Base de datos 1

Alumno: Santiago Vietto

Docente: Juan José Vulcano

Institución: UCC

Año: 2021

Introducción

Sistemas de gestión de bases de datos (SGBD)

_ Vamos a estar trabajando en lo que se denominan sistemas gestores de bases de datos. Un gestor de base de datos es un volumen de información, un contenedor de información, que habla sobre una empresa o problema en particular. Los gestores de bases de datos, el gestor, puede administrar muchísimas bases de datos pero cuando nosotros definamos y entremos en el contexto de una base de datos, esa base trata sobre un problema en particular, en donde por ejemplo el problema o la empresa podría ser la UCC y dentro de ese gestor de base de datos guardaríamos información relacionada con los alumnos, profesores, materias, notas, regularidades, etc, o sea cuando definimos sobre el gestor de base de datos un problema o empresa en particular, todo ese contenedor de datos debería administrar, manejar y conocer todas las reglas de la información relacionada a ese problema que estamos tratando, ahora, si la empresa por ejemplo cambia de la UCC a la clínica Reina Fabiola nuestro problema en ese momento cambia y ahora el gestor de bases de datos debería conocer sobre los médicos que atienden en la clínica, pacientes, obras sociales, estudios, etc. Un sistema gestor de base de datos, en un momento, va a almacenar toda la problemática relacionada a una empresa en particular. Dentro del contenedor vamos a encontrar:

- Una colección de datos que están relacionadas, o sea voy a saber que vinculación tienen por ejemplo los alumnos con los profesores, o materias, etc.
- El conjunto de programas de acceso a los datos, por ejemplo que posibilidad tiene un alumno de ver una calificación, de modificar la nota, que en este caso los alumnos no podrían modificar las notas pero si los docentes y secretarios. Los programas que dan acceso a los datos son parte de la información que administra el gestor de base de datos.
- Y además el entorno que resulta conveniente y eficiente en su uso, o sea quien va a usar o no los datos que están dentro de este entorno.

Objetivo: el objetivo de un SGBD es gestionar grandes volúmenes o cantidades de información. Debemos saber que lo que hace un gestor de base de datos es almacenar y administrar grandes volúmenes de información. Lo que se enumera a continuación es parte de lo que hace un gestor de base de datos:

- Tiene que tener capacidad de definir las estructuras de datos para almacenar y administrar esa información, para que no se pierda o se repita.
- Proveer mecanismos para su manipulación.
- Garantizar que la información almacenada sea confiable, o la fiabilidad de la información almacenada.
- Soportar las caídas del sistema.

- Controlar intentos de accesos no autorizados, dar permisos a que la gente toque o no algunos datos.
- Administrar el uso del SGBD compartido entre usuarios evitando resultados anómalos.

_ La diferencia entre base de datos y gestor de base de datos es que el gestor es la herramienta que va a guardar, administrar o manipular la información de la base de datos, y la base de datos es el contenedor físico de la información. Un gestor de base de datos es por ejemplo ORACLE, que es un software, en donde adentro vamos a tener una o muchas bases de datos. Otros gestores de base de datos son MySQL, SQL server de Microsoft, también Microsoft Access que es un tipo de base de datos más hogareñas. Es decir, son diferentes herramientas capaces de gestionar grandes volúmenes de información.

Aplicaciones de los sistemas de bases de datos: el tipo de aplicaciones que se almacenan en estas bases de datos y las administran los gestores son por ejemplo:

- Banco: cualquier tipo de transacciones bancarias, se administran a través de un gestor.
- Líneas aéreas: reservas de pasajes, información de la planificación de vuelos, pilotos, tickets vendidos, devoluciones, etc.
- Universidades: como dijimos, registros, alumnos, cursos, etc.
- Ventas: todos los gestores de bases de datos administran información de clientes, productos, compras, proveedores, resúmenes de cuentas, stock, etc.
- Minoristas en línea: el seguimiento de ordenes de compras, recomendaciones personalizadas, etc.
- Producción: gestionar todo lo que sea software de producción de la empresa, inventarios, pedidos, cadena de producción, avance de fabricación, etc.
- Recursos humanos: información sobre los empleados, salarios, impuestos, beneficios, capacitaciones, crecimiento del personal, evaluación del desempeño, etc.

_ Las bases de datos y los gestores, nos permiten administrar información relacionada con todos los aspectos de nuestra vida. La mayoría de la información que nosotros estamos manipulando o vemos en internet está montada sobre gestores de bases de datos.

Objetivo principal de los sistemas de bases de datos

_ Inicialmente, las aplicaciones de los sistemas de bases de datos se construyeron encima de los sistemas de archivos, o sea 30 años atrás no existían los gestores entonces se tenían archivos con grandes volúmenes de información que se construían con cierta estructura y diseño para guardar información. El gestor de bases de datos centraliza este manejo de archivos, toma el control del manejo de archivos y empieza a administrarlo según su criterio para garantizar que estos archivos y datos sean consistentes. Cuando se usaban sistemas de archivos uno de los mayores riesgos o inconvenientes que había eran:

- Redundancia o inconsistencia de datos: nos olvidamos de algo o construimos algo de una forma inconsistente, diversos formatos de archivos, duplicación de la información en diferentes archivos, volviendo difícil el acceso a los datos. El SGBD logra concentrar toda la información en un solo ámbito de manera tal que nosotros estemos seguros de donde tenemos que buscar esa información y como esta información va a estar guardada sin riesgos a que la información este desparramada por distintos lugares y tenga un rápido acceso (consistente).
- Dificultad en el acceso a los datos: necesidad de escribir un programa nuevo por cada tarea nueva.
- Aislamiento de datos o diversos archivos con diferentes formatos. El gestor de base de datos aísla la información del resto de las herramientas, entonces podemos acceder a la información desde una central telefónica, aplicación móvil, página web, software de computadora, y todos accedemos al mismo contenedor de bases de datos, por ejemplo el acceso al saldo de un banco desde distintos lugares pero el importe o información es el mismo y se extrae del mismo lugar.
- Problemas de integridad: la ventaja ahora de un gestor de base de datos es resolver estos problemas de integridad. Por ejemplo los sistemas de archivos tenían problemas con las restricciones de consistencia (ejemplo el saldo de una cuenta bancaria > 0) que pasan a formar parte del código del programa que se hace, en donde había dificultad para añadir restricciones nuevas o modificar las existentes. Los controles de integridad están monitoreados por el gestor de base de datos, en donde esto ya no depende de un programador que se equivoque o no en el software que realiza, porque el gestor de base de datos es el que tiene el control sobre la información. Por ejemplo, por más que el software permita colocar una calificación de 11 a un alumno, el gestor de base de datos lo impide ya que permitiría poner de 1 a 10, entonces es el SGBD quien tiene el control de las calificaciones en el caso de la universidad.
- Problemas de atomicidad: los fallos a veces pueden dejar a las bases de datos en un estado de inconsistencia si se han llevado a cabo actualizaciones parciales, en donde el gestor de base de datos no admite que la base de datos quede inconsistente. Por ejemplo, si queremos hacer una transferencia bancaria, el

gestor de base de datos de ninguna manera va a dejar que se saque dinero de una cuenta y no se transfiera otra, ya que o se hace la transferencia o no se hace nada. El gestor de base de datos no dejaría que se produzcan fallos de integridad en la base, pero antes esto si sucedía con el manejo se sistema de archivos.

- Anomalías en el acceso concurrente: esto hace referencia por ejemplo a que no pueden haber dos personas modificando un saldo de una cuenta bancaria al mismo momento, es por eso que el control de concurrencia es realizado por el gestor de base de datos, que nos da la seguridad de que esto no puede suceder, en donde el sistema debería permitir que el primero que haya optado o llego más rápido para realizar una operación será el que hará la operación y el otro no, porque la instancia simultanea no se permite. Se necesita el acceso concurrente para obtener un buen rendimiento, pero el acceso concurrente sin control puede conducir a tener datos inconsistentes.
- Problemas de seguridad: el gestor de base de datos dice que puede hacer cada persona, que permisos tiene y que puede tocar cada una, el mismo motor de base de datos nos permite administrar estos problemas de seguridad. El sistema de control de seguridad del gestor de la base de datos nos garantiza los accesos a la información que este habilitada. Por ejemplo, un médico podría ver la información de su paciente pero no ve que les pasa a otros pacientes que él no atiende por más que la información este toda metida en la misma base de datos. Estas son restricciones de seguridad que el mismo motor nos permite administrar. El SGBD resuelve la dificultad de proporcionar acceso a parte de los datos pero no a todos.

_ Entonces el objetivo principal de un gestor de base de datos es administrar información, y administrarla de manera certera y segura, además de ofrecer soluciones para todos los problemas anteriores en los que antes se encargaba de solucionarlos manualmente un ingeniero.

Niveles de abstracción

_ Un gestor de base de datos nos ofrece una abstracción, es decir, diferentes niveles de abstracción a como están guardados los datos.

Nivel físico: los datos están guardados a un nivel físico en algún lugar del disco duro en algún lugar del mundo, por ejemplo, los datos físicamente están en el disco duro del servidor del campus de la UCC. En este nivel se describe como se almacenan realmente los datos, por ejemplo, el nombre del cliente.

Nivel lógico: este me hace interpretar que los datos, por ejemplo, la clave del alumno, el nombre, su dirección, nombre de padres, etc, parecen ser que están conectados aunque a nivel físico quizá estos datos puedan estar en cualquier lugar diferente, pero a nivel lógico vamos a tener la sensación de que todos los datos del alumno parecen estar todos

metidos en un mismo lugar físico cercano o relacionado. Este nivel describe que datos se almacenan en la base de datos y las relaciones que existen entre ellos.

Nivel de vistas: en este, el mismo gestor de la base de datos me puede ocultar o mostrar información de acuerdo a lo que me haga falta. Volviendo con el ejemplo de los datos del alumno, a nivel físico los datos de los alumnos puede que estén en el mismo disco duro o en distintos, a nivel lógico vamos a tener la sensación de que todos los datos del alumno están todas en el mismo lugar y a nivel de vistas yo me creo que en la base de datos solamente están los alumnos de ingeniería cuando en realidad puede ser que estén todos los de toda la universidad mezclados. Los programas de aplicación ocultan tipos de datos. Las vistas también pueden ocultar información por razones de seguridad.

_ Entonces, el nivel de abstracción si se quiere es, hay algo que esta guardado a nivel físico en el disco duro, hay algo lógico que nos parece ser que vemos, y están las vistas que es lo que nos creemos finalmente que sucede aunque lógicamente la información esta ordenada de otra forma y físicamente también este de otra forma. El gestor de base de datos va a administrar todo esto, que se va a volver para nosotros todo esto absolutamente transparente o intrascendente, ya que no importa donde esta guardada la información, sino que sabemos que el gestor va a colectar la información que necesitamos y nos la va a mostrar en el momento que nos haga falta independientemente de donde estén los datos físicamente y como estén lógicamente guardados.

Visión de los datos

_ Nosotros vamos a tener una visión de la información, la cual creemos que es la real, aunque físicamente por debajo haya otra cosa que esté dispuesta totalmente de manera diferente, y de esto se va a encargar el gestor de base de datos, pero para el usuario esto es absolutamente transparente, ya que el mismo no sabe y no le importan donde están los datos, sino que cree que lo único que ve es lo que existe, aunque solo sea una vista que nos hace creer lo que vemos, es decir, lo que tenemos es una visión de los datos que es lo que el gestor nos deja creer.

_ Cada uno tiene una vista de lo que puede hacer, aunque toda la información este contenida en la misma base de datos y administrada por el mismo gestor de base de datos.

Instancias y esquemas

_ Existen dos conceptos muy importantes en el manejo de las bases de datos que son los que se denominan esquemas e instancias. Esto es similar a los tipos y variables en los lenguajes de programación.

Esquema: es la estructura lógica de la base de datos. En un esquema de la base de datos vamos a guardar, por ejemplo, las notas de los alumnos en donde las notas tienen una forma (números del 0 al 10 y no pueden ser otros). Ahora, cuando definimos el esquema, definimos en qué lugar del disco duro los vamos a guardar, que forma tiene ese dato (numero, float, long), etc. El esquema cambia muy rara vez. Otro ejemplo, la base de datos se compone de información acerca de un grupo de clientes y cuentas, y de las relaciones entre ellos, análogo a la información del tipo de una variable en un programa:

- Esquema físico: diseño de la base de datos a nivel físico.
- Esquema lógico: diseño de la base de datos a nivel lógico.

Instancia: es el contenido real que tiene la base de datos en un instante de tiempo determinado. Por ejemplo, en el momento en el que se inicia una materia, ningún alumno todavía se sacó ninguna nota, entonces la instancia en ese momento, para la materia base de datos por ejemplo, es que nadie se sacó ninguna nota en ningún parcial, pero en unos días la instancia va a empezar a mostrar que notas tiene cada uno en el primer parcial, en el segundo, etc. Las instancias van cambiando constantemente y el esquema es siempre el mismo o por lo menos durante mucho más tiempo va a ser el mismo, ya que en algún momento puede ser que se cambie el esquema, en donde por ejemplo se puede decidir que ahora los alumnos se podrían sacar un 11. Por ejemplo, el esquema dice que no puedo tener un saldo negativo en el banco, pero la instancia dice en este momento que tengo un saldo de $3000 pero si saco $1000 ya me quedan $2000, donde la instancia cambia constantemente pero el esquema es el mismo.

_ El gestor de base de datos nos va a permitir definir los esquemas y nos va a permitir administrar las instancias.

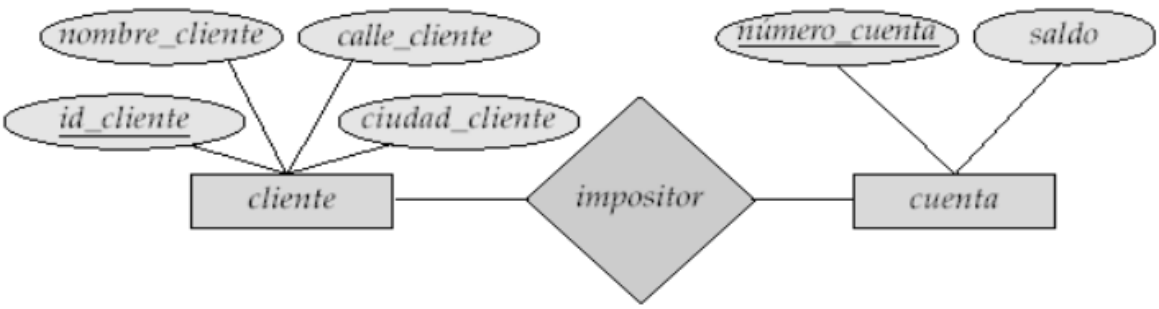
Independencia física de los datos: es la capacidad de modificar el esquema físico sin cambiar el esquema lógico. Las aplicaciones dependen del esquema lógico. En general, las interfaces entre los diferentes niveles y componentes deben definirse adecuadamente de modo que los cambios en algunas partes no influencien otras seriamente.

Modelos de datos

_ El modelo de datos es una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y consistencia entre esos datos. A los datos, las relaciones, la semántica y la consistencia entre los mismos, lo administra el gestor de base de datos. Por ejemplo, por un lado tenemos el nombre de un profesor, y por otro el nombre de un alumno, pero por otro lado existe una relación en donde el profesor Juan Vulcano da la materia base de datos y hay alumnos como Pedro, Martina, etc, que en este momento están cursando la materia, entonces existe una relación entre el profe y los alumnos aparte de los datos del profe y los alumnos. Existe también una semántica de los datos (datos numéricos, alfanuméricos, fechas, etc), y existe una consistencia de los datos (nota no > 10 y < 0), todo esto gestionado por el gestor de base de datos. Todo esto va a funcionar por medio de lo que se llaman relaciones.

Modelo relacional: es una colección de tablas para representar los datos y sus relaciones. Cada tabla tiene columnas con un nombre único que representan atributos y cada atributo tienen ciertas propiedades. Este modelo está basado en registros y es el modelo de datos más ampliamente usado. El modelo relacional establece vínculos.

Modelo entidad-relación: (principalmente para el diseño de bases de datos) que es el diseño de base de datos que trata de modelar lo que pasa en el mundo real en un esquema que se denomina entidad-relación. Se basa en una percepción del mundo real y consiste en una colección de objetos básicos, entidades y relaciones. Modela una empresa como una colección de entidades y relaciones. Se representa gráficamente mediante un diagrama entidad-relación.



Modelo de datos orientado a objetos: parece una extensión del modelo entidad-relación. Incluye los conceptos de encapsulación, métodos y la identidad de los objetos.

Modelo de datos semiestructurados: permite la especificación de datos donde los elementos de datos individuales del mismo tipo pueden tener diferentes conjuntos de atributos. La posibilidad de especificar nuevas etiquetas y crear estructuras de etiquetas anidadas convierten a XML en un mecanismo perfecto para el intercambio de datos, no solo de documentos.

Modelos anteriores:

- Modelo de red.
- Modelo jerárquico.

Lenguaje de manipulación de datos (LMD)

_ LMD es un lenguaje para acceder o manipular los datos organizados mediante el modelo de datos apropiado. El LMD también se conoce como lenguaje de consultas. Existen dos clases de lenguajes:

Procedimentales: el usuario especifica que datos se necesitan y cómo han de obtenerse dichos datos.

Declarativos (no procedimentales): el usuario especifica qué datos se necesitan sin especificar cómo se han de obtener.

SQL

_ SQL es un lenguaje de consulta de datos (este es el lenguaje central) estándar y ampliamente utilizado, que tiene dos partes, una es el lenguaje de manipulación de los datos que nos permite generar y modificar todas las instancias. Y la otra parte es el lenguaje de definición de datos que es el lenguaje que nos permite crear los esquemas. SQL es un lenguaje no procedimental.

Lenguaje de definición de datos (LDD)

_Es la notación de especificación para definir el esquema de la base de datos. El compilador LDD genera un conjunto especial de tablas denominado diccionario de datos. El diccionario de datos contiene metadatos:

- Esquema de base de datos.
- Lenguaje de almacenamiento y definición de datos. Especifica la estructura de almacenamiento y los métodos de acceso utilizados.
- Restricciones de integridad, restricciones de dominio, integridad referencial (restricción references en SQL), y asertos.
- Autorización.

Diseño de base de datos

_ Debemos saber cómo armar la estructura de información, para poder almacenar todo el volumen de datos y que ese diseño tenga un criterio lógico y ordenado de acuerdo a lo que dice el negocio y de acuerdo a ciertas decisiones informáticas que son las que nos van a decir como guardar los datos en la base. Proceso de diseño de la estructura general de una base de datos:

Diseño lógico: decidir el esquema de la base de datos. El diseño de la base de datos requiere encontrar una buena colección de esquemas de relación:

- Decisión de negocio: atributos que se deberían registrar en la base de datos.
- Decisión informática: relación de esquemas que se deberían utilizar y cómo se deberían distribuir los atributos entre los distintos esquemas de relación.

Diseño físico: decidir sobre las características físicas de la base de datos.

Gestión de almacenamiento

_ El gestor de almacenamiento hace referencia a como se guarda u organiza la información en el disco duro, también hace referencia a como se generan índices y asociaciones. Es un módulo de programa que proporciona la interfaz entre los datos de bajo nivel en la base de datos y los programas de aplicación y consultas emitidas al sistema. El gestor de almacenamiento es responsable de las siguientes tareas:

- La interacción con el gestor de ficheros.
- El almacenamiento, recuperación y actualización eficiente de los datos.

Gestión de transacciones

Transacción: es una colección de operaciones que se llevan a cabo como una única función lógica en una aplicación de base de datos.

Componente de gestión de transacciones: asegura que la base de datos permanezca en un estado consistente (correcto) a pesar de los fallos del sistema (por ejemplo, fallos de energía y caídas del sistema operativo) y de los fallos en las transacciones.

Gestor de control de concurrencia: controla la interacción entre las transacciones concurrentes para asegurar la consistencia de la base de datos.

