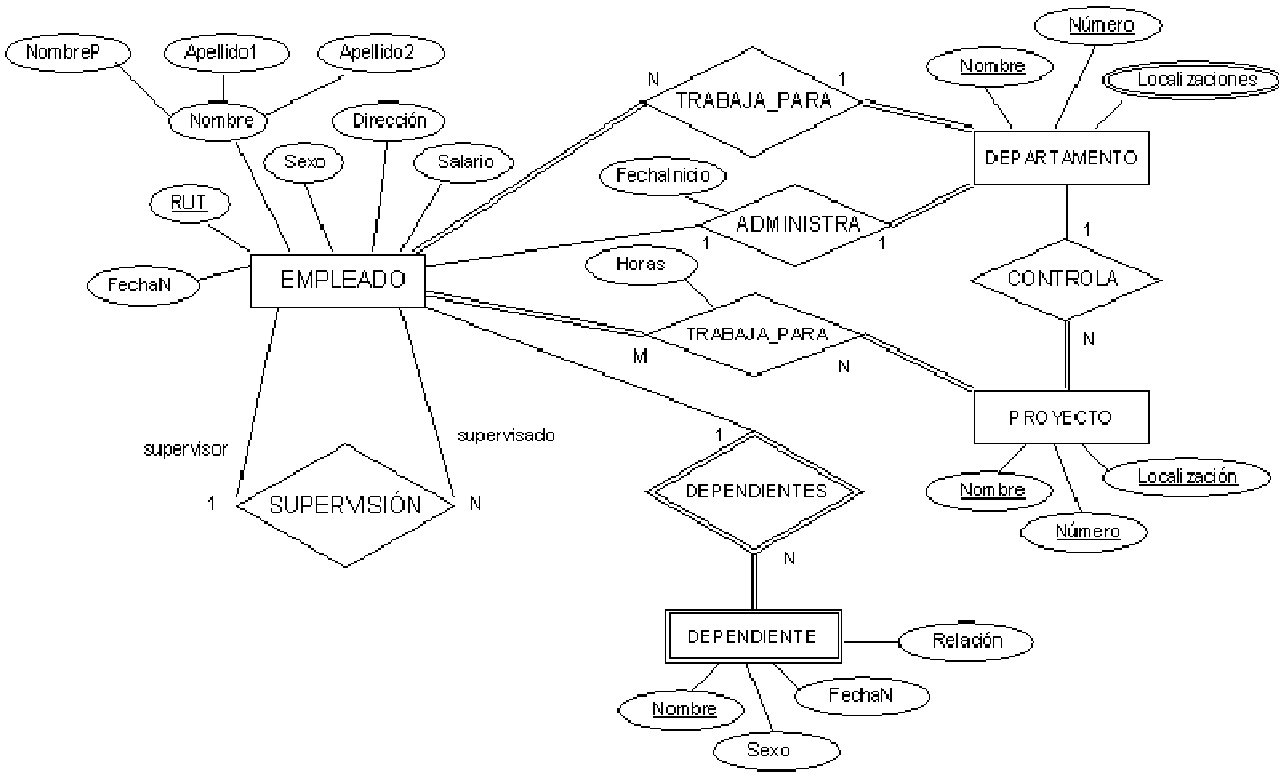
1. La compañía está organizada en departamentos. Cada departamento tiene un nombre único. Un número único, y un empleado particular quien lo administra. Se quiere saber la fecha en que el empleado administrador empezó a hacerse cargo del departamento. Un departamento puede tener varios locales.

2. Cada departamento controla un cierto número de proyectos. Cada proyecto tiene un nombre y número únicos, y un local.

3. Para cada empleado se desea tener su nombre, rut, dirección, salario, sexo y año de nacimiento. Un empleado es asignado a un departamento, pero puede trabajar en varios proyectos, los que no son necesariamente controlados por el mismo departamento. Se quiere saber el número de horas semanales que un empleado trabaja en cada proyecto. Se quiere además saber cuál es el supervisor directo de cada empleado.

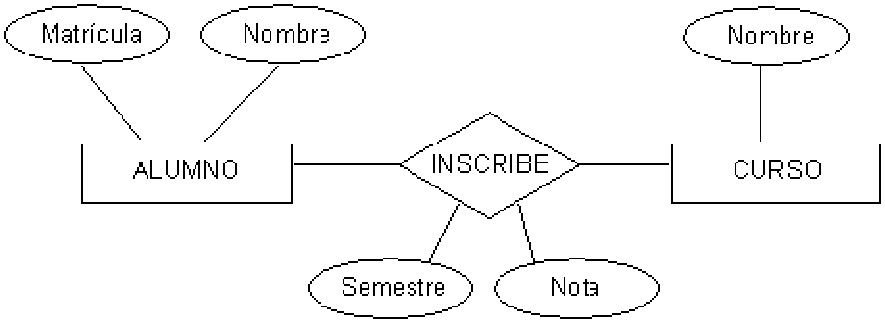
4. Se desea conocer las personas dependientes de cada empleado para propósitos de seguros. De cada dependiente se desea conocer el nombre, sexo, fecha de nacimiento y relación con el empleado.

La siguiente figura muestra el esquema de esta base de datos, a través de una notación gráfica llamada *diagrama ER*.



En este diagrama los rectángulos representan conjuntos de entidades, las elipses representan atributos y los rombos representan conjuntos de relaciones.

Usando esta notación, podemos ahora hacer el diagrama E-R del ejemplo anterior de los alumnos y los cursos matriculados.



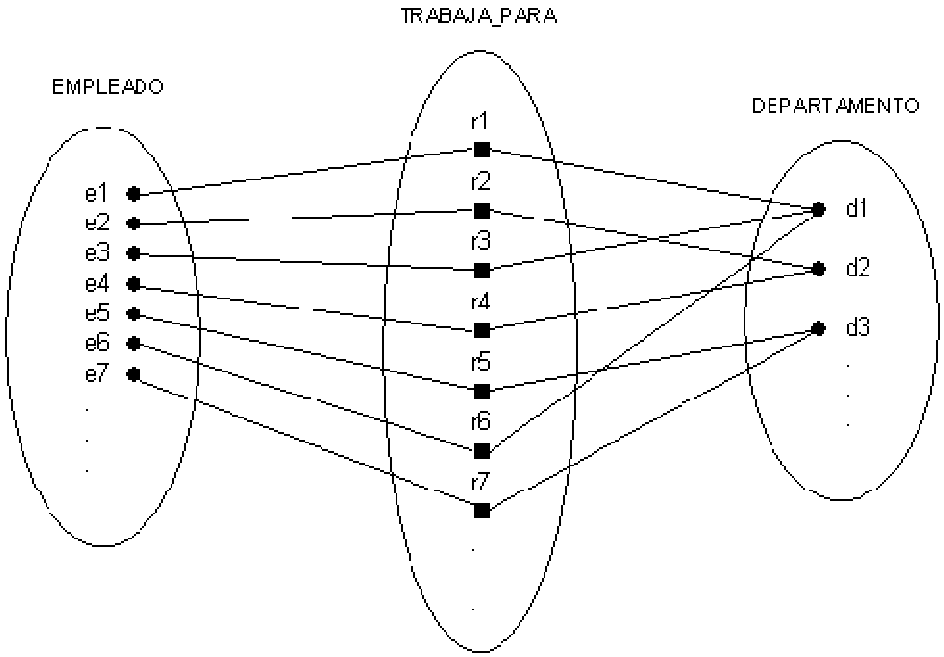
Tipos de relaciones

Un tipo de relación R entre n tipos de entidades E_1, ..., E_n define un conjunto de asociaciones entre estos tipos.