Arquitectura de la base de datos

_ La arquitectura de una base de datos se ve muy influenciada por el sistema informático subyacente sobre el que se está ejecutando:

- Centralizado.
- Cliente-servidor: donde nos creemos que los usuarios son clientes y los servidores nos proveen la información, como sucede en la realidad.
- Paralelo (multi-procesador).
- Distribuido.

Modelo entidad-relación: diseño de base de datos

_ Nosotros tenemos que trabajar en un proceso de diseño, nos contactamos con alguien del mundo, que nos cuente el problema que tiene y nosotros deberíamos modelar como resolver ese problema que nuestro cliente nos plantea. En muchos casos este problema está vinculado a utilizar o manejar ciertos datos o información, entonces el modelado del diseño de la base de datos donde vamos a guardar la información que va a administrar nuestro software o programa, para que el cliente utilice, el buen diseño de esta base de datos es clave para los resultados del software que vamos a desarrollar. Los software seguramente puedan seguir funcionando bien más allá que la base de datos no sea buena, pero un mal diseño de la base de datos nos va a generar muchas dificultades luego al momento de administrar el software y al momento de producir los resultados que el cliente quiere. Un buen diseño inicial nos simplifica mucho más luego toda la construcción del software (esto es como hacer bien los planos de una casa acorde a lo que el cliente pide para que luego no este cambiando cosas ya que el diseño original no fue lo apropiado).

Proceso de Diseño

_ El diseñador de la base de datos debe interactuar con los usuarios para comprender las necesidades y los requerimientos, donde el objetivo es construir una base de datos que pueda cumplir con los requisitos del mismos. El responsable de diseñar la base de datos no siempre es conocedor del negocio sobre el cual va a trabajar, entonces deberá reunirse con el usuario o los expertos para levantar requerimiento y asi poder entender todo lo que la base de datos, que va a administrar el software, debe cumplir como requisitos para lo que el cliente desea. A partir de ahí surge lo que se denomina como diseño conceptual

Fase inicial: el diseñador debe interactuar con los expertos y usuarios del dominio.

Diseño conceptual: el diseñador elige el modelo de datos, y aplicando sus conceptos, traduce los requisitos en un esquema conceptual donde se describe lo que el cliente pide.

Especificación de requisitos funcionales: los usuarios describen los datos que necesitan pero también describen las operaciones que se llevaran a cabo con estos datos.

Paso del modelo abstracto al modelo físico: donde primero se traduce el esquema conceptual de alto nivel al modelo de datos de la implementación del sistema de bases de datos. Luego se pasa al modelo físico en el que se especifican las características físicas de la base de datos, archivos y estructuras internas.

_ El proceso de diseño tiene que ser consistente, y debe evitar 2 peligros:

- Redundancia: cuando por ejemplo que haya datos que están en 2 o 3 lugares, es decir, tenemos que evitar la información duplicada ya que genera inconsistencia.
- Incompletitud: que es la falta de datos, no nos tenemos que olvidar de nada de lo que el cliente nos pide.

_ Para evitar estos vamos a hacer algunas acciones que nos permitan evitar estos errores. Por ejemplo si tenemos dos relojes marcando la hora es probable que ya no sepamos cual es verdaderamente la hora porque muy difícilmente los dos relojes marquen exactamente la misma hora, ya que va a haber una pequeña diferencia entre ambos. Entonces si tenemos un solo reloj (si el dato está en un solo lugar), entonces asumimos que esa es la hora la que marca nuestro reloj, pero si tenemos dos relojes ya puede ser que los dos no marquen exactamente lo mismo y se nos produce una duda o inseguridad de cuál es la hora verdadera, por lo que la redundancia, o sea el duplicar los datos, nos puede llevar a una instancia de duda si ese dato es correcto o no.

Modelado

Modelo entidad-relación

_ Para modelar nuestra base de datos, vamos a utilizar un modelo que se llama modelo entidad-relación, que es un modelo que tiene ciertas características y que nos permite representar mediante un diagrama un problema del mundo real. Entonces, decimos que una base de datos que vamos a modelar se puede diseñar como una colección o grupo de entidades y las relaciones que existen entre esas entidades.

Entidad: es un objeto del mundo real que es distinguible de todos los demás, por ejemplo una persona, una empresa, un caso, un aula, un auto, una planta, etc, en donde cada objeto o cosa del mundo real la vamos a considerar una entidad.

Atributos: las entidades tienen atributos, por ejemplo las personas tienen nombres, edades, direcciones, fechas, etc. Estas son ciertas características o propiedades que se le atribuyen a cada entidad.

Conjunto de entidades: es un grupo de entidades del mismo tipo que comparten las mismas propiedades o atributos, por ejemplo el conjunto de todas las personas con atributos como nombre, apellido, DNI, dirección, etc, conforman el conjunto de entidades personas, y esto también es válido para empresas, árboles, vacaciones, etc, todas entre sí.

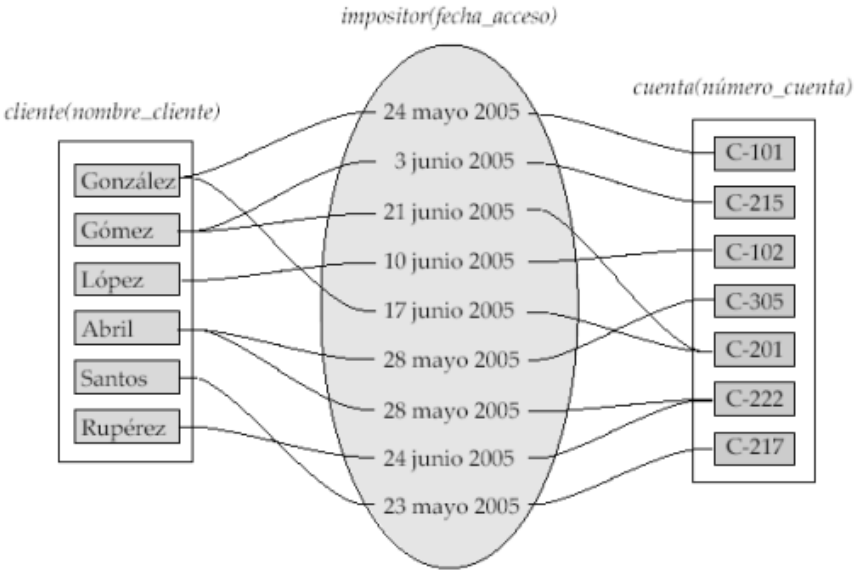
Relación: es una asociación o vinculación entre varias entidades, en donde por ejemplo tenemos relaciones para vincular a las personas, y podemos decir que Juan Vulcano que es una entidad, en este momento es profesor de Facundo que es un alumno y otra entidad, entonces existe una relación entre el profesor y el alumno, donde Juan es profesor de Facundo en la materia base de datos, que aparte la materia base de datos es una entidad

y el conjunto de materias conforman el conjunto de entidades materia y pensamos lo mismo con el conjunto de entidades profesor y alumnos.

Conjunto de relaciones: es un conjunto por ejemplo que relaciona a todos o algunos profesores con todos o algunos alumnos con todas o algunas materias, y asi construimos el modelo entidad-relación. Una relación matemática entre $n \geq 2$ entidades, cada una de ellas tomadas de los conjuntos de entidades, por ejemplo Juan Vulcano le da clases a todo el curso en la misma materia, donde entidades se vinculan con otras pero con las mismas características. Cada conjunto de relaciones tiene también sus atributos y propiedades.

_ Un ejemplo podemos plantear es que Santiago tiene una cuenta bancaria, en donde Santiago es una entidad para el banco, o sea es un cliente, y existe otra entidad que es la cuenta bancaria, y existe una relación en la que Santiago es el dueño de la cuenta. Otro ejemplo seria en mercado libre, donde tenemos un comprador, un producto y un vendedor, es decir, tres entidades con distintos permisos y existe una relación entre el comprador que compra determinado producto a un vendedor y asi se establece la relación. También como ejemplo podemos decir que en el seno de una familia, todas las personas son entidades y la relación es de orden de jerarquía pero la relación madre hijo es una relación puntual que no es la misma entre madre y padre, en donde la persona en si es una entidad y ser madre, esposo o hijo es la relación que los vincula. Como ejemplo también tenemos un avión es una entidad y un piloto es otra entidad, y existe una relación que dice que el señor que es piloto va a manejar el avión y va a hacer un vuelo de tal lugar a otro, en donde el vuelo parece ser otra entidad en si misma donde compramos un ticket de ese vuelo, y asi aparecen entidades.

Atributos descriptivos: una relación puede tener atributos propios denominados atributos descriptivos de la relación, que pueden ser también propiedad de un conjunto de relaciones. Por ejemplo decimos que Gonzales el día 24 de mayo de 2005 abrió una cuenta bancaria que es la cuenta 101, entonces Gonzales es la entidad cliente, la cuenta 101 es otra entidad y se estableció una relación en la que Gonzales abrió la cuenta y la hizo un determinado día. Otro ejemplo, una madre con un hijo, en la relación madre-hijo una de las comidas que más le gusta al hijo que le haga son milanesas, y en tal fecha, tal cantidad, etc.



Grado de un conjunto de relaciones

_ El grado de conjunto de relaciones referencian a que si yo vinculo dos entidades, la relación es de grado dos, si vinculo tres entidades la relación es de grado tres, y asi sucesivamente. Por ejemplo, profesor con alumno y con matera es una relación de grado tres, y si agregamos un aula la relación es de grado cuatro. Esto se refiere al número de conjuntos de entidades que participan en un conjunto de relaciones. Los conjuntos de relaciones que implican a dos conjuntos de entidades se denominan binarios (o de grado dos), ternarias, cuaternarias, etc. En general, la mayoría de los conjuntos de relaciones en un sistema de bases de datos son binarios, las relaciones entre más de dos conjuntos de entidades no son muy comunes.

Atributos

_ Una entidad se representa mediante un conjunto de atributos, que describen propiedades que posee cada miembro de un conjunto de entidades. Por ejemplo la entidad cliente podría tener los atributos id-cliente, nombre-cliente, calle-cliente, ciudad-cliente, etc, y la entidad préstamo podría tener los atributos número-préstamo, cantidad, etc. Los conjuntos de entidades son los que comparten distintos atributos, por ejemplo todos los clientes tienen un id-cliente, todos los alumnos de la facultad tienen una clave UCC, dirección, nombre, apellido, etc, es decir, serian atributos relacionados con esa entidad, como los alumnos.

Dominio: el conjunto de valores permitidos para cada atributo.

_ Tenemos los siguientes tipos de atributos:

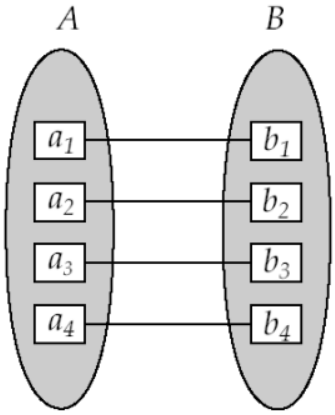
- Compuestos: por ejemplo, nombre completo (primer nombre, segundo nombre, primer apellido y segundo apellido), dirección (calle, ciudad, provincia, código postal), calle (número, nombre, dpto, piso), etc. Por otro lado cada uno de los elementos de dirección son atributos simples.
- Simple: tenemos por ejemplo calle, numero, dpto, piso, barrio, etc.
- Monovalorados: son aquellos que toman un solo valor.
- Multivalorado: por ejemplo, el número de teléfono, en donde hay personas que tienen dos o más.
- Derivados: se pueden derivar de los valores de otros atributos. Por ejemplo el atributo no derivado es fecha de nacimiento (atributo rígido) y con este podemos calcular la edad, en donde este sería un atributo derivado (atributo que puede cambiar). No es conveniente que estos estén en la base de datos, sino que hay que calcularlos en un momento y traerlos para que cualquier cambio que se produzca en la base yo pueda tener actualizado el dato y no tener redundancia e inconsistencia. Estos se pueden calcular en función de otra información de otro atributo.

Restricciones

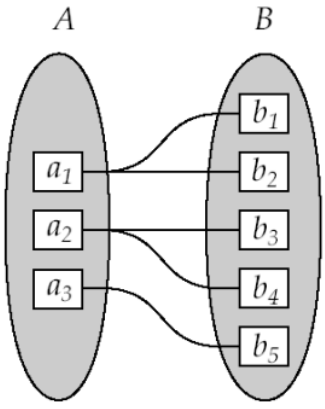
_ En un esquema de desarrollo entidad-relación se pueden definir ciertas restricciones a las que el contenido de la base de datos se debe adaptar. Algunas que pueden suceder son:

Correspondencia de cardinalidades: la cardinalidad expresa el número de entidades a las que se puede asociar otra entidad a través de un conjunto de relaciones. Es mejor para describir conjuntos de relaciones binarios, en donde para este caso la correspondencia de cardinalidades puede ser:

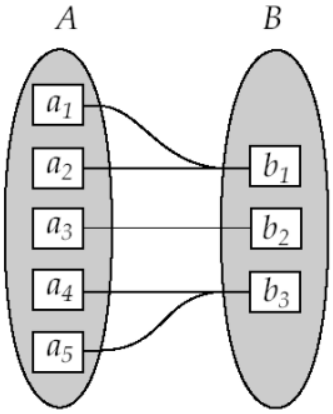
- Uno a uno: tomamos como ejemplo la materia base de datos que tiene un solo titular, entonces la relación entre materia y profesor titular parece ser uno a uno. Otro ejemplo puede ser una facultad con un decano.



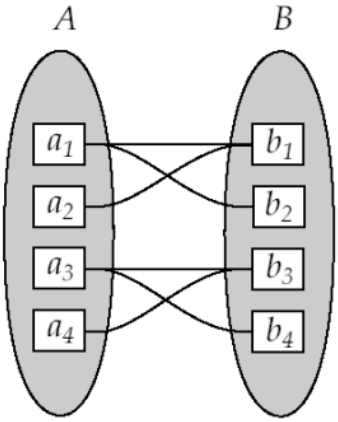
- Uno a varios: tomamos como ejemplo una materia que es dictada por varios profesores (titular, jefe de TP, ayudante de catedra). Otro ejemplo sería una persona que puede tener varios autos.



- Varios a uno: podríamos considerar que muchos hijos tienen una sola madre.



- Varios a vario: tomamos como ejemplo que muchos alumnos pueden cursar muchas materias y viceversa muchas materias son cursadas por muchos alumnos.



_ Hay que hablar con el especialista y entender como es el negocio. Si no se entendió bien las cardinalidades, mi modelo seria incorrecto.

Restricciones de participación: son parecidas a las de cardinalidad pero no iguales . Se dice que un conjunto de entidades tiene una participación total, si la cantidad E participa al menos en una relación R. Y se dice que es parcial, si la entidad E puede o no participar en la relación. De otra forma, decimos que se dice que la participación de un conjunto de entidades E en un conjunto de relaciones R es total si cada entidad de E participa al menos en una relación de R. Si sólo algunas entidades E participa en relaciones de R, se dice que la participación es parcial. Por ejemplo una mama puede tener muchos hijos, donde la cardinalidad era uno a varios, ahora nos preguntamos si es posible que la señora no tenga

ningún hijo, y cómo es posible, la participación madre con hijo puede ser parcial, o sea que puede ser que no tenga ningún hijo, ahora nos preguntamos si hijo puede ser que no tenga ninguna madre, y como esto no es posible entonces hijo se vincula con madre en participación total. Otro ejemplo, un cliente puede comprar muchos productos, un producto puede ser comprado por muchos clientes, la relación entre estos dos es varios a varios, ahora un cliente puede ser que no compre nada por lo tanto la participación es parcial, además a un producto puede ser que no lo compre nadie por lo tanto la participación también es parcial. Por ejemplo, se puede esperar que cada entidad préstamo esté relacionada al menos con un cliente, por lo tanto la participación de préstamo en el conjunto de relaciones prestatario es total. En cambio un individuo puede ser cliente de un banco aunque no tenga concedido ningún préstamo, por lo tanto la participación de cliente en la relación prestatario es parcial.

_ Siempre que establezcamos relaciones entre las entidades deberíamos mirar que vinculo hay, que cardinalidad tienen, y que participación tienen. Esto es importante porque tanto la cardinalidad como la participación nos va a ayudar al momento de diseñar la base de datos.

Claves: conceptualmente cada entidad es distinta de las otras. Desde el punto de vista de la base de datos esto se debe expresar en función a los atributos. Esto significa que no se permite que haya dos entidades que tengan exactamente el mismo valor de todos sus atributos, por ejemplo, no puede haber dos personas con mismo nombre, DNI, dirección, teléfono, etc, sino que tiene que haber algo que diferencia a una persona de la otra. Esto es una necesidad esencial para el diseño de la base de datos, no podemos admitir que dos entidades sean iguales, sino que tiene que haber algo que las diferencia una de la otra. Cada entidad debe ser identificable de otra entidad, y para eso necesitamos que uno o varios atributos sean diferentes ya que como dijimos antes, no se permite que ningún par de entidades de un conjunto de entidades tenga exactamente el mismo valor en todos sus atributos.

- Superclave: se denomina superclave, de un conjunto de entidades, al conjunto de uno o más atributos que permiten identificar unívocamente una entidad determinada. Por ejemplo, si hablamos de los alumnos de la facultad una superclave podría ser clave de alumno, el DNI no porque en un país puede que no se repita pero quizás en otro país este el mismo número, entonces una super calve seria DNI + nacionalidad, ya que las superclaves pueden ser un atributo o varios. El mail es otro componente que nos identifica por lo tanto es otra super clave, al igual que el número de celular. La firma digital es una superclave, al igual que el número de pasaporte. Un ejemplo podría ser también Apellido + nacionalidad + DNI puede ser una superclave pero parece ser que sobra el apellido, también podemos tener Id_cliente + nombre del cliente. Sea $K \subseteq R$, en donde K es una

superclave de R si los valores de K son suficientes para identificar una tupla única de entre todas de la relación r(R), y K es una clave candidata si K es mínima.

- Clave candidata: una clave candidata de un conjunto de entidades es una superclave lo más chica que se pueda, mínima. Por ejemplo, nacionalidad + DNI (sin apellido), clave alumno, número de cuenta, id cliente, pasaporte, huella dactilar, iris del ojo.
- Clave primaria: de todas las claves candidatas que tengo, elijo la que más me gusta y porque sospecho que no va cambiar nunca, para el problema que estoy resolviendo. Aunque pueden existir varias claves candidatas, una de ellas se elige como la clave primaria, y esta es una decisión del diseñador de la base de datos. La clave primaria se debe elegir de manera que sus atributos, nunca o casi nunca cambien. Este es como el atributo esencial que me permite identificar a una persona. Las calves candidatas rechazadas son por lo general porque pueden cambiar, y las super claves no son efectivas. Se denomina clave primaria a una clave candidata que ha elegido el diseñador de la base de datos como medio principal para la identificación de las tuplas de una relación.

_ Las claves candidatas, primaria y superclave son propiedades de toda la relación, no de una tupla. La selección de una clave representa una restricción de la empresa del mundo real que se está modelando. La clave primaria debe escogerse de manera que los valores de sus atributos no se modifiquen nunca, o muy rara vez, como por ejemplo, el atributo domicilio o dirección no debe formar parte de una clave primaria.