Puede ser visto como un conjunto de *instancias de la relación* r_i, donde cada r_i asocia n entidades (e_1, ..., e_n), y cada entidad e_j en r_i es un miembro del tipo de entidad E_j ($1 <= j <= n$).

Un tipo de relación es un subconjunto del producto cartesiano E_1 x E_2 x ... x.E_n.

Ejemplo. Algunas instancias de la relación TRABAJA_PARA del ejemplo anterior, podrían ser las siguientes.



Un tipo de relación podría también interpretarse como un conjunto de pares ordenados, en este caso: (e1, d1), (e2, d2), (e3, d1), (e4, d2), (e5, d3), (e6, d1), (e7, d3).

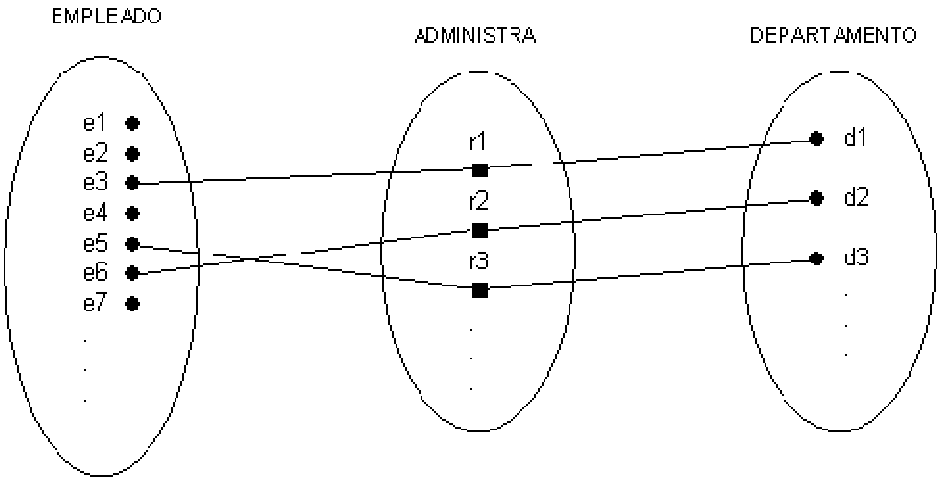
Según el número de entidades relacionadas (o *razón de cardinalidad*), se pueden definir tres tipos de relaciones:

1. Relaciones Uno a Uno (1:1). Una entidad A está asociada a lo más con una entidad B, y una entidad B a lo más con una entidad A. Ejemplo: "Ser jefe de" es una relación 1:1 entre las entidades empleado y departamento.

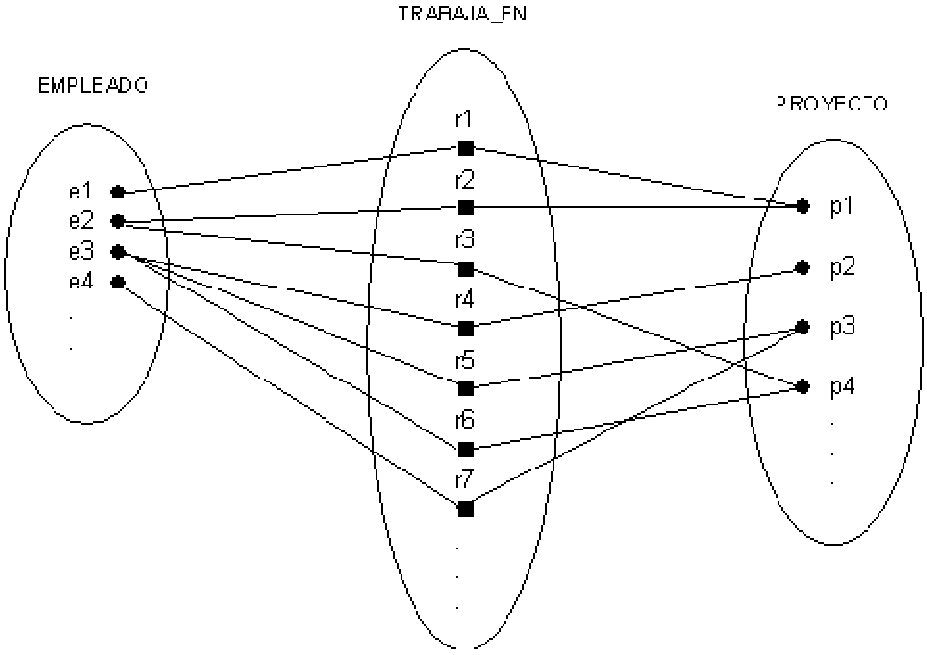
2. Relaciones Uno a Muchos (1:n). Una entidad A está asociada con una o varias entidades B. Una entidad B, sin embargo, puede estar a lo más asociada con una entidad A. Ejemplo: "Ser profesor" es una relación 1:n entre profesor y curso, suponiendo que un curso sólo lo dicta un profesor.

3. Relaciones Muchos a Muchos (n:m). Una entidad A está asociada con una o varias entidades B, y una entidad B está asociada con una o varias entidades A. Ejemplo: "Estar inscrito" es una relación n:m entre las entidades alumno y curso.

El siguiente es un ejemplo de la relación ADMINISTRA, con participación parcial de EMPLEADOS, y participación total de DEPARTAMENTOS.

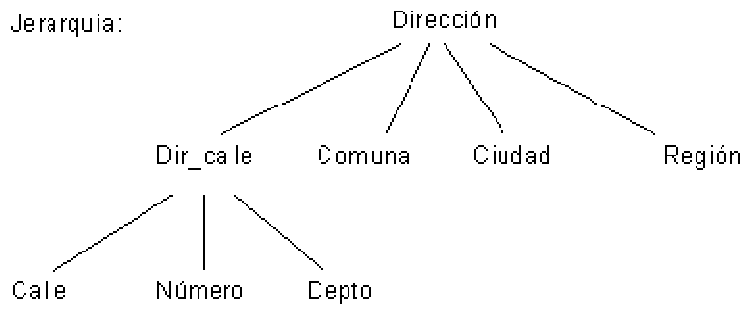


La siguiente figura muestra un ejemplo de la relación M:N TRABAJA_PARA.



Tipos de atributos

Los *atributos compuestos* se pueden dividir en sub-partes más pequeñas, que representan atributos más básicos con significados propios. Por ejemplo, una "dirección" puede sub-dividirse en: dir-calle, comuna, ciudad, región. Ejemplo:



Los atributos no sub-divididles se llaman *atómicos* o *simples*. Si no hay necesidad de referirse a los elementos individuales de una dirección, entonces la dirección completa puede considerarse un atributo simple.

Atributos de valor simple son los que tienen un sólo valor para una entidad particular.

Por ejemplo: edad.

Atributos multivalorados pueden tener un conjunto de valores para una misma entidad. Por ejemplo: "títulos profesionales" (una persona puede no tener ninguno, uno, dos o más).

En algunos casos una entidad particular puede no tener valores aplicables para un atributo. Ejemplo: "depto". Para estas situaciones tenemos un valor especial llamado *nulo*. También, si no se conoce el valor.

Un *tipo de entidad* define un conjunto de entidades con los mismos atributos.

Ejemplo:

Nombre del tipo de entidad: EMPLEADO
Atributos: Nombre, Edad, Sueldo
Conjunto de entidades: (Juan Pérez, 55, 800.000), (Federico Pardo, 40, 550.000), (Rodrigo Pozo, 25, 400.000).

En los diagramas E-R, un tipo de entidad se representa como una caja rectangular, los nombres de los atributos como elipses y las relaciones

como rombos. Los atributos multivalorados se representan con elipses dobles.

Un tipo de atributo usualmente tiene un atributo cuyos valores son distintos para cada entidad individual (atributo *clave* o *llave*) y sus valores se usan para identificar cada entidad unívocamente. Para una entidad tipo PERSONA, un atributo clave típico es el RUT. Algunas veces, varios atributos juntos forman una clave (la combinación debe ser distinta). Estos atributos clave aparecen subrayados en los diagramas.

Cada atributo simple tiene un *conjunto de valores* o *dominio* asociado, que especifica el conjunto de valores que puede asignarse a cada entidad individual. Por ejemplo, si las edades de los empleados pueden variar entre 16 y 70, entonces el dominio de Edad es {x □ N / 16 <= x <= 70}. Los dominios no se muestran en los diagramas.

Un atributo A del tipo de entidad E cuyo dominio es V, puede definirse como una función de E al conjunto potencia V (conjunto de todos los subconjuntos de V):

A: E <e> P(V)

El valor del atributo A para la entidad e es A(e). Un valor nulo se representa por el conjunto vacío.

Para un atributo compuesto A, el dominio V es el producto cartesiano de $P(V_1)$, ..., $P(V_n)$ donde V_1, ..., V_n son los dominios de los atributos simples que forman A:

$V = P(V_1) \times P(V_2) \times ... \times P(V_n)$.

Notemos que atributos compuestos y multivalorados pueden ser anidados de cualquier manera. Podemos representar anidamiento agrupando componentes de un atributo compuesto entre paréntesis ( ), separando componentes con comas, y mostrando atributos multivalor a dos entre llaves {}.

Ejemplo: Si una persona puede tener más de una dirección, y en cada una de ellas hay múltiples teléfonos, podemos especificar un atributo DirTel para una PERSONA así:

{ DirTel ({ Teléfono (CódigoArea, NumTel) }, Dirección (DirCalle (Calle, Número, NumDepto), Comuna, Ciudad, Región)) }

La persona Juan Pérez puede tener una instancia de este atributo así:

{ DirTel ({ Teléfono (2, 442-2855) }, Dirección (DirCalle (Blanco, 2120, nulo), Santiago, Santiago, RM)), DirTel ({ Teléfono (2, 241-3416) },

Dirección (DirCalle (Manuel Montt, 74, 201), Providencia, Santiago, RM)) }

Este modelo considera la Base de Datos (BD) como una colección de relaciones. De manera simple, una relación representa una *tabla*, en que cada fila representa una colección de valores que describen una entidad del mundo real. Cada fila se denomina *tupla*.

Dominios, tuplas, atributos, relaciones

Un *dominio* D es un conjunto de valores atómicos. Atómico quiere decir que cada valor en el dominio es indivisible. Es útil dar nombres a los dominios. Ejemplo:Números-telefónicos-locales: el conjunto de número de teléfono de 7 dígitos.

RUTs: números de 8 dígitos más un carácter que puede ser del 0 al 9 o K Nombres: el conjunto de nombres de personas Notas: valores entre 1.0 y 7.0

También se puede especificar un tipo de datos o formato para cada dominio. Un *esquema de relación* R, denotado $R(A_1, A_2, ..., A_n)$ está constituido por un nombre de relación R y una lista de atributos $A_1, ..., A_n$. Cada atributo A_i es el nombre de un rol jugado por el dominio D en el esquema de la relación R.

D se llama el dominio de A_i y se denota $dom(A_i)$. Un esquema relacional se usa para describir una relación. R es el nombre de esta relación. El *grado de una relación* es el número n de atributos del esquema de la relación.

Ejemplos:

ESTUDIANTE(Nombre, Rut, Teléfono, Dirección, Edad, Carrera, Prom-nota) tiene grado 7.
dom(Nombre) = Nombres
dom(Teléfono) = Números-telefónicos-locales
etc.

Def. Una *relación* o *instancia de relación* r del esquema de relación $R(A_1, A_2, ..., A_n)$, denotado también como r(R) es un conjunto de n-tuplas $r = \{t_1, t_2, ..., t_m\}$. Cada n-tupla t es una lista ordenada de n valores $t = <v_1, ..., v_n>$, donde cada valor v_i, $i <= i <= n$, es un elemento de $dom(A_i)$ o es un valor nulo.

Ejemplo:

ESTUDIANTE	Nombre	Rut	Teléfono	Dirección	Edad	Carrera	Prom-nota

	Benjamín Gonzále z	13.245.62 2-1	224-4211	Rosas 3241	19	Plan común	4.8
	Sergio Soto	12.341.22 8-5	nulo	Gay214 2	20	Ing. Ind.	5.1
	...	...	...	...	...	...	...

Cada tupla representa una entidad de estudiante en particular. La definición de relación puede replantearse así: Una relación r(R) es un subconjunto del producto cartesiano de los dominios que definen r:

r(R) <e> (dom(A_1) x dom(A_2) x ... x dom(A_n))

El número total de tuplas en el producto cartesiano es:

|dom(A_1)| * |dom(A_2)| * ... * |dom(A_n)|

Una instancia de relación refleja sólo las tuplas válidas que representa un estado particular del mundo real. A medida que el mundo real cambia, también lo hace la relación, transformándose en otro estado de relación (el esquema R es relativamente estático y no cambia excepto muy pocas veces).

Notación

Un esquema de relación R de grado n se denota R(A_1, A_2, ..., A_n)

Una n-tupla t en una relación r(R) se denota t = <v_1, ..., v_2>, donde v_i es el valor correspondiente del atributo A_i

t[A_i] se refiere al valor v_i en t para el atributo A_i

t[A_u, A_w, ..., A_z] donde A_u, A_w, ..., A_z es una lista de atributos de R, se refiere a las subtuplas de valores <v_u, v_w, ..., v_z> de t correspondientes a los atributos especificados en la lista

Las letras Q, R, S denotan nombres de relación

Las letras q, r, s denotan estados de relación

Las letras t, u, v denotan tuplas

Los nombres de atributos se califican con el nombre de relación a la cual pertenecen. Por ejemplo, ESTUDIANTE.Nombre o ESTUDIANTE.Edad

Para la tupla t = <Benjamín González, 13.245.622-1, 224-4211, Rosas 3241, 19, Plan común, 4.8>, tenemos t[Nombre] = <Benjamín González>, t[Rut, Prom-notas, Edad] = <13.245.622-1, 4.8, 19>

Restricciones

Las restricciones de dominios especifican que el valor de cada atributo A debe ser un valor atómico del dominio dom(A).

Una relación se define como un conjunto de tuplas. Por definición todos los elementos de un conjunto son distintos. Luego todas las tuplas de una relación deben ser distintas. Esto implica que dos tuplas no pueden tener la misma combinación de valores para todos sus atributos. Pero puede haber otros subconjuntos de atributos de un esquema de relación R con la propiedad de que no haya dos tuplas en una instancia de relación r de R que tengan la misma combinación de valores para esos atributos. Supongamos que denotamos tal subconjunto de atributos por SC. Entonces para cada dos tuplas distintas t1 y t2 en una instancia de relación r de R, tenemos la restricción:

t1[SC] <e> t2[SC]

Cualquier conjunto de atributos SC es denominado *super llave* del esquema de relación R. Cada relación tiene al menos una super llave (el conjunto de todos sus atributos). Una *llave* o *clave* K de un esquema de relación R es una súper llave de R con la propiedad adicional de que al sacar cualquier atributo A de K deja un conjunto de atributos K' que no es súper llave de R (una clave es una súper llave mínima).

El valor de un atributo clave se usa para identificar unívocamente una tupla en una relación. El hecho que un conjunto de atributos constituya una clase es una propiedad del esquema de la relación, y es invariante en el tiempo.