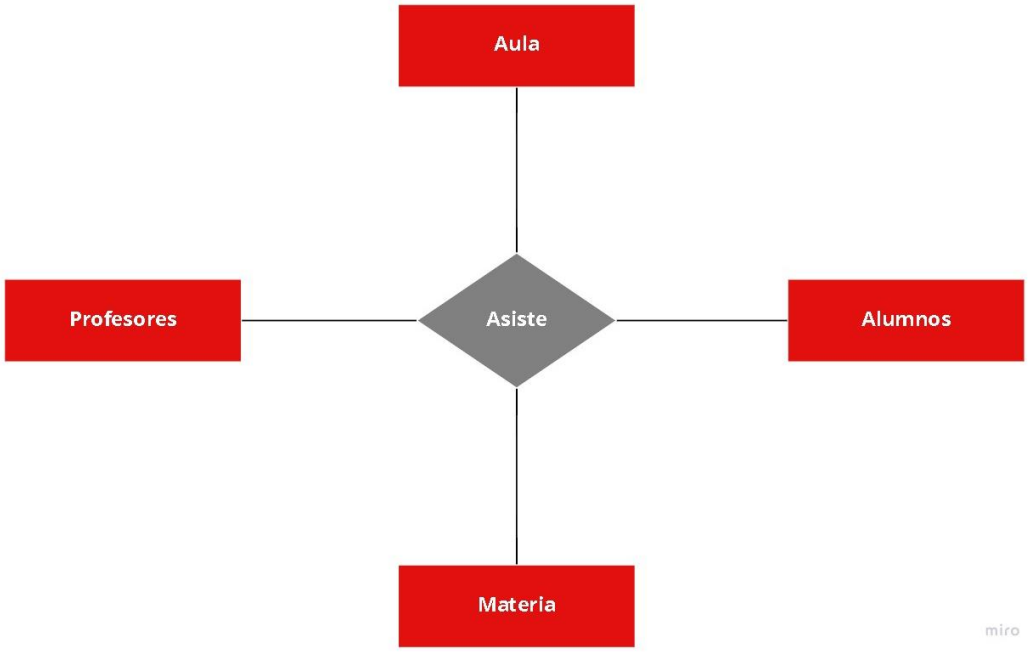
Conjunto de entidades: las claves, tanto primarias, candidatas y superclaves, no dependen de una entidad, sino que son propiedades o características del conjunto de entidades, más que de cada una de las entidades. La designación de una clave representa una restricción de la empresa real que se está modelando, es decir, no es algo que inventamos, ya que nosotros podemos crear la clave de alumno por ejemplo, pero la designación de esta como clave primaria o como clave candidata tiene algo del modelo real, en donde a partir de entender cómo funciona el modelo real aparece la clave. La clave no aparece como producción sino que aparece como una interpretación de la realidad que estoy mirando. Recordamos que los modelos entidad-relación intentan describir mediante un modelo o diseño de datos, tratando de representar un problema real que estoy mirando, y para eso hablo con el especialista para poder entender esa realidad. Además, dos entidades cualquiera del conjunto de entidades no pueden tener el mismo valor de los atributos de su clave al mismo tiempo.

Conjunto de relaciones: sea R un conjunto de relaciones que involucra a los conjuntos de entidades E_1 , E_2 , …, E_n. Y sea clave primaria (E_i) el conjunto de atributos que forma la clave primaria del conjunto de entidades E_i.

_ Si el conjunto de relaciones R no tiene atributos asociados, entonces el conjunto de atributos, clave primaria (E_1) U clave primaria (E_2) U ... U clave primaria (E_n), describe una

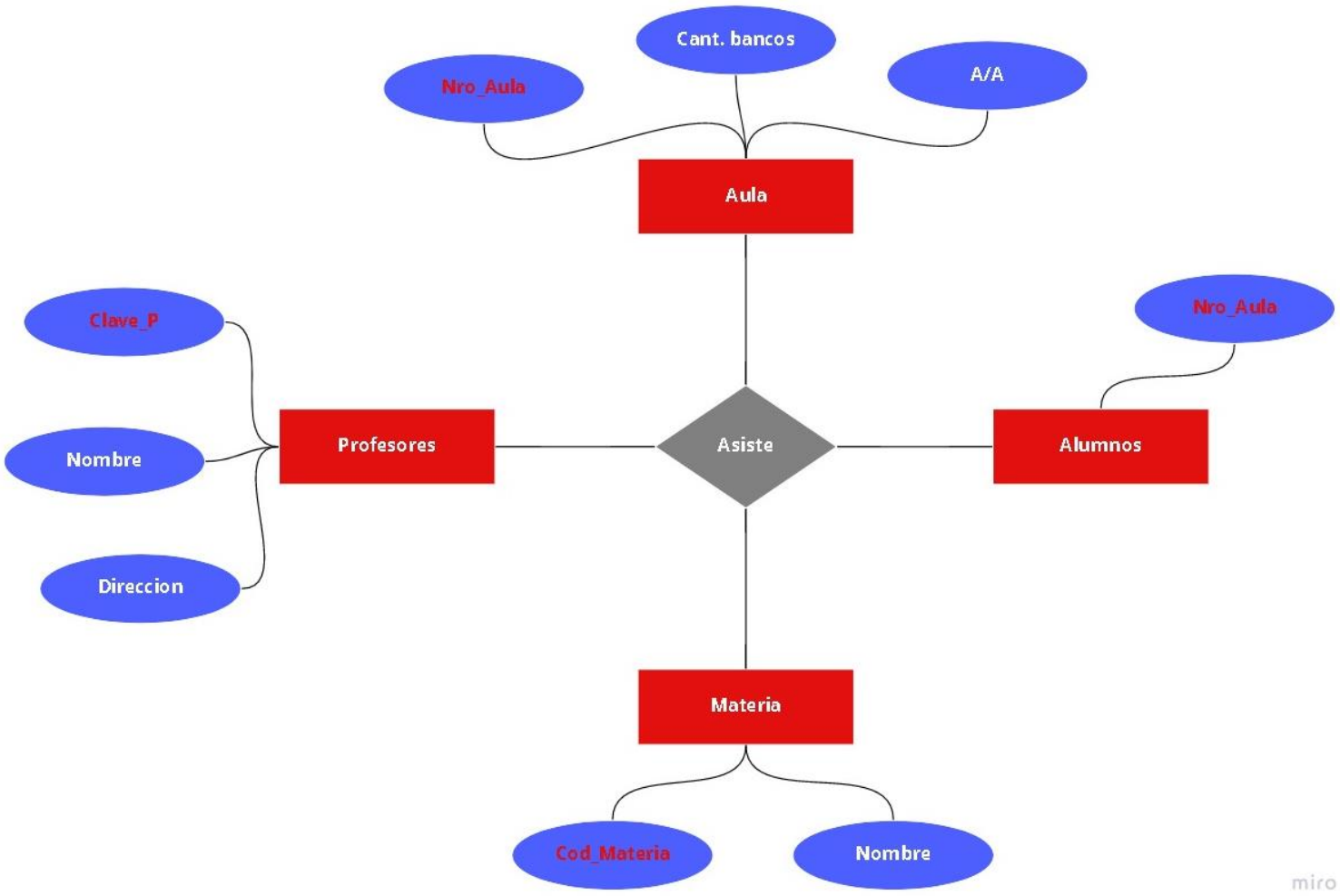
relación concreta del conjunto R. Es decir, la clave primaria se conforma con las claves primarias de cada una de las entidades que se relacionan. En cambio si el conjunto de relaciones R tiene asociado los atributos { a_1 , a_2… a_m}, entonces el conjunto de atributos, clave primaria (E_1) U clave primaria (E_2) U …. U clave primaria (E_n) U { a_1 , a_2… a_m}, describe una relación concreta del conjunto R. Es decir, puede ser que la clave primaria se conforme como la unión de las claves primarias de todas las entidades más algunos atributos propios de la relación R.

_ Nosotros tenemos dos o más entidades que se establecen un vínculo, donde las relaciones pueden ser binarias, ternarias, cuaternarias, etc. Por ejemplo un profe (entidad 1) da clase en una materia (entidad 2) a determinados alumnos (entidad 3) en una determinada aula (entidad 4), en donde establecemos una relación entre estas cuatro entidades, y como vemos en el gráfico, mediante un concepto que es el de asistir, o sea el profesor asiste a los alumnos en una determinada aula para dictar una determinada materia. El conjunto de relación asiste va a indicar a todos los profesores que tienen relación con todos los alumnos en determinadas aulas y en determinadas materias.

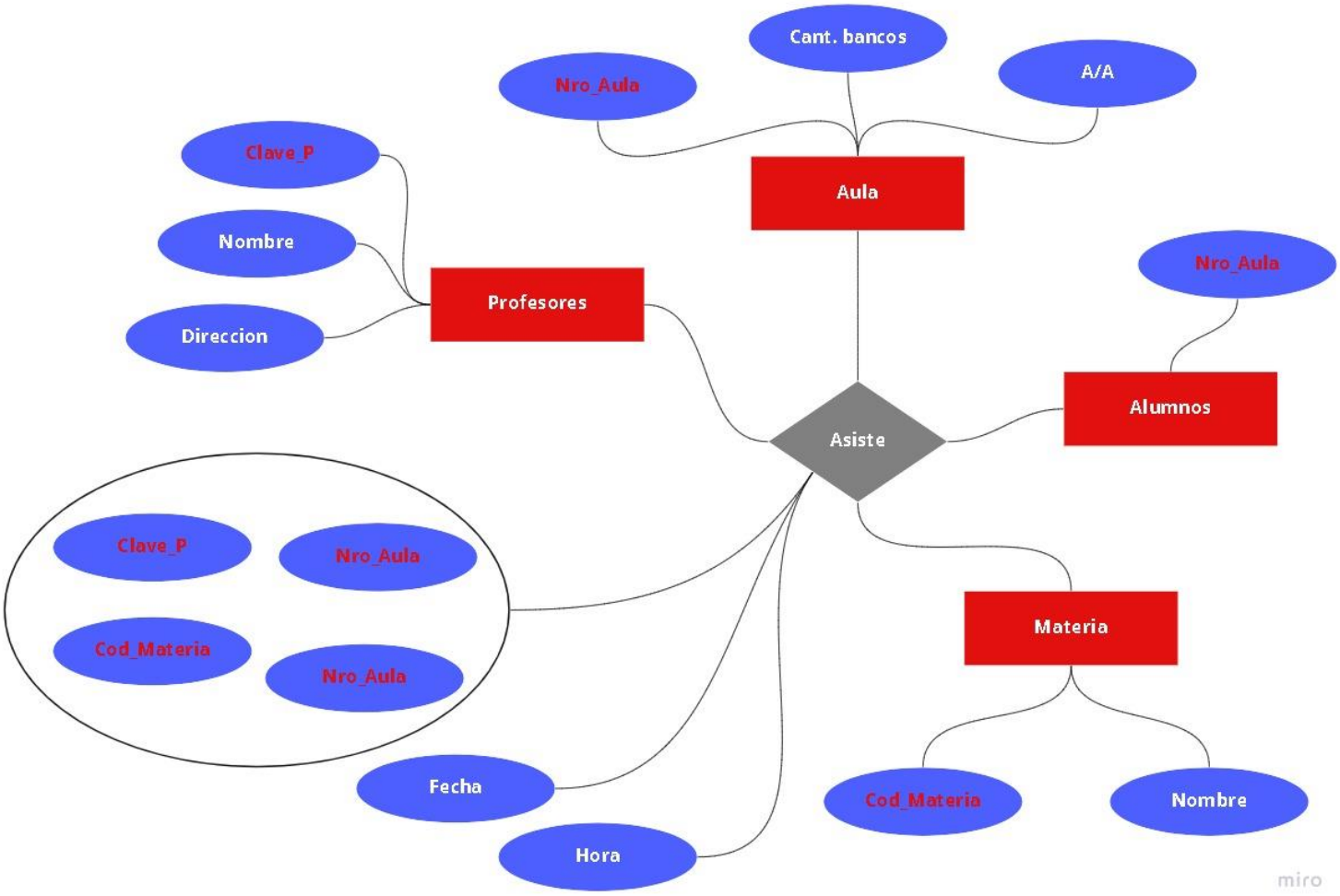


_ El profesor tiene atributos como la clave del profesor, nombre, dirección, entre otros. El atributo clave del profesor lo distinguimos porque es un atributo importante y representa la clave primaria que elegimos como representación de los profesores. Los alumnos también tienen sus atributos entre los cuales el atributo principal es la clave de alumno. Las aulas también tienen un atributo principal que podría ser número de aula que será el identificador de esa aula, y además tenemos otros atributos como cantidad de bancos, si tiene aire acondicionado, entre otros siempre relacionados con las aulas. En las materias, como puede ser que están dos materias con el mismo nombre donde una es para sistemas

y la otra para electrónica pero son distintas, además del nombre, deberíamos tener un identificador que puede ser el código de materia y seria la clave primaria.



_ Ahora, nos preguntamos cómo se conforma o construye la calve primaria de la relación asiste. Esta clave primaria se construye con las claves primarias de cada uno de los atributos de las entidades que se relacionan. Las relaciones tienen claves. Entonces la suma de todos los atributos son la clave primaria de la relación asiste. Y además, la relación asiste podría tener algunos otros atributos propios, como por ejemplo la fecha de la clase, la hora, etc. Pero la clave primaria se arma con la composición de las claves primarias de cada una de las entidades que se vinculan a través de esa relación.



_ Entonces, la combinación de la clave primaria de los conjuntos de relaciones se conforma con la suma de las claves primarias de cada una de las entidades que se vinculan con esa relación. Si además tuviéramos atributos asociados, podría ser que la clave primaria se conforme con la suma de las claves primarias de cada una de las entidades que se relacionan más algunos atributos propios de la relación. En este ejemplo, puede ser que la clave primaria se conforme con la suma de las cuatro claves más la fecha más la hora, pero esto va a depender del diseño y de la comprensión del mundo real en el que estemos trabajando. Lo que va a representar cada elemento de la relación es el cambio que vincula a un profesor con un alumno en esa materia en esa aula.

_ Los modelos de entidad relación terminan en algún momento convirtiéndose en información que se pasan a determinadas tablas de datos. La relación que existe se establece con las claves primarias. Si tenemos una relación, como vemos en la tabla, donde una entidad proveedor realiza envíos de una entidad partes, debemos tener una buena selección de la clave primaria, que sería el que identifica a los proveedores. En este caso no elegimos el nombre del proveedor porque podría haber proveedores que tengan

el mismo nombre, por eso por ejemplo S1, S2, S3, etc, me dice exactamente quien es el proveedor y sus datos. Con las partes sucede lo mismo, donde la identificación sería el número de partes P1, P2, P3, etc. Y la relación envíos que se genera, se establece con las claves primarias, entonces por ejemplo S1 P3 cant 400 donde el proveedor de P1 envía 400 partes de P4. Las relaciones se conforman con la clave primaria del proveedor, con la clave primaria de la parte y una unidad de cantidad.

PROVEEDORES

S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES
S2	JAIMES	10	PARIS
S3	BERNAL	30	PARIS
S4	CORONA	20	LONDRES
S5	ALDANA	30	ATENAS

PARTES

P#	PNOMBRE	COLOR	PESO	CIUDAD
P1	TUERCA	ROJO	12	LONDRES
P2	PERNO	VERDE	17	PARIS
P3	BIRLO	AZUL	17	ROMA
P4	BIRLO	ROJO	14	LONDRES
P5	LEVA	AZUL	12	PARIS
P6	ENGRANAJE	ROJO	19	LONDRES

ENVIOS

S#	P#	CANT
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	400
S4	P2	200
S4	P4	300
S4	P5	400

_ Si queremos saber que nota nos sacamos en un examen los datos que necesitaríamos son la clave del alumno, la clave primaria de la materia y la comisión, porque la nota es una relación que dice que tal alumno en tal materia se sacó una nota. Si un alumno se puede sacar muchas notas en una materia y si en esa materia hay muchos alumnos, la relación es muchos a muchos, entonces para ver la nota que nos sacamos necesitamos saber que alumno, en que materia y probablemente algún dato más por ejemplo si es en el primer o segundo parcial, donde la clave primaria se termina conformando de la suma de clave alumno, clave materia y algún atributo adicional como la fecha o número de parcial, para poder encontrar la nota misma. En el caso de un auto por ejemplo, tenemos atributos como la patente, el número de motor, y el número de chasis, donde estos son importantes en la identificación del vehículo.

Claves de conjuntos de relaciones

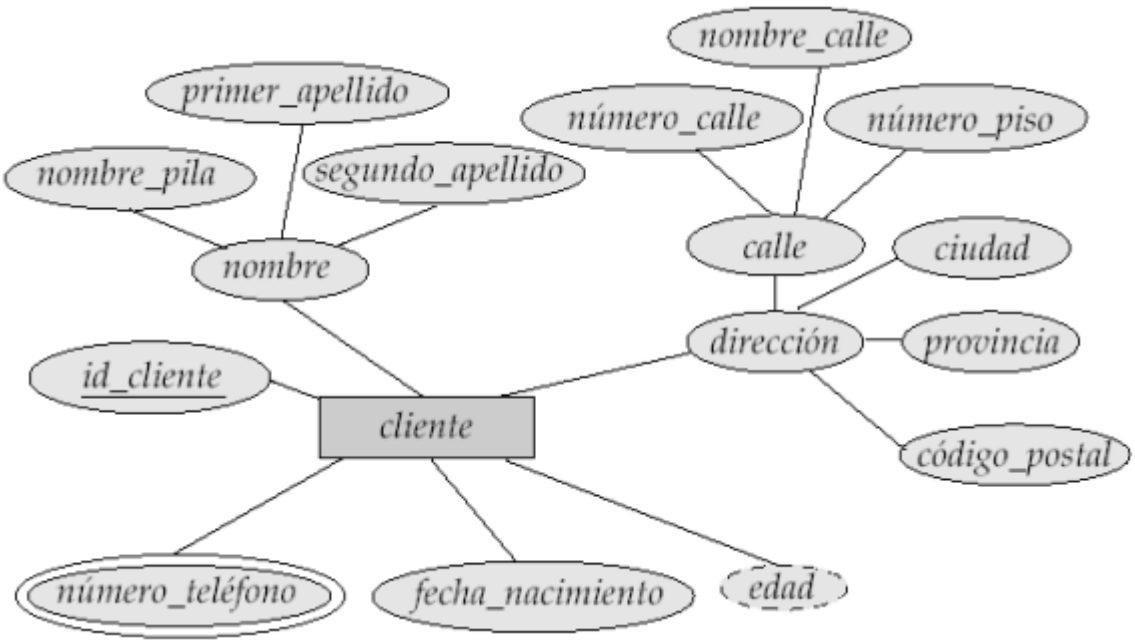
_ La combinación de claves primarias de los conjuntos de entidades participantes constituyen una superclave de un conjunto de relaciones. Por ejemplo (id_cliente, número_cuenta) es la superclave de impositor, esto quiere decir que un par de conjuntos de entidades pueden tener, a lo sumo, una relación en un conjunto de relaciones en particular. También por ejemplo si se desea hacer el seguimiento de todos los fecha-acceso a cada cuenta de cada cliente, no se puede asumir una relación para cada acceso. Aunque se puede utilizar un atributo multivalorado. Cuando se decida cuáles son las claves candidatas se deberá considerar la correspondencia de cardinalidad. En el caso de que exista más de una clave candidata, al seleccionar la clave primaria se deberá tener en cuenta la semántica del conjunto de relaciones. (esto lo vamos a ver nuevamente más adelante).

Diagramas E-R

_ Nosotros podemos empezar a dibujar diagramas.

- Los rectángulos representan conjuntos de entidades. Los rectángulos dobles, representa entidades débiles.
- Los rombos representan conjuntos de relaciones.
- Las líneas enlazan atributos con conjuntos de entidades, y éstos con conjuntos de relaciones.
- Las elipses representan atributos. Donde las elipses dobles representan atributos multivalorados y las elipses discontinuas denotan atributos derivados.
- El subrayado indica atributos clave primarios, como veremos más adelante.

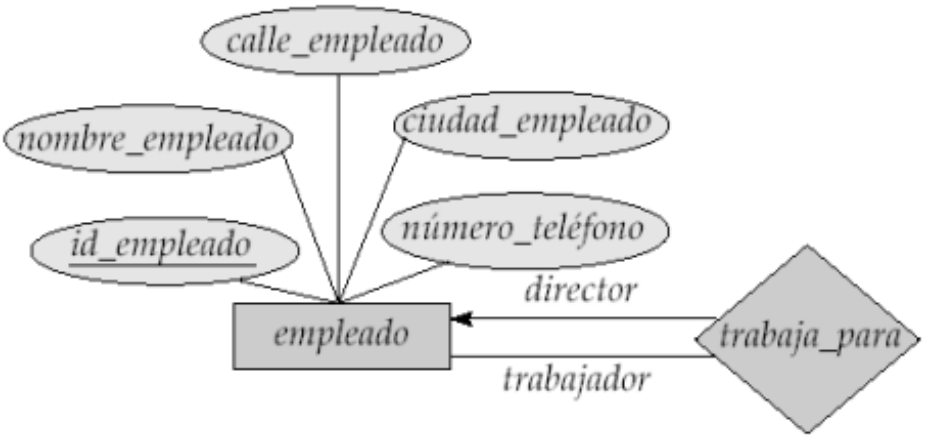
_ Con dibujos vamos modelando la información que el especialista del negocio nos contó y que vamos a necesitar tener en la base de datos para poder almacenar toda la información, para no olvidarnos nada y que no estén duplicados ningunos de los atributos. Según el grafico a continuación, tenemos una entidad que se llama cliente con una clave que lo identifica llamada Id_cliente, además tiene atributos como nombre, en donde es un atributo compuesto, también tiene una dirección, también compuesto, formado por otros atributos simples y compuestos a su vez como el atributo calle. Además el cliente tiene un atributo edad que es un atributo derivado de la fecha de nacimiento, y por último vemos que tiene un atributo número de teléfono que es multivalorado, lo que significa que puede tener más de un número de teléfono. Como vemos acá, edad, es un atributo derivado que va cambiando, porque depende de la fecha actual y la de nacimiento, en donde si sabemos esas dos podemos calcularla. Entonces no es un atributo directo sino que viene de otro.



_ Como vemos acá, edad, es un atributo derivado que va cambiando, porque depende de la fecha actual y la de nacimiento, en donde si sabemos esas dos podemos calcularla. Entonces no es un atributo directo sino que viene de otro.

Papeles

_ Tenemos una entidad empleados, donde algunos empleados trabajan para un determinado director, entonces hay una relación que se llama “trabaja para” que vincula a un empleado con otro empleado. En el vínculo, la relación conecta dos veces con la misma entidad, pero en dos sentidos, “trabaja para” y “es dirigido por”. En el caso de que tengamos situaciones como estas deberíamos explicar lo que es el papel, donde la raya si va para un lado es trabajador y cuando va para el otro es director. Si tenemos más de una raya deberíamos explicar el significado de cada una, y a eso se le llama papel de la relación.



_ Los conjuntos de entidades de una relación necesitan no ser distintivas. Las etiquetas “director” y “trabajador” se denominan papeles; especifican cómo las entidades de empleados interactúan a través del conjunto de relaciones trabaja-para. Los papeles se indican en diagramas E-R etiquetando las líneas que conectan rombos con rectángulos. Las etiquetas de papeles son opcionales y se utilizan para aclarar la semántica de la relación.

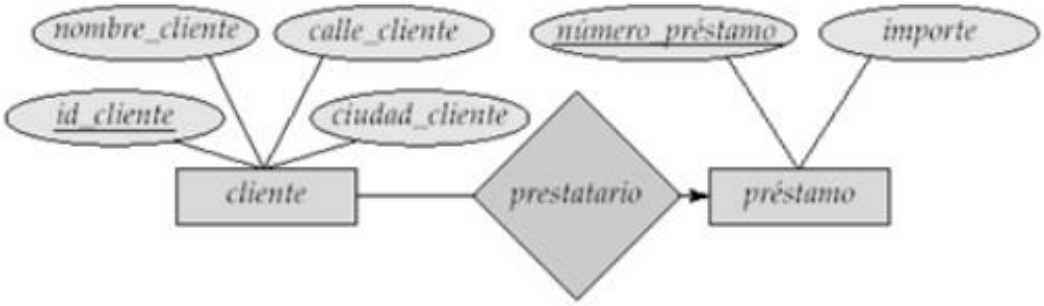
Restricciones de cardinalidad

_ Las flechitas o no flechitas son una forma de representar la cardinalidad. Las restricciones de cardinalidad se expresan o bien dibujando una línea directa (→), cuyo significado es uno, o bien con una línea indirecta (—), cuyo significado es varios, entre el conjunto de relaciones y el conjunto de entidades. En distintos modelos existen distintas formas de ver estos dibujos.

_ Como vemos en el ejemplo, las flechas nos indican que un cliente tiene un préstamo o un cliente no puede tener muchos préstamos, representando de esta manera una relación uno a uno:

_ Ahora vemos el caso en el que un cliente puede tener muchos prestamos, pero a un préstamo lo tiene solamente un cliente, representando de esta manera una relación uno a varios:

_ También puede ser que un cliente tenga un solo préstamo, y un mismo préstamo puede ser entregado a varios clientes, representando de esta manera una relación varios a uno:

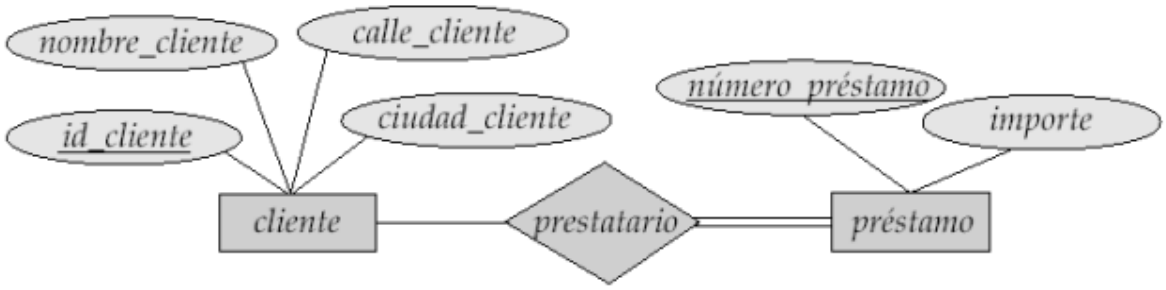


_ Y en esta otra un cliente puede tener muchos préstamos y un préstamo puede haber sido otorgado a muchos clientes, representando de esta manera una relación varios a varios:

Participación de un conjunto de entidades en un conjunto de relaciones

Participación total: (indicada con doble línea) cada entidad del conjunto de entidades participa en, al menos, una relación del conjunto de relaciones. Por ejemplo, la participación de préstamo en prestatario es total, ya que cada préstamo debe tener un cliente asociado a él a través de prestatario, es decir, los prestamos tienen si o si por lo menos un cliente por lo que la participación de préstamo es total.

Participación parcial: puede ser que algunas entidades no participen en ninguna relación del conjunto de relaciones. Por ejemplo, la participación de cliente en prestatario es parcial, es decir, puede haber clientes que no tengan ningún préstamo por lo que sería participación parcial.



Notación alternativa de los límites de cardinalidad

_ Esto también se suele representar con números. Los límites de cardinalidad también pueden expresar restricciones de participación. Donde el cero representa participación parcial, en donde puede ser, en este caso, que el cliente tenga hasta cero prestamos, el uno representa participación total y en este caso tenemos que el préstamo tiene que tener por lo menos un cliente, y el asterisco representa que el cliente puede tener muchos préstamos. En este caso no tenemos flechas y son los números los que vinculan participación y cardinalidad.

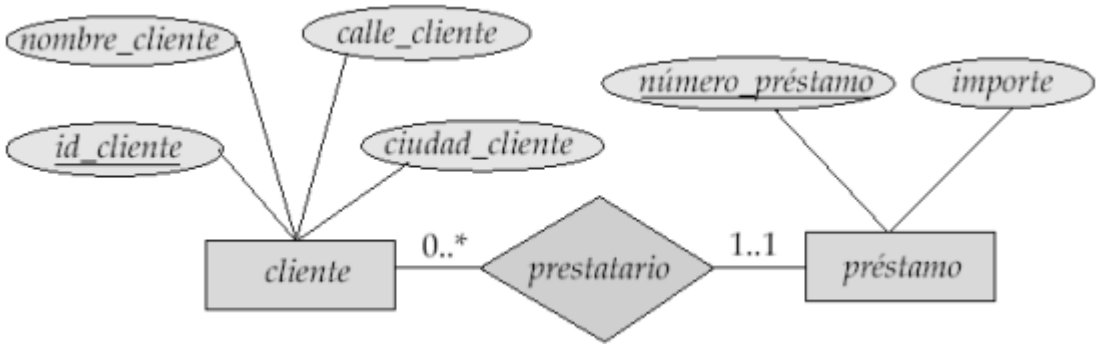
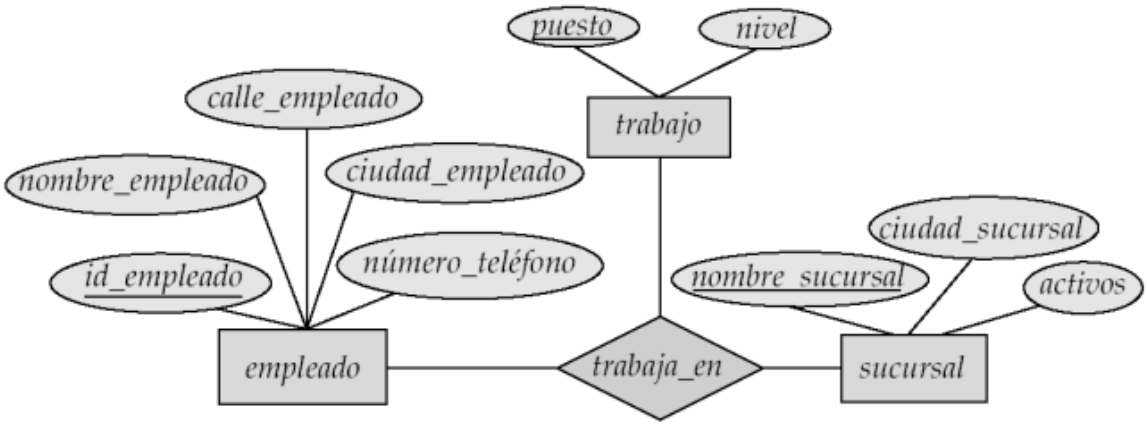


Diagrama E-R con una relación ternaria

_ Cuando tenemos una relación de tipo ternaria, como por ejemplo la siguiente, que dice que un empleado trabaja en una determinada sucursal haciendo un determinado trabajo, donde todo sucede en la misma relación. La persona podría tener distintos roles en distintos lugares de trabajo. Donde la relación “trabaja en” tendría como clave que empleado, en que sucursal y que puesto de trabajo tiene ese empleado.



_ Cuando una relación es ternaria expresar los vínculos muchas veces se vuelve bastante complejo. Por eso vamos a tratar de favorecer relaciones binarias antes que ternarias.

Tenemos algunos procesos que nos permiten transformar relaciones ternarias en varias relaciones binarias.

Restricciones de cardinalidad en relaciones ternarias: se permite, como máximo, una flecha fuera de relación ternaria (o de mayor grado) para indicar una restricción de cardinalidad. Por ejemplo una flecha desde trabaja-en a trabajo indica que cada empleado trabaja, a lo sumo, en un trabajo en cada rama. Si existe más de una flecha, hay dos formas de definir el significado. Por ejemplo una relación ternaria R entre A, B y C con flechas hacia B y C puede significar que cada entidad A está asociada con una única entidad de B y C, o que cada par de entidades de (A, B) está asociado con una única entidad C, y cada par (A, C) está asociado con un único B. Cada alternativa se ha utilizado en formalismos diferentes λ Para impedir que se produzcan confusiones, se declara ilegal la utilización de más de una flecha.

Cuestiones de diseño

_ Acá aparecen algunos criterios que nosotros como diseñadores podríamos tener diferentes miradas sobre los mismos. No todas las personas van a diseñar el modelo de la misma forma, si bien hay reglas para cumplir que son obligatorias, como por ejemplo los criterios para elegir las claves primarias ya que se tienen ciertas consideraciones pero podemos elegir o diseñarla y en ese momento decide el diseñador. Por ende tenemos en cuenta las siguientes:

Utilización de los conjuntos de entidades frente a atributos: la elección entre si debería ser un conjunto de entidades o un atributo depende principalmente de la estructura de la empresa cuyo modelo se está diseñando, y de la semántica asociada con el atributo en cuestión. Por ejemplo, el teléfono podría ser un conjunto de entidades, si quiero guardar tipo, ubicación, etc. Hay cuestiones por ejemplo si algún atributo, por alguna característica de este se convierte en una entidad, como el caso del envió en mercado libre, en la que se le da una jerarquía por sus atributos convirtiéndose en una entidad con sus propiedades.

Utilización de los conjuntos de entidades frente a los conjuntos de relaciones: una posible pauta es la de designar un conjunto de relaciones para describir una acción que se produce entre las entidades. Algo que parece un conjunto de relaciones se puede convertir en un conjunto de entidades, y esta es una decisión abierta.

Conjuntos de relaciones binarias frente a n-arias: aunque no es posible sustituir un conjunto de relaciones no binarias (n-aria, para n > 2) por un determinado número de conjuntos de relaciones binarias distintas, un conjunto de relaciones n-arias muestra con más claridad que varias entidades participan en un relación sola relación. Convertir relaciones n-arias en varias relaciones binarias también es un criterio abierto, se recomienda hacerlo pero podemos optar por elegir una relación de tipo ternaria por algún criterio.

Ubicación de los atributos de las relaciones: la razón de cardinalidad de una relación puede afectar a la ubicación de sus atributos. Por lo tanto los atributos de los conjuntos de relaciones uno a uno o uno a varios puede estar asociados con uno de los conjuntos de entidades participantes, en lugar de con el conjunto de relaciones. La ubicación de los atributos si van en una relación o en una entidad también va a ser un criterio que va a depender en muchos casos si la relación es uno a uno o uno a varios, pero también son cuestiones de diseño que el diseñador puede tomar para la resolución.

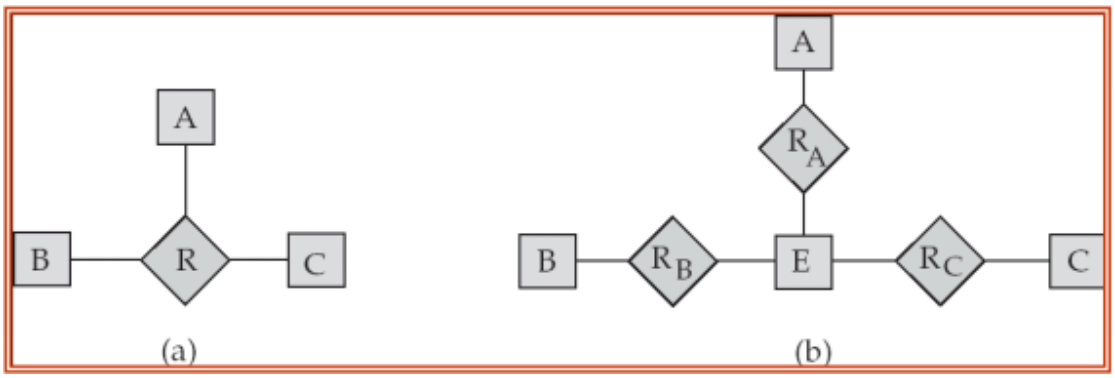
Relaciones binarias frente a no binarias

_ Algunas relaciones que parecía que eran no binarias se pueden representar mejor utilizando relaciones binarias. Por ejemplo, una relación ternaria padres, que asocia a un niño con su padre y madre, se sustituye mejor por las dos relaciones binarias padre y madre. La utilización de dos relaciones binarias permite la información parcial (por ejemplo si sólo se conoce a la madre o al revés). Pero existen algunas relaciones que, naturalmente, no son binarias por ejemplo “trabaja en”. Si la relación es ternaria podría ser que sucedan cosas que no vinculen a la persona con el padre o la madre en ambos casos. La relación ternaria vuelve las cosas un poco más compleja que si fueran todas relaciones binarias. Pero todo depende del problema a solucionar.

Conversión de relaciones no binarias en binarias: en general, cualquier relación no binaria se puede representar utilizando relaciones binarias creando un conjunto artificial de entidades. Sustituir R entre los conjuntos de entidades A, B y C por el conjunto de entidades E, y tres conjuntos de relaciones:

- R_A , relacionando E y A
- R_B , relacionando E y B
- R_C, relacionando E y C

_ Creamos un atributo de identificación especial para E. Luego añadimos cualquier atributo de R a E. Por cada relación (a_i , b_i , c_i) en R, creamos una entidad nueva e_i en el conjunto de entidades E, añadimos (e_i , a_i) a R_A, (e_i , b_i) a R_B, y (e_i , c_i) a R_C. Entonces A, B y C se relacionan con E a través de las relaciones R_A, R_B, y R_C, habiendo logrado crear tres relaciones binarias.



_ También es necesario traducir las restricciones. Puede que no sea posible traducir todas las restricciones. Puede que existan instancias del esquema traducido que no se pueden corresponder con ninguna instancia de R. por ejemplo, ejercicio: añadir restricciones a las relaciones R_A, R_B y R_C para asegurar que la nueva entidad creada se corresponde exactamente con una entidad de cada uno de los conjuntos de entidades A, B y C. No se puede impedir la creación de un atributo de identificación al hacer de E un conjunto de entidades débil (descrito con brevedad) identificado por los tres conjuntos de relaciones.

Las correspondencias de cardinalidades afectan al diseño E-R

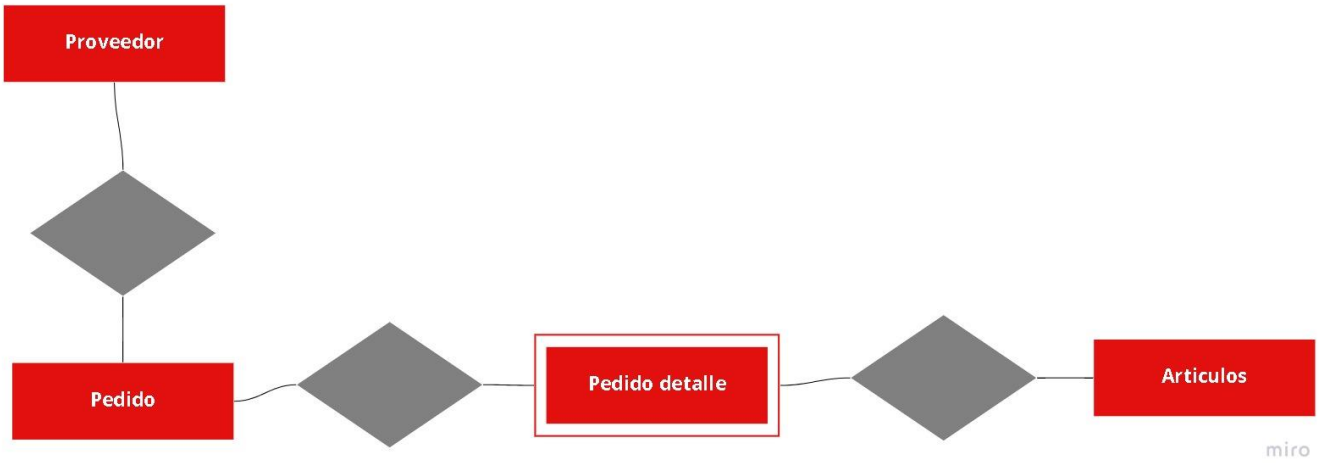
_ Algunas cuestiones de cardinalidad que también van a partir del diseño, como por ejemplo cuando que sucede si tenemos una fecha, y tenemos que elegir entre si la ponemos como atributo en la relación o en la cuenta. Si tenemos por ejemplo que Gonzales abrió una cuenta bancaria, ahora nos preguntamos si la fecha de apertura de la cuenta es un atributo de la cuenta bancaria o es un atributo de la relación sobre qué día abrió la cuenta Gonzales. Entonces, si la relación es uno a muchos se recomienda que el atributo este en la entidad que es una, si la relación es varios a uno se recomienda que este en la entidad que es uno, si la relación es uno a uno puede estar en cualquiera de las dos, y si la relación es muchos a muchos el atributo se debería poner en la relación y no en las entidades independientes.