En general, un esquema de relación puede tener más de una clave, y en ese caso, cada una de las llaves es una *llave candidata*. Una de las llaves candidatas se designa como llave primaria de la relación. Usamos la convención de que los atributos que forman la llave primaria de un esquema de relación se subrayan.

Base de datos relacional

Un *esquema de Base de Datos (DB) relacional* es un conjunto de esquemas de relación S = (R_1, R_2, ..., R_m) y un conjunto RI de restricciones de integridad.

Una *instancia de DB relacional* DB de S es un conjunto de instancias de relación DB = {r_1, ..., r_n} tal que cada r_i es una instancia de R_i y tal que las relaciones r_i satisfacen las restricciones de integridad especificadas en RI.

Ejemplo:

EMPLEADO

NPILA	APPAT	APMAT	RUT	FNAC	DIRECCION	SEXO	SUELDO	RUTSUPERV	NDEPTO

DEPARTAMENTO

DNOMBRE	DNUMERO	RUTGERENTE	GERFECHAINIC

UBICACIONES_DEPTO

DNUMERO	DUBICACION

PROYECTO

PNOMBRE	PNUMERO	PUBICACION	DNUM

TRABAJA_EN

ERUT	PNO	HORAS

CARGA

ERUT	NOMBRE_CARGA	SEXO	FNAC	PARENTESCO

Los siguientes datos corresponden a una instancia de la base de datos.

EMPLEADO	NPILA	APPAT	APMAT	RUT	FNAC	DIRECCION	SEXO	SUELDO	RUTSUPERV	NDEPTO

	Juan	Pérez	García	12345678	9-1-55	Toesca 965	M	120	3334455	5
	Alicia	Zelaya	Roa	99988777	19-7-58	Blanco 2120	F	105	9876532	4
	Juana	Besa	Martínez	98765432	20-6-31	Mapocho 2540	F	240	8886655	4
	Francisco	Cea	Daza	33344555	8-12-45	Condell 221	M	310	8886655	5
	Jaime	Ramos	Salas	88866555	10-11-30	Vitacura 1015	M	360	nulo	1

DEPARTAMENTO	DNOMBRE	DNUMERO	RUTGERENTE	GERFECHAINIC
	Of. Central	1	88866555	19-6-71
	Administración	4	98765432	1-1-85
	Investigación	5	33344555	22-5-78

UBICACIONES_DEPTO	DNUMERO	DUBICACION
	1	Providencia
	4	Ñuñoa
	5	La Florida
	5	Pirque

PROYECTO	PNOMBRE	PNUMERO	PUBICACION	DNUM

	Producto X	1	La Florida	5
	Producto Y	2	Pirque	5
	Computarización	10	Ñuñoa	4
	Reorganización	20	Providencia	1

TRABAJA_EN	ERUT	PNO	HORAS
	12345678	1	32.5
	12345678	2	7.5
	33344555	2	10.0
	99988777	10	10.0
	98765432	10	10.0
	98765432	20	15.0
	88866555	20	nulo

CARGA	ERUT	NOMBRE_CARGA	SEXO	FNAC	PARENTESCO
	33344555	Alicia	F	5-4-86	Hija
	33344555	Teodoro	M	25-10-83	Hijo
	33344555	Ximena	F	3-5-54	Cónyuge
	98765432	Rodolfo	M	28-2-32	Cónyuge
	12345678	Alicia	F	5-5-57	cónyuge

Observemos que DNUMERO es el mismo para DEPARTAMENTO y PROYECTO. Pero el mismo concepto se llama DNO en EMPLEADO y DNUM en PROYECTO No hay problema.

La *restricción de integridad de entidad* establece que ningún valor de llave primaria puede ser nulo. Esto es porque ellas identifican tuplas de la relación.

La *restricción de integridad referencial* se especifica entre dos relaciones y se usa para mantener la consistencia entre tuplas de las dos relaciones. Informalmente, una tupla en una relación que hace referencia a otra relación debe referirse a una tupla existente en esa relación. Por ejemplo, NDEPTO de EMPLEADO debe coincidir con el DNUMERO de alguna tupla de la relación DEPARTAMENTO. Para una definición formal, necesitamos el concepto de llave foránea.

Def. Un conjunto de atributos LF en el esquema de relación R_1 es una *llave foránea* de R_1 si satisface las siguientes reglas:

1. Los atributos en LF tienen el mismo dominio que los atributos de la llave primaria LP de otro esquema de relación R_2. Los atributos LF se dicequ*ereferencian*larelaciónR_2.

2. Un valor de LF en una tulpa t_1 de R_1 ya sea es nulo, o ocurre como un valor de LP para alguna tupla t_2 de R_2.

Ejemplo: NDEPTO en una tupla t_1 de EMPLEADO debe coincidir con el valor de una llave primaria DNUMERO en alguna tupla t_2 de la relación DEPARTAMENTO, o el valor de DNO puede ser nulo si el empleado no pertenece a un departamento.

Las restricciones anteriores no consideran las *restricciones semánticas* que quizás puedan especificarse y sostenerse en una BD relacional. Por ejemplo, "el sueldo de un empleado no puede exceder el de su supervisor", "el número máximo de horas que puede trabajar un empleado en todos los proyectos es 56".

Operaciones de actualización

La operación *Insert* provee una lista de valores de atributos para una nueva tupla t que se va a insertar en una relación R. Ejemplo:

Insert <"Cecilia", "Rodríguez", "Kolonsky", "67767898", "5-4-50", "Ejército 565", "F", 100, nulo, "4"> en EMPLEADO. Esta inserción satisface todas las restricciones, así que es aceptable.

Insert <"Alicia", "Zelaya", "Roa", "99988777", "15-3-50", "Gay 1315", "F", 120, "98765432", "4"> en EMPLEADO. Viola la restricción de clave porque otra tupla con el mismo Rut ya existe en la relación. Inaceptable.

Insert <"Cecilia", "Rodríguez", "Kolonsky", nulo, "5-4-50", "Ejército 565", "F", 100, nulo, 4> en EMPLEADO. Viola la restricción de integridad (nulo para clave primaria Rut). Inaceptable.

Insert <"Cecilia", "Rodríguez", "Kolonsky", "67767898", "5-4-50", "Ejército

565", "F", 100, "98765432", 7> en EMPLEADO. Viola la restricción de integridad referencial porque no existe una tupla en DEPARTAMENTO con DNUMERO = 7.

El DBMS puede hacer dos opciones cuando hay violación de restricciones. Una es rechazar la inserción. La otra es intentar corregir la razón de rechazo de la inserción.

Operaciones de borrado

La operación *Delete* borra tuplas de una relación. Es posible que se viole la *integridad referencial* cuando una tupla que se quiere borrar es referenciada por las llaves foráneas de otras tuplas de la BD. Las tuplas a borrar se especifican a través de condiciones sobre los atributos de la relación. Ejemplos:

Delete tupla con ERUT = "99988777" AND PNO = 10 en la relación TRABAJA_EN. Esta operación es aceptable.

Delete tupla con RUT = "99988777" en la relación EMPLEADO. Inaceptable porque dos tuplas en TRABAJA_EN se refieren a esta tupla. Si se borra, hay violaciones a la integridad referencial.

Hay tres opciones con respecto a una operación de borrado que causa una violación. La primera es *rechazar* la operación. La segunda es intentar *propagar* el borrado. Una tercera opción es *modificar* los valores de los atributos referenciantes que causan la violación (cada uno de estos valores puede ser puesto en nulo o cambiado para referenciar otra tupla válida). Hay que observar que si un atributo referenciante que causa una violación, es parte de la llave primaria, no puede ser nulo, pues se violaría la integridad de entidad.