Conjuntos de entidades débiles

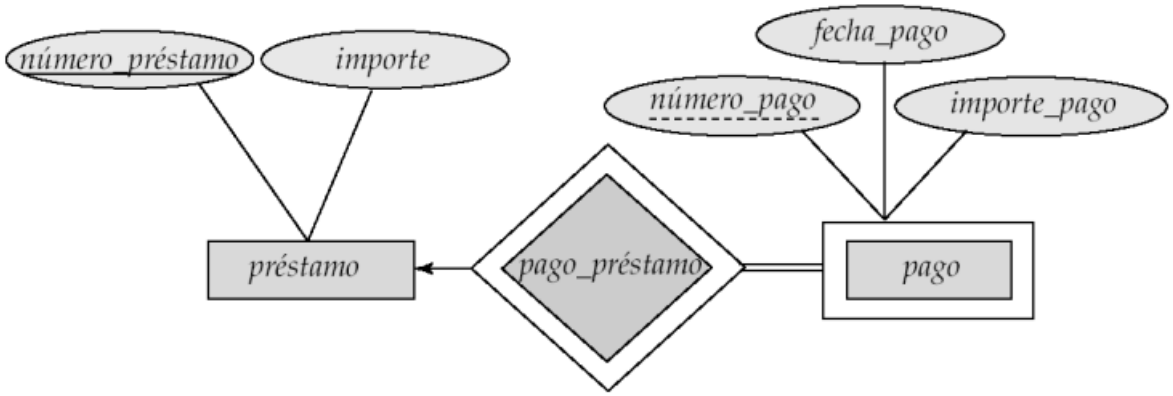
_ Existen un conjunto de entidades, que no tienen un identificador propio o no tienen una clave que los logre identificar, que se llaman entidades débiles. La entidad débil depende de alguna otra identidad o conjunto de identidades que la identifique. De otra forma decimos que a los conjuntos de entidades que no tienen una clave primaria se denominan conjuntos de entidades débiles. La existencia de un conjunto de entidades débiles depende de la existencia de un conjunto de identidades identificadoras. Se debe asociar al conjunto de entidades identificadoras a través de un conjunto de relaciones uno a varios total desde la identificadora al conjunto de entidades débiles. Un caso específico seria por ejemplo un proveedor vendía mercadería, y la envía en un determinado comprobante o pedido, entonces tenemos el proveedor y tengo un numero de pedido que dice que tal proveedor me envió tal pedido de mercadería, y en ese pedido existen varios productos que el proveedor me ha enviado. La relación entre pedido y mercadería es uno a muchos, ahora los artículos que están en el pedido detalle no tienen una identificación propia respecto del pedido que los identifique de forma independiente, sino que dependen del discriminador pedido, y a estas se las denomina un conjunto de entidades débiles.

_ Dicho de otra forma tenemos un proveedor envía un determinado pedido, ese pedido tiene un conjunto de artículos, y esos artículos poseen sus características. Proveedor, pedido y articulo son entidades fuertes ya que cada uno es identificado con su clave, pero pedido detalle es una entidad débil, en donde estas no existen por sí mismas, ya que el

detalle del pedido no puede existir si no existiera un pedido, ya que si no existiera un pedido no podríamos detallar que mercadería enviaron. Para dibujar una entidad débil en el modelo hacemos un rectángulo doble. La entidad débil no tiene una clave primaria por sí misma, sino que esta se conforma con el número de pedio más un discriminador del detalle.



_ La relación identificadora se representa utilizando un rombo doble. El discriminador (o clave parcial) de un conjunto de entidades débil es el conjunto de atributos que lo distinguen entre todas las entidades de un conjunto de entidades débiles. La clave primaria de un conjunto de entidades débiles se forma con la clave primaria del conjunto de entidades fuertes del que depende la existencia del conjunto de entidades débiles, más el discriminador de dicho conjunto de entidades débiles. Un conjunto de entidades débiles se representa por medio de rectángulos dobles. El discriminador de un conjunto de entidades débiles se subraya con una línea discontinua. La clave primaria del conjunto de entidades fuertes no se almacena explícitamente junto con el conjunto de entidades débiles, ya que está implícita en la relación identificadora.



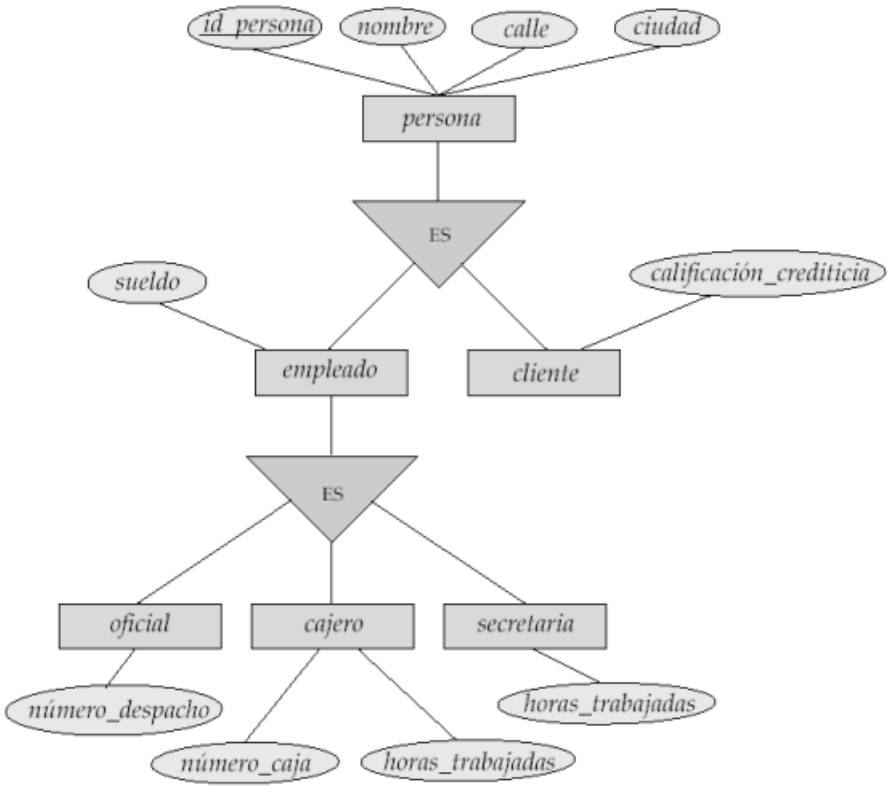
Características del modelo E-R

_ Generalización y especialización son el mismo concepto visto desde abajo hacia arriba y desde arriba hacia abajo.

Especialización:

- Proceso de diseño descendente: se designan subgrupos dentro de un conjunto de entidades que se diferencian de alguna forma del resto de las entidades del conjunto.
- Estos subgrupos se convierten en conjuntos de entidades de nivel inferior que tienen atributos o participan en relaciones que no son aplicables al conjunto de entidades de nivel superior.
- Se representan por medio de un componente triángulo que contiene el nombre del conjunto de entidades (p. e. cliente “es una” persona).
- Herencia de atributos: un conjunto de entidades de nivel más bajo hereda todos los atributos y la participación en las relaciones del conjunto de entidades de nivel superior al cual está enlazado.

_ Por ejemplo, tenemos una entidad que se llama persona con todos sus atributos, y la persona puede tener dos características, podría ser empleado de la empresa o podría ser cliente de la empresa, en donde si es empleado, vamos a tener todos los atributos de la persona más el sueldo del empleado, en cambio sí es cliente, la persona tiene la misma información más un atributo extra del cliente que es la calificación crediticia. Entonces, si es empleado, aparte de cobrar un sueldo, tiene distintos roles, en donde esto se llama especialización y hace que de este diagrama entidad las personas cumplan con ciertas características. La decisión de que esto sea una especialización o no es criterio del diseñador.



Generalización:

- Proceso de diseño ascendente: combina un determinado número de conjuntos de entidades que comparten las mismas características en un conjunto de entidades de nivel superior.
- La especialización y la generalización son inversiones simples una de la otra y se representan en un diagrama E-R del mismo modo.
- Ambos términos, especialización y generalización, son perfectamente intercambiables.

Restricciones de diseño sobre la especialización/generalización

_ Se restringe si las entidades pueden ser miembros de un conjunto dado de entidades de nivel inferior. Esto puede ser definido por:

- Condición: por ejemplo todos los clientes por encima de 65 años son miembros del conjunto de entidades ciudadano-jubilado, en donde ciudadano-jubilado “es una” persona.
- Por el usuario: se especifica.

_ Se restringe si las entidades pueden o no pertenecer a uno o más conjuntos de entidades de nivel inferior dentro de una generalización simple. Tenemos los siguientes casos:

- Disjunto: una entidad sólo puede pertenecer a un único conjunto de entidades de nivel inferior. Se marca en un diagrama E-R escribiendo disjunto al lado del triángulo.
- Solapado: una entidad sólo puede pertenecer a más de un conjunto de entidades de nivel inferior. Para nuestro ejemplo, empleado y cliente son solapado. Pero podrían ser disjuntos. Un auto no puede ser moto o camión.

Restricción de completitud: especifica si una entidad del conjunto de entidades de nivel superior debe pertenecer o no al menos a uno de los conjuntos de entidades del nivel inferior en una generalización. Tenemos los siguientes casos:

- Total : una entidad sólo puede pertenecer a uno de los conjuntos de entidades de nivel inferior.
- Parcial: una entidad no necesita pertenecer a uno de los conjuntos de entidades de nivel inferior.

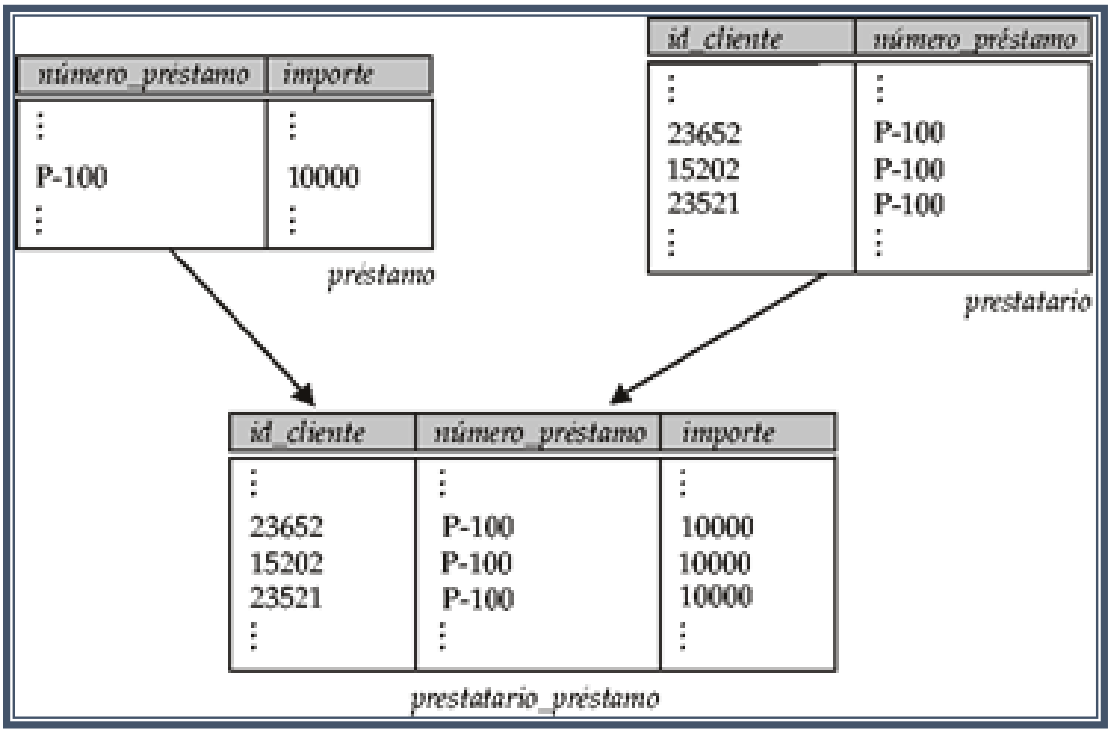
_ Por ejemplo, la generalización de persona es parcial, ya que no es obligatorio que las personas sean clientes o empleados. Pero todo depende de la forma que queramos ver.

Diseño de bases de datos relacionales: Normalización

_ Nosotros hicimos hasta el otro día un trabajo de escuchar al cliente y haber planteado un modelo de entidad relación, que nos permite hacer una especie de plano o dibujo que represente lo que el cliente nos fue pidiendo. Ese modelo sobre el final del proceso, se plantea que ese diagrama debería convertirse en estructuras de datos relacionales, en tablas donde cada una va a ir conformando parte de estos diagramas. Una entidad se convierte automáticamente en un esquema relacional y cada uno de los atributos que esa entidad tiene van a ser atributos del esquema relacional, y las relaciones tienen también un mecanismo de conversión en esquemas relacionales cumpliendo las restricciones de cardinalidad.

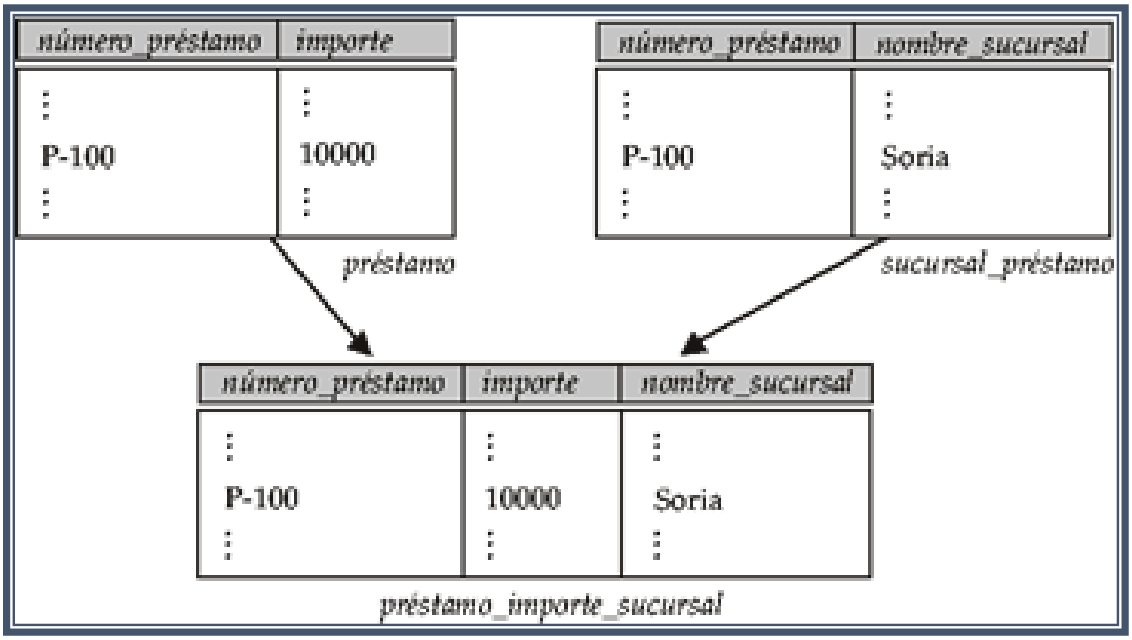
Combinación de esquemas

_ Según la tabla de préstamo tenemos que se le ha otorgado un préstamo a alguien, a ese préstamo se le puso un número, en este caso 100, y el importe fue de 10000. En la tabla de prestatario, tenemos que ese préstamo 100 se le otorgo a tres clientes. A esta información se la expresa en dos tablas distintas, pero también la podemos expresar en una sola tabla. Dependiendo de cómo hayamos establecido el diseño de la tabla, o sea nuestra forma de expresar los vínculos en el esquema relacional entre los atributos, nos puede generar dudas, por ejemplo en este caso, donde se plantea que hay tres clientes que piden un préstamo de 10000 cada uno, no sabemos si el préstamo fue de 10000 pesos para tres personas o si el prestamos que se hizo fue de 30000 pesos. La información de la tabla nos puede generar dudas debido a que es información no clara o no segura.



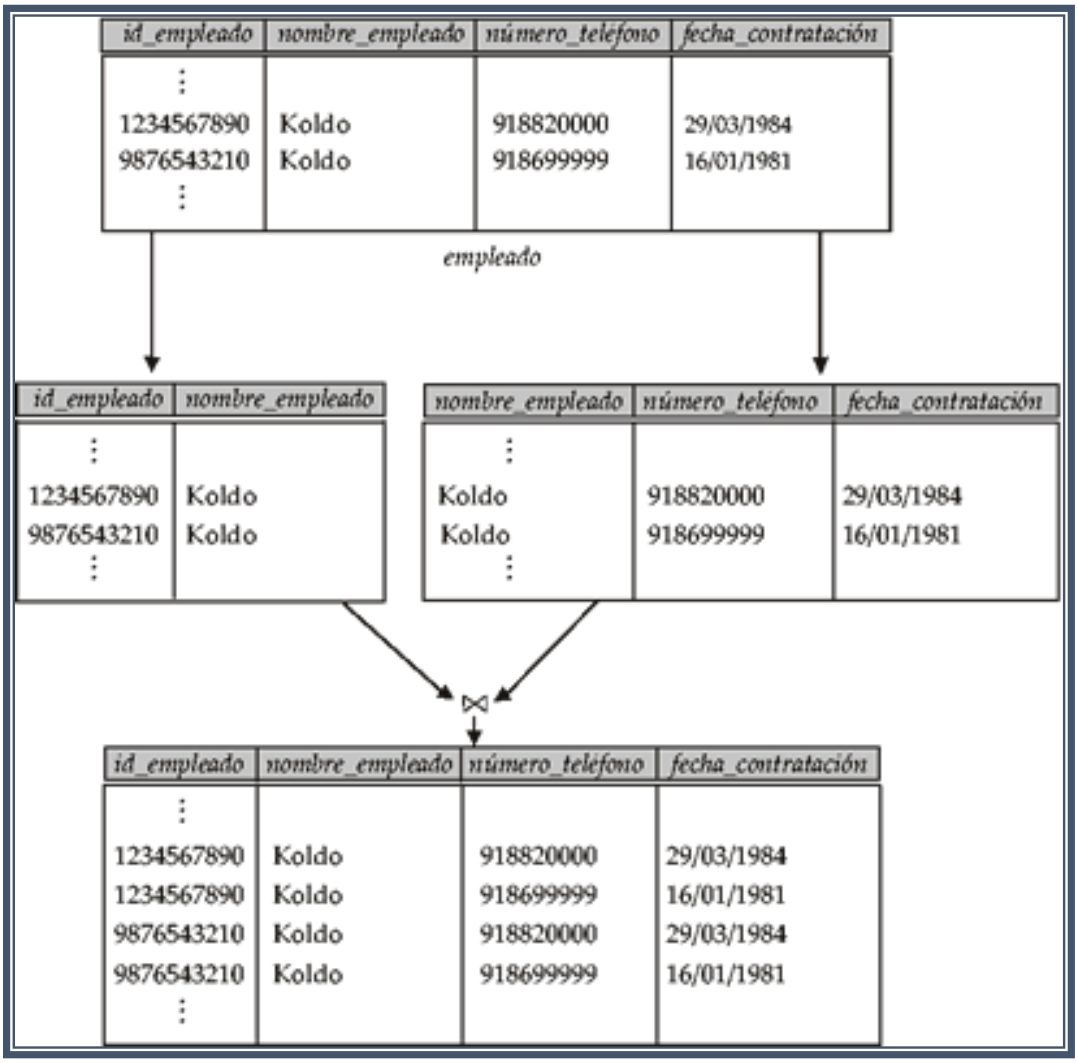
_ El problema que tenemos es que como nos aseguramos que lo que estamos montando en nuestros esquemas relacionales representa realmente el problema que estamos teniendo.

_ Podemos tener el caso en el que de vuelta el préstamo numero 100 es un préstamo que se otorgó por 10000 pesos, y por otro lado en otra tabla tenemos que el préstamo 100 se otorgó en una sucursal que se llama Soria. En este caso, en la nueva tabla, no repetimos atributos, y pareciera que las cosas están más claras:



Dependencias funcionales: ahora suponemos que se ha empezado con prestatario_préstamo. Pensamos en cómo se podría dividir o descomponer en prestatario y préstamo. Para esto vamos a tomar un concepto que se denomina dependencia funcional, en donde nosotros deberíamos poder representar nuestra relación entre los datos a través de este concepto. Entonces, vamos a decir que hay ciertos atributos que dependen funcionalmente de otros atributos y nosotros deberíamos detectar estas dependencias funcionales. Por ejemplo, la expresión número_préstamo $\rightarrow$ importe, indica que el importe de un préstamo depende funcionalmente del número de préstamo, en donde si sabemos el número de préstamo podemos garantizar cual es el importe que se le presto al cliente en ese préstamo, porque el importe depende funcionalmente del número de préstamo. Esto nos va a ayudar a darnos cuenta cuando es bueno descomponer y cuando las descomposiciones no son buenas, cuando separamos en dos esquemas relacionales y nos equivocamos, o cuando unimos en dos esquemas relacionales y nos equivocamos o acertamos.

_ A continuación tenemos un ejemplo de mala descomposición. Tenemos algunos datos del empleado, luego descomponemos la información principal en dos tablas como vemos a continuación, y sucede que la descomposición es incorrecta porque si queremos volver a construir o a unificar las dos relaciones en una sola ya que nos genera una confusión sobre a quien le pertenecen los datos, ya que al tener dos empleados que se llamen igual, al unificar las tablas, ya no sabemos a cuál de los dos empleados les pertenecen sus datos:



_ Esta mala descomposición sucede ya que está vinculada a que atributos dependen funcionalmente de quien y también de quien no dependen funcionalmente.

_ Ahora vemos, por ejemplo, la información de todas las clases que vamos a tener en el semestre, pero si observamos bien hay información que esta inconsistente por la forma en la que esta puesta en la planilla, es decir, no da certeza de lo que estamos leyendo, como por ejemplo los parciales teóricos con las clases teóricas que parece ser que están en el mismo día, por otro lado vemos que hay también tres evaluaciones que se indican

con el número tres y no se sabe porque o para qué es. O sea estamos sacando deducciones de los datos que tenemos en la planilla tratando de descifrar cosas, pero que no son claras en el diseño de la base de datos. Nosotros necesitamos que el diseño de la base de datos tenga consistencia y certeza de la información. Entonces, nosotros nos vamos a valer de las formas normales y de las dependencias funcionales para podernos asegurar de hacer un buen diseño de la base de datos.

_ Hace 30 años no existía el concepto de formas normales, y cuando una persona se ponía a diseñar una base de datos, el diseño que hacía se decía que era medio artístico, dependía de la capacidad y de la inspiración de esa persona que la diseñaba, donde todo quedaba a criterio del diseñador. Al pasar el tiempo se generaron reglas de normalización llamadas primera, segunda, tercera, cuarta y quinta forma normal, en donde se verificaba que el diseño que se realizaba cumpliera con estas reglas, y de ser asi el diseño es de buena calidad. Luego surgió la forma normal de Boyce Codd que esta entre la forma normal tres y la cuatro, y que luego siguieron perfeccionando su esquema. Al pasar los años se obtuvo el esquema con la 1 FN (forma normal), la FN de Boyce Codd, 3 FN, 4 FN, 5 FN, 6 FN, 7FN, y estas nos hacen que el modelo nos garantice el funcionamiento y no se cometan errores.

_ Las formas normal es junto con las dependencias funcionales nos van a permitir hacer procesos que nos garanticen que el proceso que hagamos sea de buena calidad.

Primera forma normal

_ Una vez hacemos el diagrama entidad-relación, lo convertimos en un esquema relacional y ahora revisamos si este último cumple con las condiciones que tiene que cumplir.

_ La primera forma normal nos dice que un dominio es atómico si se considera que los elementos del dominio son unidades indivisibles. Por ejemplo, algunos dominios no atómicos pueden ser conjuntos de nombres, apellidos, dirección, es decir, atributos compuestos. Entonces, todos los atributos del esquema relacional tienen que ser atómicos, y un dominio es atómico si no lo puedo dividir en dos dominios.

_ La regla dice que un esquema relacional R cualquiera está en primera forma normal si los dominios de todos los atributos de R son atómicos. Esta es la primera regla que debemos cumplir en el diseño de nuestra base de datos, y nos dice que todos los atributos de un esquema relacional cualquiera tiene que tener dominios que sean atómicos, que no se puedan dividir en dos, y si se pueden dividir tenemos un problema que resolver. Los valores no atómicos complican el almacenamiento de datos redundantes (repetidos) de almacenamiento y estimulación. La atomicidad es realmente una propiedad de cómo se utilizan los elementos del dominio. Las cadenas de caracteres se considerarían normalmente indivisibles, ahora si ahí adentro está el nombre 1 y nombre 2, ahí debemos

dividirlo en dos partes y separar cada atributo. Por ejemplo, tenemos un atributo compuesto direcciones formado por calle, barrio, número y calle, en el diagrama entidad-relación podemos dejar la dirección como atributo compuesto ya que simplemente es un dibujo, pero al momento de hacer el diagrama de clases debemos colocar los cuatro atributos por separado y no el atributo dirección, ya que de lo contrario no estaríamos cumpliendo con la primera forma.

_ El atributo clave de la UCC no es atómico, ya que los dos primeros dígitos representan el año en el que entramos a la facultad y los otros números representan que alumno somos dentro de ese año. Lo correcto sería que este fuese atómico, pero la razón por la que esto es asi es porque en el momento en que se diseñó la base de datos de la facultad fue hace más de 30 años y en ese entonces no existían las formas normales, lo cual este mal diseño nos lleva a varios problemas ya que tenemos un atributo compuesto por dos datos, en donde uno de los problemas que lleva esto es que hay un desperdicio de claves, entre otros.

_ Entonces, mantener el atributo todo unido es una mala codificación de la información, y esa estructura de datos no está cumpliendo con la primera forma normal, por ende debemos chequear el diseño de nuestro esquema relacional y verificar que están en la primera forma normal, y para eso verificamos atributo por atributo y ver que todos sean atómicos. Para el caso de los atributos multivalorados, esto tampoco estaría permitido.

Objetivo: revisamos si un esquema relacional en particular R es una forma “buena”, revisando cada una de las formas normales desde la uno en adelante. En el caso de que una relación R no esté en una forma normal “buena”, la acción que vamos a hacer es descomponer R en un conjunto de relaciones {R1 , R2 , ..., Rn } de forma que cada una de esas relaciones estén en una forma normal “buena” y de que la descomposición es una descomposición de reunión sin pérdida, o sea si juntamos R1, R2, Rn de nuevo para reconstruir R deberíamos tener toda la información sin tener perdida. Esta teoría se basa en dependencias funcionales y dependencias multivaloradas.

Dependencias funcionales

_ Las dependencias funcionales son restricciones legales del conjunto de relaciones, es decir, del modelo. Las dependencias funcionales no las inventamos, sino que vienen dadas de información que yo detecto en el mundo real. Con esas relaciones legales yo voy a hacer cumplir ciertas reglas para asegurarme de que el modelo este bien diseñado. Las dependencias funcionales requieren que el valor de un cierto conjunto de atributos determine únicamente el valor para otro conjunto de atributos, o sea determinados atributos determinan a determinados atributos, o a uno, o varios a uno, etc. El concepto de dependencia funcional se parece al concepto clave.

_ Sea R el esquema relacional de una relación $\alpha \subseteq R$ y $\beta \subseteq R$, en donde existe una dependencia funcional que dice que $\alpha \rightarrow \beta$ que se cumple para la relación R sí y sólo si en cualquier relación legal r(R), para todos los pares de tuplas t1 y t2 de r que concuerdan con los atributos α, también concuerdan con los atributos β, es decir, t1 [α] = t2 [α] $\Rightarrow$ t1 [β] = t2 [β]. No pueden aparecer dos α iguales con dos β diferentes. Esta es la condición que establecen las dependencias funcionales. Si tenemos dos filas de la relación, y si en algunas filas se repiten los atributos alfa y se repiten en otras, se tienen que repetir los atributos de beta en ambas tuplas, pero si eso no sucede no se cumple entonces la regla de dependencia funcional. Entonces, suponemos que tenemos la relación R con dos atributos A y B, donde A toma los valores 1, 1 y 3 y B toma los valores 4, 5 y 7, y ahora decimos que existen dos tuplas donde alfa que vale 1 y beta vale 4, y aparte hay otra tupla donde alfa vale 1 y beta vale 5, y esto significa que A$\rightarrow$B no se cumple, es decir, A no determina funcionalmente a B porque para esto 1 en A debería valer 4 y repetirse asi en la segunda tupla pero B vale 5, pero si podemos decir que B$\rightarrow$A, es decir, que B define funcionalmente a A ya que 4 determina a 1, 5 determina a 1 y 7 determina a 3, por lo que no hay duda a quien determina B aunque se repitan los valores en A.

1	4
1	5
3	7

_ Si el valor alfa en dos tuplas, el valor beta tiene que ser igual, pero no sería lo mismo a la inversa. Si decimos alfa tenemos que asegurar una única respuesta de beta, pero si dudamos la respuesta de beta, no hay dependencia funcional. Si pensamos en el mundo real, clave de la UCC determina funcionalmente a apellido y nombre, puede haber dos claves UCC que tengan el mismo nombre y apellido, no hay problema porque clave de la UCC sigue determinando apellido y nombre, pero lo que no funciona es al revés porque si decimos un nombre que a la vez esta repetido, ahí entramos en duda a que clave nos referimos. Tenemos como ejemplo una función Y=f(X) en donde decimos que Y depende funcionalmente de X ya que al decir cuánto vale X sabemos cuánto vale Y, entonces X$\rightarrow$Y.

_ Recordamos que la detección de las dependencias funcionales viene del mundo real y no de mirar los valores que tenemos en la base de datos. Podemos encontrar dependencias funcionales entre atributos, aunque no sean claves primarias, pero las dependencias funcionales es un concepto más amplio que las claves, por ejemplo podríamos decir que la edad de una persona depende funcionalmente de su fecha de nacimiento, entonces hay atributos que dependen funcionalmente de otros atributos aunque no sean claves. Cuando nosotros estamos ante un problema del mundo real, nosotros tenemos que detectar cuales son las dependencias funcionales que hay, entonces la tarea es escuchar al cliente, hacemos el diagrama E-R, y definimos la lista de las dependencias funcionales y

anotarlas, donde una vez hecho esto recién ahí vamos a empezar a convalidar un buen diseño de base de datos, si no descubrimos las dependencias funcionales estamos en problemas.

_ A modo de ejercicio analizamos la siguiente tabla y tenemos que por ejemplo:

- C3 no determina funcionalmente a C4, ya que para un mismo valor repetido de C3 hay diferentes valores de C4.
- C6 si determina funcionalmente a C7.
- C7 no determina funcionalmente a C6, ya que para un mismo valor repetido de C7 hay diferentes valores de C6.
- C6 si determina funcionalmente a C5.
- C1 y C2 si determina funcionalmente a C3, C4, C5, C6 y C7 porque no se repiten nunca.

C1	C2	C3	C4	C5	C6	C7
FAC	1	12/4	JUAN	CBA	L1	100
FAC	2	12/4	ESTELA	BSAS	L2	200
FAC	3	12/4	BETO	BSAS	L3	300
FAC	4	12/4	BETO	CBA	L9	400
NCR	1	15/4	JUAN	MZA	L5	300
NCR	2	15/4	OMAR	CBA	L1	100
NDE	1	15/4	JUAN	CBA	L1	100
NDE	2	15/4	JUAN	CBA	L1	100
NDE	3	19/4	VICTOR	BSAS	L4	100
FAR	1	20/4	DELIA	MZA	L6	200
FAR	2	20/4	BETO	MZA	L7	200

_ Ahora suponemos que K es una superclave para el esquema de relación R si y sólo si $K \rightarrow R$ (si K determina todos los atributos de R), entonces K es una clave candidata (la más chica de las superclaves) para R si y sólo si $K \rightarrow R$, y para ningún $\alpha \subset K$, $\alpha \rightarrow R$. Las dependencias funcionales nos permiten expresar restricciones que no se pueden expresar utilizando superclaves. Tomando como ejemplo el caso del importe, tenemos que se espera que número-préstamo$\rightarrow$importe, pero no se espera que se cumpla número-préstamo$\rightarrow$nombre-cliente, ya que el número de préstamo nos dice cuanto es el importe del mismo, pero no nos puede decir el nombre del cliente porque los clientes son varios.

_ Las claves determinan funcionalmente a todos los atributos, en cambio la dependencia funcional puede determinar algunos atributos.

Uso de las dependencias funcionales: se utilizan las dependencias funcionales para comprobar las relaciones que acabamos de definir, y ver si son legales según un conjunto dado de dependencias funcionales. Entonces detectamos las dependencias funcionales y verificamos si la relación r es legal según el conjunto F de dependencias funcionales, y si sucede esto, se dice que r satisface a un conjunto F de dependencias funcionales. También las dependencias funcionales se utilizan para especificar las restricciones del conjunto de

relaciones legales, entonces se dice que F se cumple en R si todas las relaciones legales de R satisfacen el conjunto de dependencias funcionales F.

Dependencia funcional trivial: estamos diciendo algo obvio y sin sentido donde A determina a A, y no tiene gracia la definición. Por ejemplo:

- nombre_cliente, número_préstamo $\rightarrow$ nombre_cliente
- nombre_cliente $\rightarrow$ nombre_cliente

Transitividad: dado un conjunto F de dependencias funcionales, hay otras ciertas dependencias funcionales que están implicadas lógicamente por F, por ejemplo si A$\rightarrow$B y B$\rightarrow$C, entonces se puede inferir que A$\rightarrow$C como consecuencia de las dependencias anteriores.

_ Tanto el concepto de transitividad como el de trivialidad son conceptos que nos harina definir un montón de dependencias al vicio, por ende las vamos a evitar estas dependencias.

Forma normal de Boyce-Codd

_ Un esquema de relación R está en FNBC respecto a un conjunto de dependencias funcionales F si, para todas las dependencias funcionales de F+ (cierre del conjunto de las dependencias funcionales) de la forma $\alpha \rightarrow \beta$ donde $\alpha \subseteq$ R y $\beta \subseteq$ R, se cumple al menos una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es trivial (es decir $\beta \subseteq \alpha$).
- α es una superclave de R.

_ Lo que nos propone Boyce-Codd es que, tenemos el modelo E-R, donde escribimos todas las entidades y todas las relaciones, en donde de ese paquete de entidad-relaciones aparecieron un montón de esquemas relacionales, y ahora tenemos que busquemos todas las dependencias funcionales que se puedan y luego verificamos que para cada una de las relaciones R se debe cumplir con que si hay una dependencia funcional de la forma $\alpha \rightarrow \beta$ (alfa determina a beta), entonces o es trivial o tiene que ser superclave, ahora si no se cumple con estas condiciones entonces no están en la forma normal de Boyce-Codd por ende debemos romper para que se cumpla. Si el diseño de la base de datos no cumple con la forma normal de Boyce-Codd significa que no es de buena calidad, y puede haber redundancia en la información o en los datos, además de traer problemas.

_ Entonces, una vez que encontramos todas las dependencias funcionales, debemos determinar si o son triviales o superclaves. Si no se cumplen estas condiciones, entonces no está en FNBC. A modo de ejercicio analizamos nuevamente el cuadro de la primera forma normal, que representa un esquema relacional R, en donde como primera medida determinamos las siguientes dependencias funcionales:

- C6→C5, C7
- C1, C2→C3, C4, C5, C6, C7

_ Una vez que tenemos las dependencias funcionales, tenemos que determinar que sean o triviales o superclave. Entonces, la dependencia funcional de C1 y C2 por un lado no es trivial, pero si es superclave ya que determinan a todos los atributos. Ahora con la dependencia de C6 tenemos que no es trivial ni tampoco superclave, por lo tanto esta relación no cumple con la FNBC y por ende debemos resolverlo.

- C6→C5, C7 (no trivial y no superclave)
- C1, C2→C3, C4, C5, C6, C7 (no trivial pero si superclave)

_ Cuando se hace referencia a F+, hablamos de que Boyce-Codd insiste en que tenemos que encontrar todas las dependencias funcionales y verificar que esto sea asi, ya que si no encontramos todas, esto no va a funcionar. _ Podemos considerar la siguiente dependencia funcional, ya que C6 más cualquier otro atributo termina determinando el resto, pero por más que sigamos escribiendo estas dependencias funcionales redundantes, el resultado final va a ser lo mismo que las relaciones R1 y R2.

- C6, C3→C5, C7
- C6, C3→C5, C7
- Etc.

_ Con lo cual Boyce-Codd nos dice que no es necesario buscar todas las dependencias funcionales redundantes porque el resultado es el mismo. Pero a su vez Boyce-Codd nos dice que si queremos podemos buscar todas las dependencias funcionales que derivan de una sola como en este caso, y él nos garantiza de que vamos a obtener el mismo resultado que obtuvimos con la primer dependencia funcional que encontramos, con lo cual no es necesario que probemos o corroboremos con todas. Por ende esto funciona para todas las dependencias funcionales.

<u>Descomposición de un esquema en FNBC:</u> entonces si encontramos un alfa que determina beta y que a la vez no es trivial y no es superclave, entonces la dependencia funcional $\alpha \rightarrow \beta$ nos está trayendo un problema, entonces para resolver esto dividimos R en dos relaciones:

- R1 = (α U β). Es decir, la primer relación R1 se arma con todos los atributos α unión todos los atributos β.
- R2 = (R - (β - α)). Y la segunda relación R2 se arma con todos los atributos que tenga R menos, β menos α.

_ Continuando con el ejemplo anterior vamos a romper el esquema relacional R, transformándola en dos relaciones R1 y R2. Una de las relaciones, R1, va a tener todos los atributos de α U β, es decir, una tabla con C6, C5 y C7 que forman la relación. Como

vemos, C5, C6 y C7 tienen tuplas repetidas, y a estas las deberíamos anular. Ahora podemos observar que C6 es la superclave de la relación R1.

C6	C5	C7
L1	CBA	100
L2	BSAS	200
L3	BSAS	300
L9	CBA	400
L5	MZA	300
L4	BSAS	100
L6	MZA	200
L7	MZA	200

_ Ahora para construir R2, siguiendo la teoría tenemos que restar β - α:

- C5, C7 - C6 = C5, C7

_ Luego tenemos que restar todos los atributos de R a C5, C7:

- C1, C2, C3, C4, C5, C6, C7 - C5, C7 = C1, C2, C3, C4, C6

_ Por lo que C6 va a vincular las dos tablas. Entonces tenemos que la relación R2 se conforma de los siguientes atributos:

C1	C2	C3	C4	C6
FAC	1	12/4	JUAN	L1
FAC	2	12/4	ESTELA	L2
FAC	3	12/4	BETO	L3
FAC	4	12/4	BETO	L9
NCR	1	15/4	JUAN	L5
NCR	2	15/4	OMAR	L1
NDE	1	15/4	JUAN	L1
NDE	2	15/4	JUAN	L1
NDE	3	19/4	VICTOR	L4
FAR	1	20/4	DELIA	L6
FAR	2	20/4	BETO	L7

_ Entonces R1 está en FNBC ya que C6 es la superclave y la relación R2 también está en FNBC porque C1 y C2 siguen siendo las superclaves. Por otro lado podemos volver a unir y establecer el vínculo o reconstruir la relación original R, pero ahora no tenemos datos redundantes ni repetidos. Por ende la FNBC nos ayudó a romper el esquema relacional para llevarlo a dos esquemas relacionales R1 y R2, pero de mejor calidad que el esquema relacional original R que era de mala calidad y con datos redundantes.

_ Es muy importante saber que ninguna corrección del modelo se acepta o se resuelve por intuición, sino que se resuelven ejecutando la descomposición de la forma en la que explica Boyce-Codd.

_ Siempre nuestra tarea va a ser como primera medida tomar un esquema relacional y revisamos si son todos atributos atómicos, y de ser asi significa que está en primera forma normal, en el caso de que no lo sean tenemos que romper el esquema para lograr la primera forma normal. Luego escribimos las dependencias funcionales, y revisamos que todas estas sean triviales o superclaves, en donde en el caso de ser asi entonces todo está en FNBC, caso contrario, tenemos que romper como explica Boyce-Codd hasta que todo se cumpla y este en FNBC.