Operaciones de modificación

La operación *Modify* se usa para cambiar valores a uno o más atributos en una tupla (o tuplas) de una relación R. Es necesario especificar una condición sobre los atributos de R para seleccionar la o las tuplas a modificar. Ejemplos:

Modify SUELDO de la tupla de EMPLEADO con RUT = "99988777" a 135.

Aceptable.

Modify NDEPTO de la tupla de EMPLEADO con RUT = "99988777" a 1. Aceptable.

Modify NDEPTO de la tupla de EMPLEADO con RUT = "99988777" a 7. Inaceptable pues viola la integridad referencial.

Modify RUT de la tupla de EMPLEADO con RUT = "99988777" a "98765432". Inaceptable pues viola restricciones de clave primaria e integridad-referencial.

El modificar un atributo que no es llave primaria ni llave foránea no tiene problemas. El modificar una llave primaria es similar a borrar una tupla e insertar otra en su lugar. Por tanto, es relevante la discusión anterior (Insert y Delete). Si se modifica un atributo de una llave foránea, el Sistema Administrador de Base de Datos (DBMS) debe asegurarse que el nuevo valor se refiere a una tupla existente en la relación referenciada.

Terminología relacional equivalente

Código	Nombre	Posición	Salario
1	Edgardo Trujillo	Gerente	19000
2	Lidimarie Fonsi	Empleada	12000
3	Jean Piaget	Empleado	13500
4	Jerome Bruner	Empleado	14000

Trabajo (Código, Nombre, Posición, Salario), donde Código es la Clave Primaria

- Relación = tabla o archivo
- Tupla = registro, fila o renglón
- Atributo = columna o campo
- Clave = llave o código de identificación
- Clave Candidata = superclave mínima
- Clave Primaria = clave candidata elegida
- Clave Ajena = clave externa o clave foránea
- Clave Alternativa = clave secundaria
- Dependencia Multivaluada = dependencia multivalor
- RDBMS = Del inglés *Relational Data Base Manager System* que significa, *Sistema Gestor de Bases de Datos Relacionales*.

- 1FN = Significa, *Primera Forma Normal* o 1NF del inglés *First Normal Form*.

Los términos Relación, Tupla y Atributo derivan de las matemáticas relacionales, que constituyen la fuente teórica del modelo de base de datos relacional.

Todo atributo en una tabla tiene un dominio, el cual representa el conjunto de valores que el mismo puede tomar. Una instancia de una tabla puede verse entonces como un subconjunto del producto cartesiano entre los dominios de los atributos. Sin embargo, suele haber algunas diferencias con la analogía matemática, dado que algunos RDBMS permiten filas duplicadas, entre otras cosas. Finalmente, una tupla puede razonarse matemáticamente como un elemento del producto cartesiano entre los dominios.

4.2 Solución de problemas

Redundancia e inconsistencia de datos.

Puesto que los archivos que mantienen almacenada la información son creados por diferentes tipos de programas de aplicación existe la posibilidad de que si no se controla detalladamente el almacenamiento, se pueda originar un duplicado de información, es decir que la misma información sea más de una vez en un dispositivo de almacenamiento. Esto aumenta los costos de almacenamiento y acceso a los datos, además de que puede originar la inconsistencia de los datos - es decir diversas copias de un mismo dato no concuerdan entre si -, por ejemplo: que se actualiza la dirección de un cliente en un archivo y que en otros archivos permanezca la anterior.

Dificultad para tener acceso a los datos.

Un sistema de base de datos debe contemplar un entorno de datos que le facilite al usuario el manejo de los mismos. Supóngase un banco, y que uno de los gerentes necesita averiguar los nombres de todos los clientes que viven dentro del código postal 78733 de la ciudad. El gerente pide al departamento de procesamiento de datos que genere la lista correspondiente. Puesto que esta situación no fue prevista en el diseño del sistema, no existe ninguna aplicación de consulta que permita este tipo de solicitud, esto ocasiona una deficiencia del sistema.

Aislamiento de los datos.

Puesto que los datos están repartidos en varios archivos, y estos no pueden tener diferentes formatos, es difícil escribir nuevos programas de aplicación para obtener los datos apropiados.

Anomalías del acceso concurrente.

Para mejorar el funcionamiento global del sistema y obtener un tiempo de respuesta más rápido, muchos sistemas permiten que múltiples usuarios actualicen los datos simultáneamente. En un entorno así la interacción de actualizaciones concurrentes puede dar por resultado datos inconsistentes. Para prevenir esta posibilidad debe mantenerse alguna forma de supervisión en el sistema.

Problemas de seguridad.

La información de toda empresa es importante, aunque unos datos lo son más que otros, por tal motivo se debe considerar el control de acceso a los mismos, no todos los usuarios pueden visualizar alguna información, por tal motivo para que un sistema de base de datos sea confiable debe mantener un grado de seguridad que garantice la autentificación y protección de los datos. En un banco por ejemplo, el personal de nóminas sólo necesita ver la parte de la base de datos que tiene información acerca de los distintos empleados del banco y no a otro tipo de información.

Problemas de integridad.

Los valores de datos almacenados en la base de datos deben satisfacer cierto tipo de restricciones de consistencia. Estas restricciones se hacen cumplir en el sistema añadiendo códigos apropiados en los diversos programas de aplicación.

Lo anterior es originado por:

- Redundancia
- Anomalías
  - Actualización
  - Inserción
  - Borrado

Creadas durante:

- Mantenimiento
- Creación
- Modificación

Donde la Solución es:

Normalización

4.3 Normalización: 1NF, 2NF, 3FN, BCNF

Si nunca antes hemos oído hablar de la "normalización de datos", no debemos temer. Mientras que la normalización puede parecer un tema complicado, nos podemos beneficiar ampliamente al entender los conceptos más elementales de la normalización.

Una de las formas más fáciles de entender esto es pensar en nuestras tablas como hojas de cálculo. Por ejemplo, si quisiéramos seguir la pista de nuestra colección de CD's en una hoja de cálculo, podríamos diseñar algo parecido a lo que se muestra en la siguiente tabla.

+------------+-------------+--------------+ .. +--------------+

| Álbum | track1 | track2 | | track10 |

+------------+-------------+--------------+ .. +--------------+

Esto parece razonable. Sin embargo el problema es que el número de pistas que tiene un CD varía bastante. Esto significa que con este método tendríamos que tener una hoja de cálculo realmente grande para albergar todos los datos, que en los peores casos podrían ser de hasta 20 pistas. Esto en definitiva no es nada bueno.

Uno de los objetivos de una estructura de tabla normalizada es minimizar el número de "celdas vacías". El darnos cuenta de que cada lista de CD's tiene un conjunto fijo de campos (título, artista, año, género) y un conjunto variable de atributos (el número de pistas) nos da una idea de cómo dividir los datos en múltiples tablas que luego podamos relacionar entre sí.

Mucha gente no esta familiarizada con el concepto "relacional", de manera sencilla esto significa, que grupos parecidos de información son almacenados en distintas tablas que luego pueden ser "juntadas" (relacionadas) basándose en los datos que tengan en común.

Es necesario que al realizar la estructura de una base de datos, esta sea flexible. La flexibilidad está en el hecho que podemos agregar datos al sistema posteriormente sin tener que rescribir lo que ya tenemos. Por ejemplo, si quisiéramos agregar la información de los artistas de cada álbum, lo único que tenemos que hacer es crear una tabla artista que

esté relacionada a la tabla álbum de la misma manera que la tabla pista. Por lo tanto, no tendremos que modificar la estructura de nuestras tablas actuales, simplemente agregar la que hace falta.

La eficiencia se refiere al hecho de que no tenemos duplicación de datos, y tampoco tenemos grandes cantidades de "celdas vacías".

El objetivo principal del diseño de bases de datos es generar tablas que modelan los registros en los que guardaremos nuestra información.

Es importante que esta información se almacene sin redundancia para que se pueda tener una recuperación rápida y eficiente de los datos.

A través de la normalización tratamos de evitar ciertos defectos que nos conduzcan a un mal diseño y que lleven a un procesamiento menos eficaz de los datos.

Podríamos decir que estos son los principales objetivos de la normalización:

- Controlar la redundancia de la información.
- Evitar pérdidas de información.
- Capacidad para representar toda la información.
- Mantener la consistencia de los datos.

La normalización es una técnica para diseñar la estructura lógica de los datos de un sistema de información en el modelo relacional, desarrollada por E. F. Codd en 1972. Es una estrategia de diseño de abajo a arriba: se parte de los atributos y éstos se van agrupando en relaciones (tablas) según su afinidad. Aquí no se utilizará la normalización como una técnica de diseño de bases de datos, sino como una etapa posterior a la correspondencia entre el esquema conceptual y el esquema lógico, que elimine las dependencias entre atributos no deseadas. Las ventajas de la normalización son las siguientes:

- Evita anomalías en inserciones, modificaciones y borrados.
- Mejora la independencia de datos.
- No establece restricciones artificiales en la estructura de los datos.

Uno de los conceptos fundamentales en la normalización es el de *dependencia funcional*. Una dependencia funcional es una relación entre atributos de una misma relación (tabla). Si x e y son atributos de la relación R, se dice que y es funcionalmente dependiente de x (se denota por $x \longrightarrow y$) si cada valor de x tiene asociado un solo valor de y (x e y

pueden constar de uno o varios atributos). A x se le denomina *determinante*, ya que x determina el valor de y. Se dice que el atributo y es *completamente dependiente* de x si depende funcionalmente de x y no depende de ningún subconjunto de x.

La dependencia funcional es una noción semántica. Si hay o no dependencias funcionales entre atributos no lo determina una serie abstracta de reglas, sino, más bien, los modelos mentales del usuario y las reglas de negocio de la organización o empresa para la que se desarrolla el sistema de información. Cada dependencia funcional es una clase especial de regla de integridad y representa una relación de uno a muchos.

En el proceso de normalización se debe ir comprobando que cada relación (tabla) cumple una serie de reglas que se basan en la clave primaria y las dependencias funcionales. Cada regla que se cumple aumenta el grado de normalización. Si una regla no se cumple, la relación se debe descomponer en varias relaciones que sí la cumplan.

La normalización se lleva a cabo en una serie pasos. Cada paso corresponde a una forma normal que tiene unas propiedades. Conforme se va avanzando en la normalización, las relaciones tienen un formato más estricto (más fuerte) y, por lo tanto, son menos vulnerables a las anomalías de actualización. El modelo relacional sólo requiere un conjunto de relaciones en primera forma normal. Las restantes formas normales son opcionales. Sin embargo, para evitar las anomalías de actualización, es recomendable llegar al menos a la tercera forma normal.

Uno de los retos en el diseño de la base de datos es el de obtener una estructura estable y lógica tal que:

1. El sistema de base de datos no sufra de anomalías de almacenamiento.
2. El modelo lógico pueda modificarse fácilmente para admitir nuevos requerimientos.

Una base de datos implantada sobre un modelo bien diseñado tiene mayor esperanza de vida aun en un ambiente dinámico, que una base de datos con un diseño pobre. En promedio, una base de datos experimenta una reorganización general cada seis años, dependiendo de lo dinámico de los requerimientos de los usuarios. Una base de datos bien diseñada tendrá un buen desempeño aunque aumente su tamaño, y será lo suficientemente flexible para incorporar nuevos requerimientos o características adicionales.

Existen diversos riesgos en el diseño de las bases de datos relacionales que afecten la funcionalidad de la misma, los riesgos generalmente son la redundancia de información y la inconsistencia de datos.

La normalización es el proceso de simplificar la relación entre los campos de un registro.

Por medio de la normalización un conjunto de datos en un registro se reemplaza por varios registros que son más simples y predecibles y, por lo tanto, más manejables. La normalización se lleva a cabo por cuatro razones:

- Estructurar los datos de forma que se puedan representar las relaciones pertinentes entre los datos.
- Permitir la recuperación sencilla de los datos en respuesta a las solicitudes de consultas y reportes.
- Simplificar el mantenimiento de los datos actualizándolos, insertándolos y borrándolos.
- Reducir la necesidad de reestructurar o reorganizar los datos cuando surjan nuevas aplicaciones.

En términos más sencillos la normalización trata de simplificar el diseño de una base de datos, esto a través de la búsqueda de la mejor estructuración que pueda utilizarse con las entidades involucradas en ella.

Pasos de la normalización:

1. Descomponer todos los grupos de datos en registros bidimensionales.
2. Eliminar todas las relaciones en la que los datos no dependan completamente de la llave primaria del registro**.**
3. Eliminar todas las relaciones que contengan dependencias transitivas.

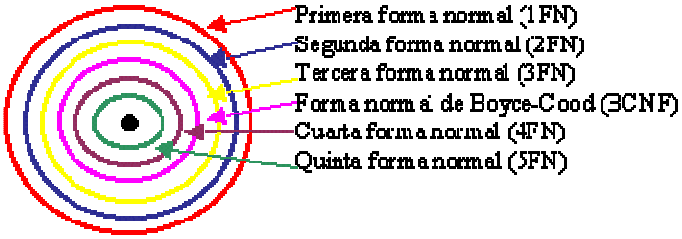
La teoría de normalización tiene como fundamento el concepto de formas normales; se dice que una relación está en una determinada forma normal si satisface un conjunto de restricciones.

Primera y segunda formas normales.

Formas normales.

Son las técnicas para prevenir las anomalías en las tablas. Dependiendo de su estructura, una tabla puede estar en primera forma normal, segunda forma normal o en cualquier otra.

Relación entre las formas normales:



Primera forma normal.

Definición formal:

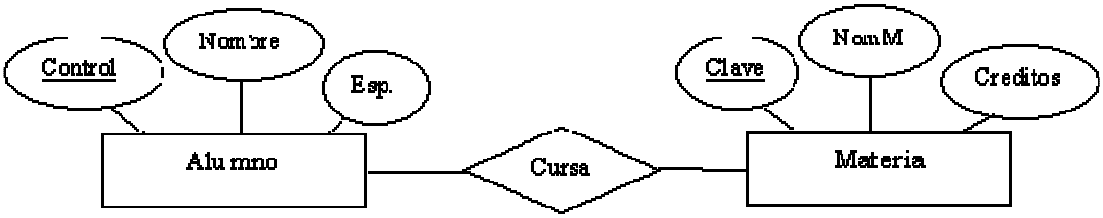
Una relación R se encuentra en 1FN si y solo sí por cada renglón columna contiene valores atómicos.

Abreviada como 1FN, se considera que una relación se encuentra en la primera forma normal cuando cumple lo siguiente:

1. Las celdas de las tablas poseen valores simples y no se permiten grupos ni arreglos repetidos como valores, es decir, contienen un solo valor por cada celda.
2. Todos los ingresos en cualquier columna (atributo) deben ser del mismo tipo.
3. Cada columna debe tener un nombre único, el orden de las columnas en la tabla no es importante.
4. Dos filas o renglones de una misma tabla no deben ser idénticas, aunque el orden de las filas no es importante.

Por lo general la mayoría de las relaciones cumplen con estas características, así que podemos decir que la mayoría de las relaciones se encuentran en la primera forma normal.