Conservación de dependencias: las restricciones, incluyendo las dependencias funcionales, son muy costosas de comprobar excepto si pertenecen a una única relación. Si es suficiente comprobar sólo aquellas dependencias de la relaciones individuales de una descomposición para asegurar que se cumplen todas las dependencias funcionales, entonces dicha descomposición preserva las dependencias. Como no es siempre posible conseguir tanto FNBC y conservación de las dependencias, se considera una forma normal más débil, denominada, tercera forma normal.

_ La FNBC no resuelve todos los problemas, para eso existen otras formas normales como vimos antes, en la que dadas ciertas condiciones o reglas se debe modificar o romper el esquema relacional para llegar a un resultado mejor. Todos los modelos de normalización funcionan de la misma manera.

Almacenamiento y estructura de archivos

_ Este capítulo nos va a dar dos nociones, uno es de cómo se guarda la información de la base de datos en el disco duro, o sea como funcionan los lugares físicos de almacenamiento de información, y por otro lado es ver como se organizan o estructuran los archivos, tanto las llaves como los datos, dentro de los discos.

Visión general de los medios físicos de almacenamiento

Clasificación de los medios físicos de almacenamiento

_ Tenemos distintos medios físicos de almacenamiento. Estos nos van a dar diferentes velocidades de almacenamiento o velocidades a la que se puede acceder a los datos. También tenemos en cuanto a coste por unidad de datos. Y también en cuanto a fiabilidad, es decir, en cuanto a pérdida de datos por fallos del suministro eléctrico o caídas del sistema o fallos físicos de los dispositivos de almacenamiento.

_ El almacenamiento se puede dividir en:

Almacenamiento volátil: el contenido de la memoria se pierde cuando se corta el suministro eléctrico, como por ejemplo la memoria RAM.

Almacenamiento no volátil: el contenido se conserva cuando se corta el suministro eléctrico. Incluye almacenamiento secundario y terciario, así como memoria principal con batería de salvaguarda. Por ejemplo disco duro, DVD, pendrive.

Medios físicos de almacenamiento

Memoria cache: es la forma más rápida y costosa de almacenamiento, esta memoria es volátil, y está gestionada por el hardware del sistema informático. No es gestionada por los motores de base de datos.

Memoria principal de la computadora: tiene un acceso rápido (de 10 a 100 nanosegundos; 1 nanosegundo = 10–9 segundos). Generalmente demasiado pequeña (o demasiado cara) para almacenar toda la base de datos, tiene capacidades de hasta unos pocos Gigabytes, ampliamente usada en la actualidad, pero las capacidades han aumentado y los costes por byte han disminuido de forma constante y rápida (aproximadamente un factor 2 cada 2 o 3 años). Esta es de tipo volátil, donde los contenidos de la memoria principal generalmente se pierden si se produce un fallo en el suministro eléctrico o una caída del sistema. Las claves primarias conviene almacenarlas en la RAM porque son mucho más fáciles de buscarlas que en el disco duro.

Memoria flash: en esta los datos superan los fallos del suministro eléctrico. Se pueden escribir datos en una posición sólo una vez, pero la posición puede ser borrada y grabada de nuevo, donde sólo puede soportar un número limitado de ciclos (10K - 1M) de escritura/borrado, y el borrado de la memoria ha de hacerse sobre una banco entero de memoria. Las lecturas son aproximadamente tan rápidas como en memoria principal, y aunque las escrituras son lentas (unos pocos microsegundos), borrar es más lento. El coste por unidad de almacenamiento es aproximadamente igual al de la memoria principal. Esta es usada ampliamente en dispositivos incorporados, tales como cámaras digitales. También es conocida como EEPROM (Memoria de lectura y borrado programable eléctricamente).

Disco magnético: los datos se almacenan sobre disco giratorios y son leídos/grabados magnéticamente. Estos son el soporte primario para el almacenamiento de datos a largo plazo; generalmente almacena toda la base de datos. Para los accesos se deben mover los datos desde disco a memoria principal y para el almacenamiento se han de volver a escribir. El acceso es mucho más lento que en memoria principal (más sobre ello posteriormente), y el acceso directo es adecuado para las lecturas de datos en disco en cualquier orden, a diferencia de la cinta magnética. Es de tipo no volátil.

- Rango de capacidades: de hasta aproximadamente 700 GB en la actualidad 4. Mucha más capacidad y mejor coste por byte que en memoria principal / memoria flash. Esto está creciendo constante y rápidamente, con perfeccionamiento tecnológico (un factor de 2 a 3 cada 2 años)

_ En resumen tenemos los discos de estado sólido y productos más nuevos, donde también podemos guardar información que no se pierde al momento del apagado de los equipos o el corte del suministro eléctrico. Un fallo de disco puede destruir datos, pero es muy raro.

Almacenamiento óptico: estos son del tipo no volátil, donde los datos se leen ópticamente, por medio de un láser, desde un disco giratorio. Los formatos más populares son CD-ROM (640 MB) y DVD (4.7 hasta 17 GB). Tenemos algunos discos ópticos de escritura única y lectura múltiple, empleados para el almacenamiento de archivos (CD-R, DVD-R y DVD+R), y también están disponibles versiones de escritura múltiple (CD-RW, DVD-RW, DVD+RW y DVD-RAM). Las lecturas y escrituras son más lentas que con discos magnéticos.

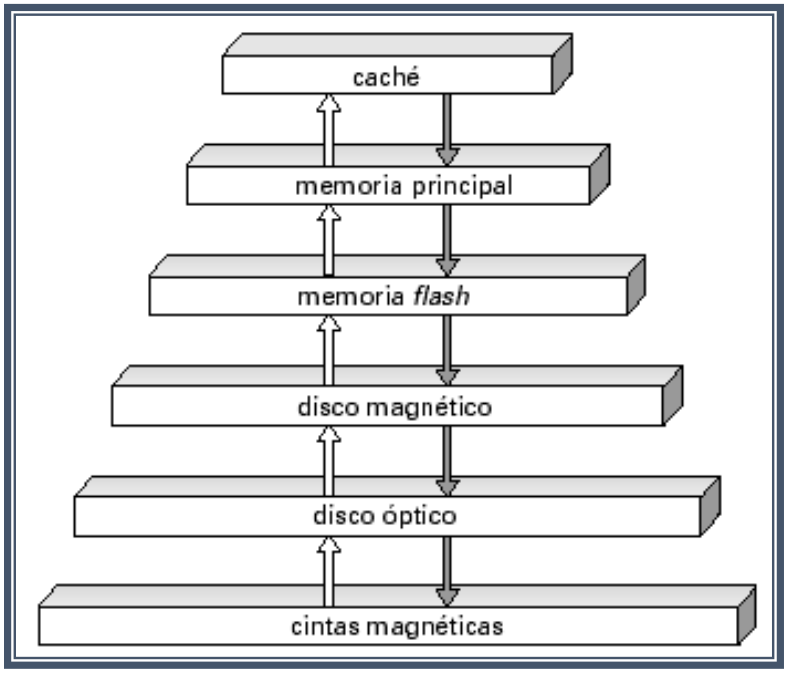
_ Tenemos el sistemas de Juke-box (sistema de paquete de disco), con gran número de discos duros removibles, unos cuantos lectores y un mecanismo para la carga/descarga automática de discos, para el almacenamiento de grandes volúmenes de datos y donde podemos grabar en cualquiera de estos discos, siendo la herramienta quien administra donde grabar y guardar los datos.

Almacenamiento en cinta: también es del tipo no volátil, empleado principalmente para copias de seguridad (para la recuperación de un fallo de disco) y para datos de archivo. Estos están obsoletos. El acceso secuencial es mucho más lento que los discos, pero tienen una capacidad muy alta (cintas disponibles de 40 a 300 GB). El coste de almacenamiento es más barato que el disco pero las unidades de cinta son caras.

_ Cambiadores de cintas disponible para el almacenamiento de cantidades masivas de datos, desde cientos de terabytes (1 terabyte = 109 bytes) hasta incluso un petabyte (1 petabyte = 1012 bytes).

Jerarquía de almacenamiento

_ Las herramientas por su jerarquía o velocidad se organizan como podemos ver a continuación, en donde cuanto más arriba mayor costo y mayor velocidad, cuanto más abajo menor costo y menor velocidad.



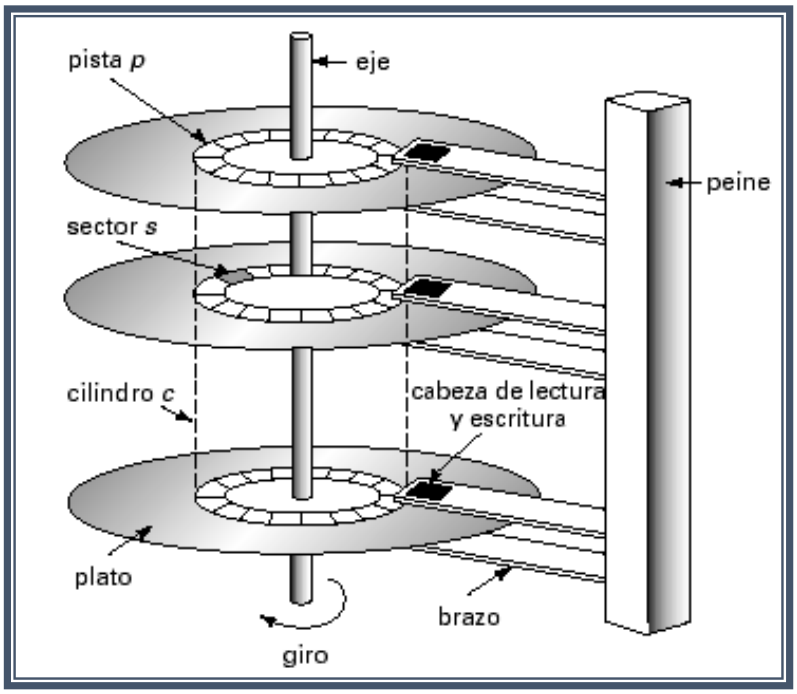
Almacenamiento principal: es el medio más rápido, pero volátil (caché, memoria principal).

Almacenamiento secundario: siguiente nivel jerárquico, no volátil, tiempos de acceso moderadamente rápidos. También llamado almacenamiento en conexión. En este tenemos por ejemplo, memoria flash, discos magnéticos.

Almacenamiento terciario: nivel jerárquico más bajo, no volátil, tiempo de acceso lento. También llamado almacenamiento sin conexión. Tenemos por ejemplo, cintas magnéticas, almacenamiento óptico.

Mecanismos de discos rígidos magnéticos

_ En el siguiente diagrama mostramos que, muchas veces el grabado de la información de la base de datos nos lleva a pensar de que los datos no están guardados en una pista del disco, sino que muchas veces lo que se hace en la estructuración del grabado de la base de datos, es que cuando nosotros grabamos todos los registros, para ganar tiempo el cabezal entra y puede grabar en una pista (pista p) por ejemplo clave del alumno, nombre, apellido, teléfono, y quizás en otro sector (sector s) se podría grabar dirección, edad, fecha de nacimiento, etc, y asi sucesivamente.

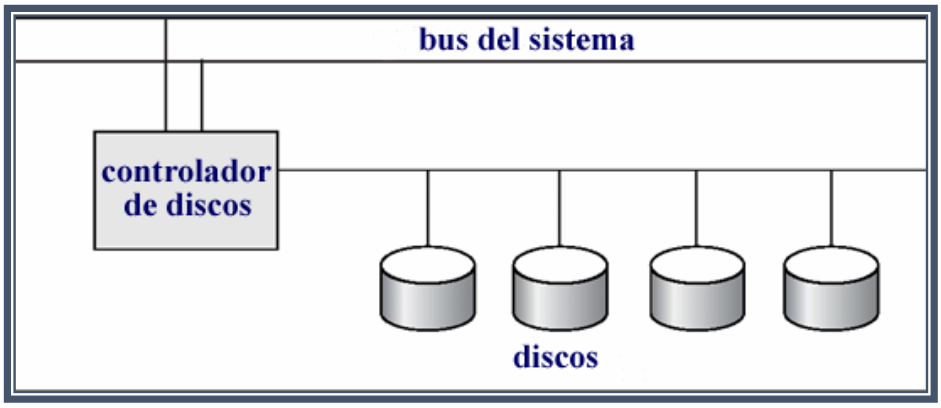


_ Un registro de datos de la base, probablemente en distintos sectores pegados de la misma zona del disco duro, sino que este en distintos platos del disco y en distintos pedazos. Se graba en todas las pistas en el mismo punto, y en el momento de leer el mismo acceso levantaría todo el registro o buena parte del registro pero no buscándolo uno al lado del otro.

_ Lo que se busca optimizar son los tiempos de grabación y de lectura que tengo adaptados al modelo físico (velocidad, potencia, características) del disco que usamos en ese momento. La información físicamente de la base de datos, no están los datos pegados uno al lado del otro, sino que están redistribuidos dentro del disco duro en la posiciones que más le favorezcan al hardware para poder accederlas de la manera más rápida posible, aprovechando las virtudes y las características del disco en particular con el que estamos trabajando.

Subsistema de discos

_ Hay una cuestión que se da en muchos sistemas gestores de bases de datos, en donde estos tienen como objetivo poder manipular grandes volúmenes de información. Entonces quien va a gestionar una base de datos tiene que prever disco duros o lugares de almacenamiento con la capacidad disponible para esta gran cantidad de datos o información. Entonces los discos tienen que tener ciertas características específicas para poder ser capaces de almacenar una gran cantidad de información.



_ Vemos múltiples discos conectados a un sistema informático por medio de un controlado. La funcionalidad de los controladores (comprobación de suma, reasignación de sectores defectuosos) frecuentemente es llevada a cabo mediante discos individuales; reduce la carga sobre el controlador.

_ Tenemos algunos estándares de interfaz de discos, respecto a diferentes velocidades y capacidades, como:

- ATA (Adaptador AT)
- SATA (Serial ATA)
- SCSI (Interconexión de pequeños sistemas informáticos)

RAID (Arrays redundantes de discos independientes)

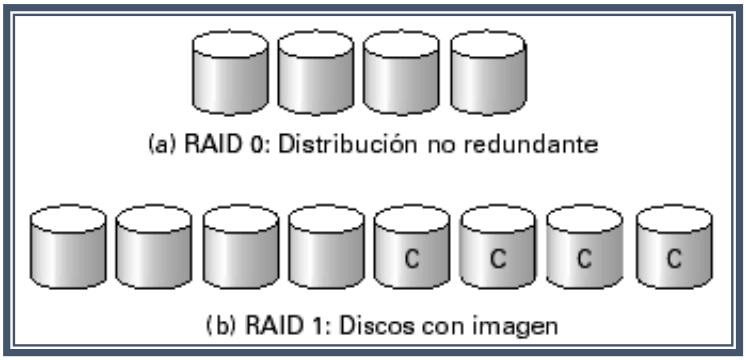
_ Los discos RAID son discos redundantes con discos independientes a los cuales puedo grabar y obtener respuesta de los mismos de manera independiente. Supongamos que tenemos que guardar la base de datos en nuestro disco, en donde la base de datos tiene que tener un nivel de confiabilidad importante, ya que necesitamos que el servidor de la base de datos responda siempre de manera absolutamente confiable. Ante la criticidad con la que deseo operar deberían ser las características del disco o servidores que busque para ser confiables las 24hrs. O sea, buscamos confiabilidad en los datos que estamos guardando en la base de datos, buscamos seguridad de poderlos acceder en el momento que los necesite, y que funcionen siempre las 24hrs del día los 365 días del año. Los sistemas RAID son sistemas que se han ido creando para garantizar esta manera de

guardar la información, es decir, la confiabilidad de la información. Estos buscan dar confiabilidad y alta eficiencia, además de la fiabilidad y redundancia. Los RAID son técnicas de organización de discos que gestionan un gran número de discos, aportando la visión de uno solo de alta capacidad y alta velocidad mediante el uso de múltiples discos en paralelo y alta fiabilidad por el almacenamiento redundante de datos, para que se puedan recuperar incluso si falla un disco. Además son técnicas del empleo de redundancia para evitar que la pérdida de datos sea crítica con gran número de discos. Cuando nuestra base de datos es muy crítica, tendremos que invertir en estos discos para ganar mejor tiempo de respuesta o para ganar una mayor confiabilidad de que los discos nunca se cambien.

Niveles de RAID

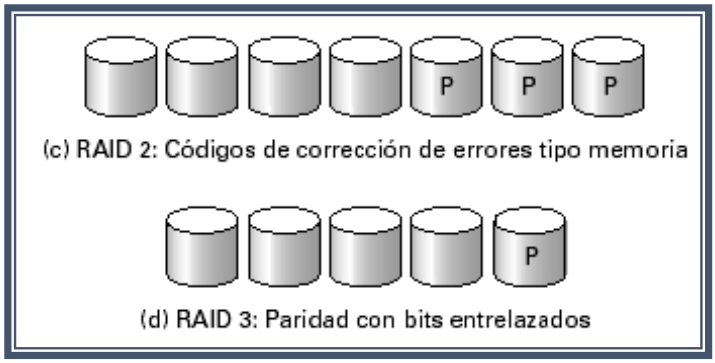
RAID de nivel 0: distribución de bloques; sin redundancia. Se utiliza en aplicaciones de alto rendimiento en las que no es crítica la pérdida de datos. Por ejemplo, me compro 4 discos y no duplico nada, y si se rompe alguno no hay solución, y básicamente es como no hacer redundancia de nada.

RAID de nivel 1: imágenes de discos con distribución de bloques. Ofrece el mejor rendimiento de escritura. Es popular en aplicaciones como el almacenamiento de archivos de registros históricos en un sistema de bases de datos. Por ejemplo, en vez de comprar 4 compramos 8, donde 4 tienen la información y los otros 4 tienen un duplicado de la misma información, entonces cuando grabamos los dos discos a la vez (original y duplicado), cuando mandamos a leer podemos leer en cualquiera de los dos, entonces perdimos la mitad de la capacidad que teníamos, es más caro, pero mejoramos el riesgo.



RAID de nivel 2: organización de códigos de corrección de errores tipo memoria (ECC) con distribución de bit. Por ejemplo, de 7 discos duros, en 4 grabamos la información y en los otros 3 no grabamos toda la información sino que ponemos lo que se llaman dígitos de paridad en el que si alguna de la información de los discos grabados se rompiera, los dígitos de paridad mediante alguna fórmula matemática me permiten reconstruir la información, con la ventaja de que no gastamos otros 4 discos sino que ahorramos uno. Asi logramos una mejor eficiencia, seguimos teniendo redundancia, y es más barata.

RAID de nivel 3: (paridad con bits entrelazados). Un solo bit de paridad es suficiente para la corrección de errores, no sólo detección, dado que se sabe el disco que ha fallado 4 Cuando se graban datos, los bits de paridad correspondientes se deben calcular y escribir sobre un disco de bit de paridad 4 Para recuperar datos en un disco dañado, calcular XOR de bits desde otros discos (incluyendo el disco de bits de paridad). O sea, si compramos por ejemplo 5 discos nada más, vamos a guardar los datos en 4 y el quinto disco es donde ponemos toda la paridad con otras características o descripciones de paridad, quizás no tan fuerte como la del RAID 2, pero si mas barato.



(c) RAID 2: Códigos de corrección de errores tipo memoria

(d) RAID 3: Paridad con bits entrelazados

RAID de nivel 4: paridad con bloques entrelazados; emplea distribución en el nivel de bloque y mantiene un bloque de paridad en un disco independiente para los correspondientes bloques de los otros N discos. Cuando se graban bloques de datos, los bloques bits de paridad correspondientes se deben calcular y escribir sobre un disco de paridad.

(e) RAID 4: Paridad con bloques entrelazados

RAID de nivel 5: paridad distribuida con bloques entrelazados; datos y paridad divididos entre N + 1 discos, en vez de almacenar los datos en N discos y la paridad en 1.. λ. Por ejemplo, con 5 discos el bloque de paridad para el n-ésimo conjunto de bloques se almacena en el disco (n mod 5) + 1, con los bloques de datos almacenados sobre los otros 4 discos.