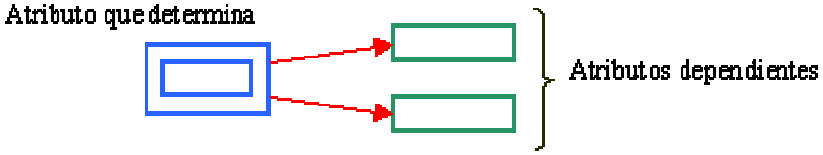
Para ejemplificar como se representan gráficamente las relaciones en primera forma normal consideremos la relación alumno cursa materia cuyo diagrama E-R es el siguiente:



Como esta relación maneja valores atómicos, es decir un solo valor por cada uno de los campos que conforman a los atributos de las entidades, ya se encuentra en primera forma normal, gráficamente así representamos a las relaciones en 1FN.

Segunda forma normal.

Para definir formalmente la segunda forma normal requerimos saber que es una **dependencia funcional:** Consiste en edificar que atributos dependen de otro(s) atributo(s).

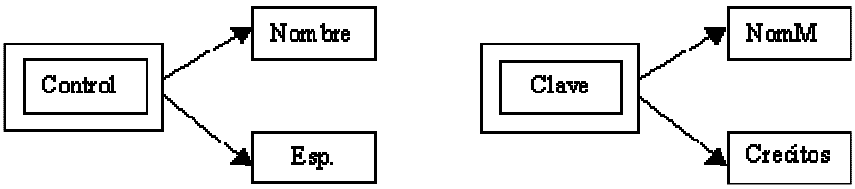


Definición formal:

Una relación R está en 2FN si y solo si está en 1FN y los atributos no primos dependen funcionalmente de la llave primaria.

Una relación se encuentra en segunda forma normal, cuando cumple con las reglas de la primera forma normal y todos sus atributos que no son claves (llaves) dependen por completo de la clave. De acuerdo con está definición, cada tabla que tiene un atributo único como clave, esta en segunda forma normal.

La segunda forma normal se representa por dependencias funcionales como:



Nótese que las llaves primarias están representadas con doble cuadro, las flechas nos indican que de estos atributos se puede referenciar a los otros atributos que dependen funcionalmente de la llave primaria.

Tercera forma normal y la forma normal de Boyce Codd.

Para definir formalmente la 3FN necesitamos definir **dependencia transitiva:** En una afinidad (tabla bidimensional) que tiene por lo menos 3 atributos (A,B,C) en donde A determina a B**,** B determina a C pero no determina a A.

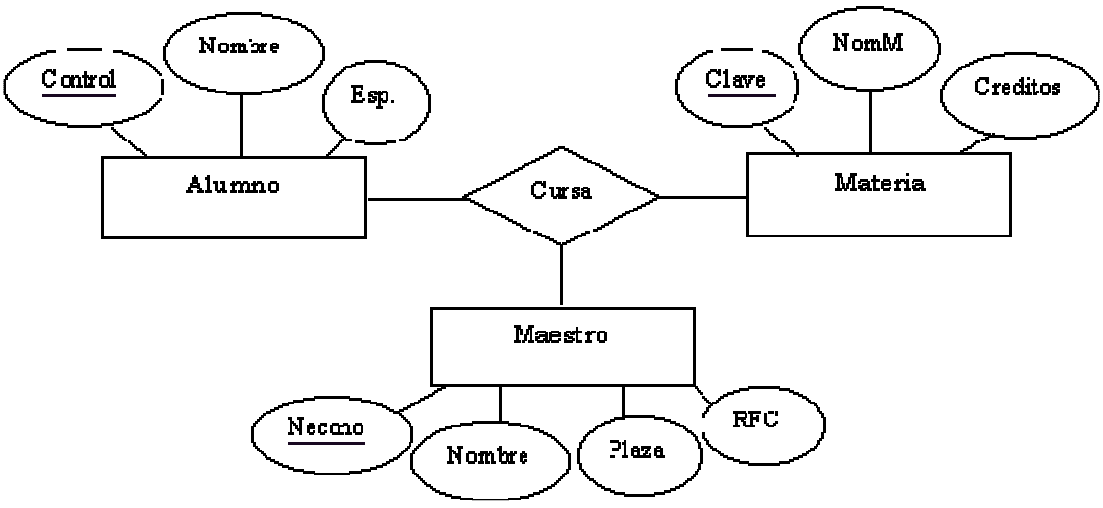
Tercera forma normal.

Definición formal:

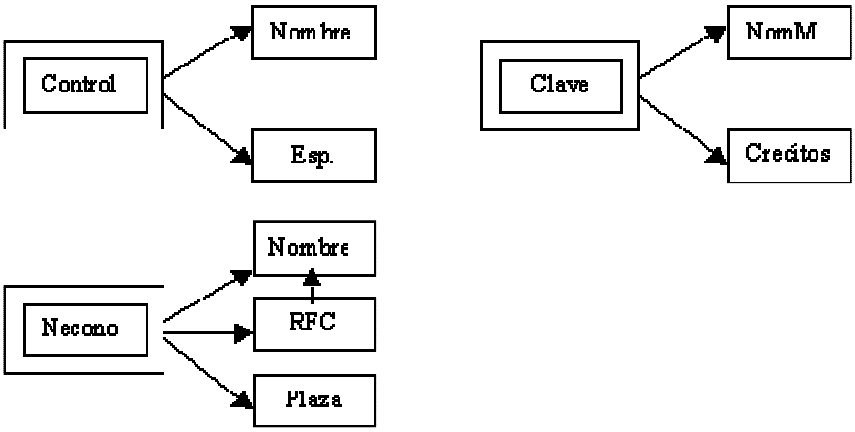
Una relación R está en 3FN si y solo si esta en 2FN y todos sus atributos no primos dependen no transitivamente de la llave primaria.

Consiste en eliminar la dependencia transitiva que queda en una segunda forma normal, en pocas palabras una relación esta en tercera forma normal si está en segunda forma normal y no existen dependencias transitivas entre los atributos, nos referimos a dependencias transitivas cuando existe más de una forma de llegar a referencias a un atributo de una relación.

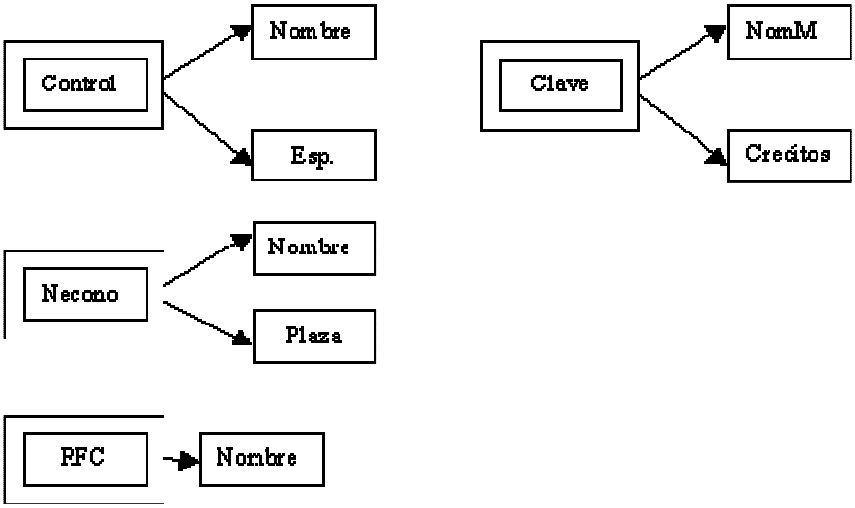
Por ejemplo, consideremos el siguiente caso:



Tenemos la relación alumno-cursa-materia manejada anteriormente, pero ahora consideramos al elemento maestro, gráficamente lo podemos representar de la siguiente manera:



Podemos darnos cuenta que se encuentra graficado en segunda forma normal, es decir que todos los atributos llave están indicados en doble cuadro indicando los atributos que dependen de dichas llaves, sin embargo en la llave Necono tiene como dependientes a 3 atributos en el cual el nombre puede ser referenciado por dos atributos: Necono y RFC (Existe dependencia transitiva), podemos solucionar esto aplicando la tercera forma normal que consiste en eliminar estas dependencias separando los atributos, entonces tenemos:



Forma normal de Boyce Codd.

Determinante: Uno o más atributos que, de manera funcional, determinan otro atributo o atributos. En la dependencia funcional (A,B)-->C, (A,B) son los determinantes.

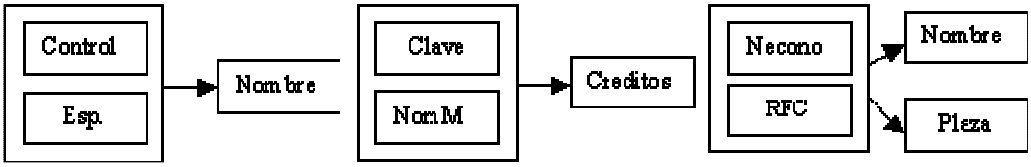
Definición formal:

Una relación R esta en FNBC si y solo si cada determinante es una llave candidato.

Denominada por sus siglas en ingles como BCNF; Una tabla se considera en esta forma si y sólo sí cada determinante o atributo es una llave candidato.

Continuando con el ejemplo anterior, si consideramos que en la entidad alumno sus atributos control y nombre nos puede hacer referencia al atributo esp., entonces decimos que dichos atributos pueden ser llaves candidato.

Gráficamente podemos representar la forma normal de Boyce Codd de la siguiente forma:



Obsérvese que a diferencia de la tercera forma normal, agrupamos todas las llaves candidato para formar una global (representadas en el recuadro) las cuales hacen referencia a los atributos que no son llaves candidato.

4.4 Criterios para normalizar

Codd se percató de que existían bases de datos en el mercado las cuales decían ser relacionales, pero lo único que hacían era guardar la información en las tablas, sin estar estas tablas literalmente normalizadas; entonces éste publicó 12 reglas que un verdadero sistema relacional debería tener, en la práctica algunas de ellas son difíciles de realizar. Un sistema podrá considerarse "más relacional" cuanto más siga estas reglas.

Regla No. 1 - La Regla de la información

Toda la información en un RDBMS está explícitamente representada de una sola manera por valores en una tabla.

Cualquier cosa que no exista en una tabla no existe del todo. Toda la información, incluyendo nombres de tablas, nombres de vistas, nombres de columnas, y los datos de las columnas deben estar almacenados en tablas dentro de las bases de datos. Las tablas que contienen tal información constituyen el Diccionario de Datos. Esto significa que todo tiene que estar almacenado en las tablas.

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. Y puede usarse el mismo lenguaje (ej. SQL) para acceder a los datos y a los metadatos (regla 4)

Regla No. 2 - La regla del acceso garantizado

Cada ítem de datos debe ser lógicamente accesible al ejecutar una búsqueda que combine el nombre de la tabla, su clave primaria, y el nombre de la columna.

Esto significa que dado un nombre de tabla, dado el valor de la clave primaria, y dado el nombre de la columna requerida, deberá encontrarse uno y solamente un valor. Por esta razón la definición de claves primarias para todas las tablas es prácticamente obligatoria.

Regla No. 3 - Tratamiento sistemático de los valores nulos

La información inaplicable o faltante puede ser representada a través de valores nulos.

Un RDBMS (Sistema Gestor de Bases de Datos Relacionales) debe ser capaz de soportar el uso de valores nulos en el lugar de columnas cuyos valores sean desconocidos o inaplicables.

Regla No. 4 - La regla de la descripción de la base de datos

La descripción de la base de datos es almacenada de la misma manera que los datos ordinarios, esto es, en tablas y columnas, y debe ser accesible a los usuarios autorizados.

La información de tablas, vistas, permisos de acceso de usuarios autorizados, etc, debe ser almacenada exactamente de la misma manera: En tablas. Estas tablas deben ser accesibles igual que todas las tablas, a través de sentencias de SQL (o similar).

Regla No. 5 - La regla del sub-lenguaje Integral

Debe haber al menos un lenguaje que sea integral para soportar la definición de datos, manipulación de datos, definición de vistas, restricciones de integridad, y control de autorizaciones y transacciones.

Esto significa que debe haber por lo menos un lenguaje con una sintaxis bien definida que pueda ser usado para administrar completamente la base de datos.

Regla No. 6 - La regla de la actualización de vistas

Todas las vistas que son teóricamente actualizables, deben ser actualizables por el sistema mismo.

La mayoría de las RDBMS permiten actualizar vistas simples, pero deshabilitan los intentos de actualizar vistas complejas.

Regla No. 7 - La regla de insertar y actualizar

La capacidad de manejar una base de datos con operandos simples aplica no sólo para la recuperación o consulta de datos, sino también para la inserción, actualización y borrado de datos'.

Esto significa que las cláusulas para leer, escribir, eliminar y agregar registros (SELECT, UPDATE, DELETE e INSERT en SQL) deben estar disponibles y operables, independientemente del tipo de relaciones y restricciones que haya entre las tablas.

Regla No. 8 - La regla de independencia física

El acceso de usuarios a la base de datos a través de terminales o programas de aplicación, debe permanecer consistente lógicamente cuando quiera que haya cambios en los datos almacenados, o sean cambiados los métodos de acceso a los datos.

El comportamiento de los programas de aplicación y de la actividad de usuarios vía terminales debería ser predecible basados en la definición lógica de la base de datos, y éste comportamiento debería permanecer inalterado, independientemente de los cambios en la definición física de ésta.

Regla No. 9 - La regla de independencia lógica

Los programas de aplicación y las actividades de acceso por terminal deben permanecer lógicamente inalteradas cuando quiera que se hagan cambios (según los permisos asignados) en las tablas de la base de datos.

La independencia lógica de los datos especifica que los programas de aplicación y las actividades de terminal deben ser independientes de la estructura lógica, por lo tanto los cambios en la estructura lógica no deben alterar o modificar estos programas de aplicación.

Regla No. 10 - La regla de la independencia de la integridad

Todas las restricciones de integridad deben ser definibles en los datos, y almacenables en el catalogo, no en el programa de aplicación.

Las reglas de integridad

1. Ningún componente de una clave primaria puede tener valores en blanco o nulos. (esta es la norma básica de integridad).

2. Para cada valor de clave foránea deberá existir un valor de clave primaria concordante. La combinación de estas reglas aseguran que haya Integridad referencial.

Regla No. 11 - La regla de la distribución

El sistema debe poseer un lenguaje de datos que pueda soportar que la base de datos esté distribuida físicamente en distintos lugares sin que esto afecte o altere a los programas de aplicación.

El soporte para bases de datos distribuidas significa que una colección arbitraria de relaciones, bases de datos corriendo en una mezcla de distintas máquinas y distintos sistemas operativos y que esté conectada por una variedad de redes, pueda funcionar como si estuviera disponible como en una única base de datos en una sola máquina.

Regla No. 12 - Regla de la no-subversión

Si el sistema tiene lenguajes de bajo nivel, estos lenguajes de ninguna manera pueden ser usados para violar la integridad de las reglas y restricciones expresadas en un lenguaje de alto nivel (como SQL).

Algunos productos solamente construyen una interfaz relacional para sus bases de datos No relacionales, lo que hace posible la subversión (violación) de las restricciones de integridad. Esto no debe ser permitido.