(f) RAID 5: Paridad distribuida con bloques entrelazados

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

_ Entonces, de esta manera se fueron fabricando distintos tipos de RAID que dan mayor o menor confiabilidad, pero que salen más caros o más baratos. La persona encargada de comprar discos duros para el servidor, debe saber que RAID está comprando y que virtudes tiene el mismo que estamos adquiriendo, en cuanto a que tan caro es, que tan confiable, que tanta seguridad me da, que tantas posibilidades de fallo me da, etc. Por ejemplo, si tenemos una base de datos que cambia constantemente, hacemos un análisis y vemos que el nivel 1 aporta un rendimiento mucho mejor en escritura que el nivel 5 pero el nivel 1 tenía un coste de almacenamiento superior que el nivel 5, sabiendo que el nivel 5 es preferido para aplicaciones con velocidad de actualización baja grandes cantidades de datos y el nivel 1 se prefiere para todas las otras aplicaciones.

Aspectos del hardware

_ El RAID se puede implementar tanto con software como por hardware:

Software RAID: las implantaciones RAID se hacen totalmente en software, sin ningún soporte hardware especial, aunque un poco más lento.

Hardware RAID: son implantaciones RAID con hardware especial. Se emplea RAM no volátil para registrar las escrituras que se están ejecutando. Debemos tener cuidado con fallos en el suministro eléctrico durante la escritura que pueden originar la corrupción del disco.

Intercambio caliente (hot swap): esto es, un disco se rompe en el pack de discos, lo sacamos y lo cambiamos por otro disco sin apagar la computadora, sin dar de baja el servidor, y al cabo de un rato reconstruyo la información de ese disco que estaba dañado con la información que tengo duplicada en los otros discos y el usuario no se dio cuenta de este fallo y cambio. Hace referencia a la sustitución del disco mientras está funcionando, sin cortar el suministro eléctrico.

Acceso al almacenamiento

Bloques: un archivo de base de datos está dividido en unidades de almacenamiento de longitud fija, llamadas bloques. Los bloques son unidades de asignación de almacenamiento y de transferencia de datos. El sistema de bases de datos busca minimizar el número de transferencias de bloques entre el disco y la memoria. Se puede reducir el número de accesos a disco manteniendo en memoria tantos bloques como sea posible.

Memoria intermedia: es una parte de la memoria principal disponible para almacenar copias de bloques del disco.

Gestor de la memoria intermedia: es un subsistema responsable de asignar el espacio de la memoria intermedia en la memoria principal.

Gestor de la memoria intermedia

_ Los programas llaman al gestor de memoria intermedia cuando necesitan un bloque del disco. Si el bloque ya está en la memoria intermedia, a la solicitud del programa se da la dirección del bloque en la memoria principal, ahora si el bloque no está en la memoria intermedia, el gestor de memoria intermedia primero asigna espacio en la memoria intermedia para el bloque reemplazando (desechando) algún otro bloque, si es necesario, para hacer espacio al nuevo bloque, donde el bloque desechado se graba de nuevo a disco, sólo si se modificó desde el momento en que fue grabado o tomado del disco. Y segundo lee el bloque desde el disco a la memoria intermedia y pasa la dirección del bloque en la memoria principal al solicitante.

_ Es decir, por ejemplo cuando se está buscando la nota que se sacó un alumno, vamos al disco duro, buscamos la nota de esa persona, la traemos a la memoria principal RAM, y asi mostramos la nota. Si tuviésemos que mostrar la nota que se sacó otro alumno, es muy probable que al traer todo un bloque de datos, este en este mismo la nota del segundo alumno que buscamos (por ejemplo buscamos Álvarez primero y ahora queremos la de Arnaudo), por lo que el sistema gestor de base de datos lo primero que hace es fijarse en la memoria RAM si esta tiene la nota de Arnaudo, para no tener que ir al disco duro porque pierde tiempo, por ende si la nota está en la memoria RAM se la muestra. Para esto los sistemas gestores de base de datos tienen la habilidad de chequear primero en la memoria principal, antes de chequear en el disco duro, si los datos están en ahí en ese momento ganando milésimas de segundos que se perderían en ir hasta el disco duro.

_ Los índices de los archivos o las claves primarias, por lo general están en la memoria RAM para poder agilizar tiempos de búsqueda en la memoria, y no en el disco duro.

Organización de archivos

_La base de datos está almacenada como un grupo de archivos. Cada archivo es una secuencia de registros. Un registro es una secuencia de campos. Un enfoque puede ser:

- Suponer que el tamaño del registro es fijo.
- Cada archivo tiene sólo registros de un determinado tipo.
- Se usan diferentes archivos para diferentes relaciones. Este caso es el más fácil de implementar; se considerarán registros de longitud variable posteriormente.

Registros de longitud fija: vemos un enfoque sencillo, en donde almacenamos el registro i empezando desde el byte $n * (i - 1)$, donde n es el tamaño de cada registro. El acceso al registro es sencillo, pero los registros pueden atravesar bloques, en donde no se permite a los registros atravesar los límites de un bloque. En el borrado del registro i, tenemos distintas alternativas:

- Mover los registros i + 1, … , n a i, ... , n - 1.
- Mover el registro n a i.
- No mover registros, sino enlazar todos los registros libres sobre una lista libre.

registro 0	C-102	Navacerrada	400
registro 1	C-305	Collado Mediano	350
registro 2	C-215	Becerril	700
registro 3	C-101	Centro	500
registro 4	C-222	Moralzarzal	700
registro 5	C-201	Navacerrada	900
registro 6	C-217	Galapagar	750
registro 7	C-110	Centro	600
registro 8	C-218	Navacerrada	700

Registros de longitud variable: los registros de longitud variable surgen en los sistemas de bases de datos de diferentes maneras:

- Almacenamiento en un archivo de múltiples tipos de registro.
- Tipos de registro que permiten longitudes variables de uno más campos.
- Tipos de registro que permiten campos repetidos (empleados en algunos de los más antiguos modelos de datos).

Modelo entidad-relación: algebra relacional

Estructura básica de base de datos relacionales

_ A continuación tenemos el ejemplo de una relación entre distintos atributos que conforman en si la relación. Tenemos los conjuntos D1 , D2 , ... , D_n, en donde una relación r es un subconjunto de D1 x D2 x … x D_n, con una cantidad de n-tuplas, como se puede observar en la tabla a continuación:

numero_cuenta	***nombre_sucursal***	***saldo***
C-101	Centro	500
C-215	Alta Cordoba	700
C-102	General Paz	400
C-305	Cerro	350
C-201	Recta	900
C-222	Juan B Justo	700
C-217	Urca	750

_ Otro ejemplo seria si:

- nombre_cliente = {Gómez, López, Santos, Pérez}
- calle_cliente = {Arenal, Mayor, Goya}
- ciudad_cliente = {León, Cerceda, Vigo}

_ Entonces podemos formar las siguientes relaciones:

- r = { (Gómez, Arenal, León), (López, Goya, León), (Santos, Arenal, Vigo), (Pérez, Mayor, Vigo) }

_ Como resultado tenemos una relación sobre nombre_cliente x calle_cliente x ciudad_cliente.

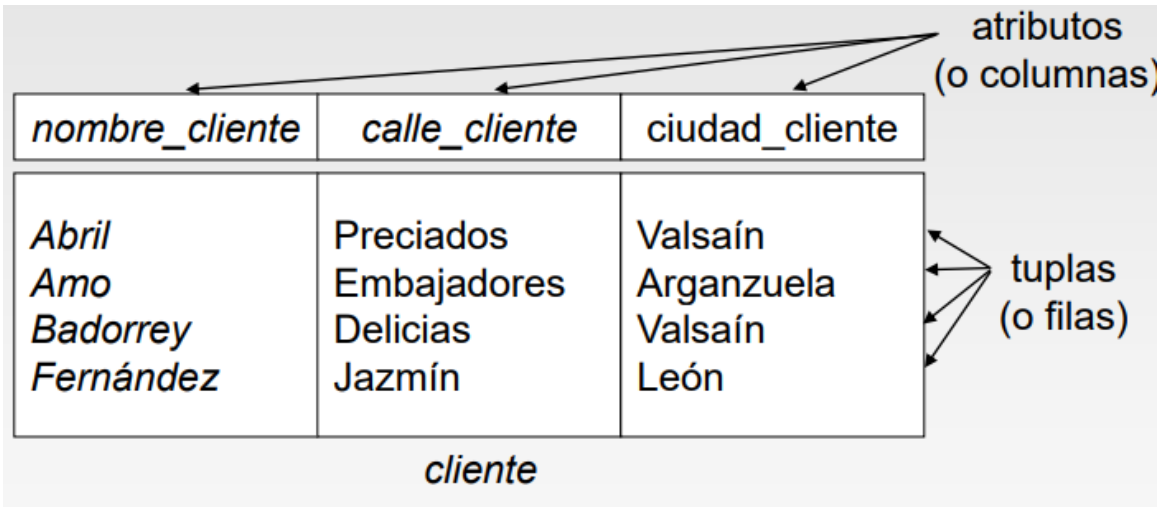
Atributos

_ Una relación contiene ciertos atributos, cada atributo de la relación tiene un nombre, y hay un dominio que son los valores que pueden tomar esos atributos. Normalmente se requiere que los atributos sean atómicos e indivisibles, para que sean más eficientes. Los valores de los atributos multivalorados y de los atributos compuestos son no atómicos.

Valor Null: existe un valor especial que se llama valor null que es miembro de todos los dominios, es decir, para todos los dominios se acepta o pueden aceptar el valor null al menos que nosotros digamos que no está permitido. El dato null significa "no sé" y crea complicaciones en la definición de algunas operaciones, en donde ignoraremos el efecto de los valores null en una primera presentación y luego consideraremos sus efectos con posterioridad. El valor null no es un dato cero, por ejemplo las notas de un alumno pueden ser del 0 al 10 pero también null, ya que podemos no saber que nota se sacó un alumno.

Esquema de la relación: un esquema de relación R es un conjunto de atributos A_n, es decir, R = (A1 , A2 , …, An), en donde esta suma de atributos conforman lo que se llama relación. Por ende, r(R) es una relación en el esquema de la relación R, por ejemplo, cliente (Esquema_cliente).

Instancia de la relación: los valores actuales (instancia de relación) de una relación se especifican en una tabla. Un elemento t de r es una tupla, representada por una fila en una tabla. Es decir tiene los valores de los atributos en un determinado momento. Se denominan atributos o columnas verticalmente, a los atributos que tiene la relación, y se denominan tuplas o filas a cada una de las instancias del esquema relacional. Este esquema relacional tiene ciertos atributos con las características de los mismos, y una instancia con los valores que esos atributos toman en un determinado momento.



Las relaciones no son ordenadas:

_ Las relaciones no son ordenadas, es decir, las tuplas no respetan ningún orden entre una y otra, no siguen una secuencia de orden entre una tupla y la siguiente. Podríamos visualizarlas de manera ordenada. El orden de la tuplas es irrelevante (las tuplas se pueden guardar con un orden arbitrario).

numero_cuenta	***nombre_sucursal***	***saldo***
C-101	Centro	500
C-215	Alta Cordoba	700
C-102	General Paz	400
C-305	Cerro	350
C-201	Recta	900
C-222	Juan B Justo	700
C-217	Urca	750

_ Tomando como ejemplo esta tabla, podemos ordenar las tuplas alfabéticamente según el nombre de sucursal, o de menor a mayor según el saldo, podríamos ordenarlas por número de cuenta, etc. Pero técnicamente las tuplas de una relación no se encuentran ordenadas y tampoco es obligatorio que lo estén. Podemos tener llaves que nos hagan creer que los números de cuenta, por ejemplo, siguen un orden, pero podrían ser llaves adicionales que apuntan a esas tuplas mediante punteros haciéndome creer que están ordenadas por número de cuenta, aunque físicamente los registros o las tuplas siguen estando desordenadas. Entonces siempre pensamos que las tuplas dentro de una relación no están ordenadas por ninguno de los atributos que visualizamos, si se da la casualidad es mera casualidad.

Base de datos: una base de datos consta de múltiples relaciones. Recordamos un gestor de base de datos que era una herramienta que administra múltiples relaciones o tablas con información. La información sobre una empresa se divide en partes, en la que cada relación almacena una parte de la información, y luego existen vínculos entre estas relaciones. En donde todo esto es lo que vamos a denominar conceptualmente la estructura de la base de datos. En esa estructura el buen diseño de la base y la normalización son importantes para el funcionamiento de la misma y asi nos aseguramos de que la información no se repita, evitando tener valores múltiples, entre otras cosas que va a controlar el gestor de la base de datos.

_ A modo de análisis, suponemos la siguiente relación “proveedores”, donde tenemos S# que es el numero o código de proveedor que parece estar ordenado pero es una casualidad, tenemos el nombre del proveedor, la situación o calificación del mismo, y además tenemos la ciudad de donde estos viven. En esta relación parece ser que S# es la clave primaria o el atributo que identifica unívocamente a cada uno de los otros atributos.

S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES
S2	JAIMES	10	PARIS
S3	BERNAL	30	PARIS
S4	CORONA	20	LONDRES
S5	ALDANA	30	ATENAS

_ Por otro lado tenemos otra relación que se llama “partes”, donde P# es el código de partes, tenemos el nombre de la parte, color, peso y la ciudad donde están almacenadas. P# parece ser la clave primaria de esta relación.

P#	PNOMBRE	COLOR	PESO	CIUDAD
P1	TUERCA	ROJO	12	LONDRES
P2	PERNO	VERDE	17	PARIS
P3	BIRLO	AZUL	17	ROMA
P4	BIRLO	ROJO	14	LONDRES
P5	LEVA	AZUL	12	PARIS
P6	ENGRANAJE	ROJO	19	LONDRES

_ Luego tenemos una relación que se llama “envíos”, en donde si hiciéramos un diagrama entidad relación tendríamos las dos entidades proveedores y partes y un rombo que las relaciona y será la relación envíos, cuya clave primaria es la suma de las claves primarias de proveedores y partes. Entonces tendríamos el código de proveedor y el código de partes, y que cantidad de esa parte envió tal proveedor (S# + P#). Nunca podría estar repetido el mismo proveedor enviando la misma cantidad parte.

S#	P#	CANT
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	400
S4	P2	200
S4	P4	300
S4	P5	400

_ Entonces tenemos un diagrama de entidad-relación con proveedores y partes, y la relación envíos que vincula a ambos.

Clave externa: antes definimos todos los conceptos de claves, pero no este. Se denomina clave externa a un atributo que es clave primaria en otra entidad o en otra relación. En el caso del ejemplo anterior S# forma parte de la clave primaria de envíos pero es una clave externa ya que es clave primaria en la relación proveedores, y lo mismo con P# .

Lenguajes de consulta

_ Antes se buscó construir instrucciones de un lenguaje que le permitan al usuario solicitar información de la base de datos. Estos lenguajes que acceden a la base de datos suelen ser de un nivel superior a los lenguajes de programación actuales. Inicialmente los lenguajes de consulta de la base de datos eran un set de instrucciones del mismo lenguaje de programación, pero cuando los diseños de la base de datos se normalizan, ordenan y organizan, surgen conceptualmente la idea de un lenguaje de consultas a la base de datos, con la idea de estandarizar formas de realizar preguntas a una base de datos, de manera tal que, al estar programando en un lenguaje de programación cualquiera, al momento de necesitar preguntarle algo a la base de datos, todos trabajemos en un estándar de preguntas y asi preguntemos por información la base de datos de la misma manera independientemente del lenguaje de programación con el que estemos trabajando.

_ Estos lenguajes de consulta se definen en dos categorías:

Procedimental: cuando el usuario indica al sistema que lleve a cabo una serie de operaciones en la base de datos para obtener el resultado esperado. En estos tenemos que expresar y especificar de donde queremos que el motor saque los datos.

No procedimental: o declarativo, cuando el usuario describe la información deseada sin dar un procedimiento para obtenerla y el mismo lenguaje no procedimental tiene la capacidad de buscar la manera para obtener el dato más allá que el usuario no le diga de donde tiene que sacar la información. Estos son lenguajes más inteligentes donde solo expresamos lo que necesitamos y el motor de base de datos solo va a realizar la búsqueda de lo que le estamos indicando.

_ Por otro lado, a los lenguajes se los define como lenguajes “puros” donde estos son la base de los lenguajes de consulta que utilizan los usuarios, existen tres tipos:

- Álgebra relacional
- Cálculo relacional de tuplas
- Cálculo relacional de dominios

Algebra relacional

_ El algebra relacional es un lenguaje puro, es un algebra para preguntarle cosas a nuestras relaciones. El lenguaje SQL opera preguntándole cosas a la base de datos mediante comandos siguiendo una lógica de algebra relacional. Asi como alguien definió las operaciones suma, resta, división y multiplicación, luego alguien fabrico una calculadora que nos permite hacer cálculos matemáticos siguiendo las reglas del algebra de la matemática sabiendo que significa cada operador, ahora en el caso de las bases de datos, nos vamos a encontrar con un algebra relacional o de conjuntos que nos enseña como preguntarle cosas a la base de datos y lo que tenemos a posterior es un lenguaje que hace las veces de una calculadora, montado sobre la lógica del algebra relacional para responder a las preguntas que le hagamos a la base de datos.

_ El algebra relacional tiene varios operadores algebraicos parecidos a los operadores de la matemática, pero en este caso los operadores básicos son seis, y la combinación de utilizar estos son los que nos van a permitir hacer preguntas a la base de datos para obtener diferentes resultados:

Operador selección σ: este operador sirve para seleccionar. A modo de análisis tenemos una relación r con sus atributos y tuplas, donde el operador selección pide que le indiquemos alguna operación matemática y condicional a la relación r, como por ejemplo σA=B ^ D > 5 (r), en donde en este caso buscamos que A sea igual a B y además D sea mayor a 5, por ende este operador va a hacer aparecer el resultado en la tabla resultante.

_ Tabla de relación r:

A	**B**	**C**	**D**
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

_ Tabla resultante:

A	**B**	**C**	**D**
α	α	1	7
β	β	23	10

_ Entonces sobre la relación r, aplicando el operador selección con el condicional que definimos obtenemos como resultado una nueva relación con nuevos datos propios de cumplir con la condición que pusimos.

_ Técnicamente el operador selección requiere de un predicado p y aplica sobre una relación r, $\sigma_p(r)$. Se define el operador selección de la siguiente manera:

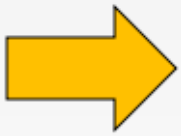
- $\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$

_ Donde seleccionamos lo que diga el predicado de la relación r, y el resultado que obtenemos es igual a todas las tuplas t de manera tal que t pertenece a la relación r y además la tupla t cumple con predicado de la condición p. Donde p es una fórmula del cálculo proposicional que consta de términos conectados por $\wedge$ (and), $\vee$ (or), $\neg$ (not). Cada término puede ser o un solo atributo, pero también puede ser un atributo operador atributo para comprar dos atributos o también puede ser atributo operador constante, para comparar un atributo con un valor constante, en donde el operador puede ser =, $\neq$, >, $\geq$, <, $\leq$.

_ Este operador elimina tuplas que no cumplen con ciertas condiciones, o deja las tuplas que cumplen con la condición que queremos. Este operador toma la relación, achica la tabla de la misma y deja las tuplas que cumplen con una condición propuesta. Un ejemplo más a continuación:

$$\sigma_{\text{snombre="SALAZAR"}}(\text{proveedores})$$

S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES
S2	JAIMES	10	PARIS
S3	BERNAL	30	PARIS
S4	CORONA	20	LONDRES
S5	ALDANA	30	ATENAS



S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES

Operador proyección $\prod$: en este caso tenemos una relación r con sus atributos, y nosotros proponemos $\prod$A,C (r) donde como resultado tenemos una nueva relación tomada de r pero que solamente tenga los atributos A y C. Entonces este operador corta la relación solamente quedándose con las columnas o los atributos que me interesan ver. Este no elimina tuplas pero si columnas.

_ Tabla de relación r:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

_ Tabla resultante:

A	C
α	1
α	1
β	1
β	2

A	C
α	1
β	1
β	2

_ Ahora aparece un criterio en el que el operador proyección, en caso de haber eliminado una columna y que en la relación resultante hubiese dos tuplas que están repetidas, automáticamente este operador elimina las tuplas repetidas para que no haya duplicación.

_ Técnicamente la definición de proyección es:

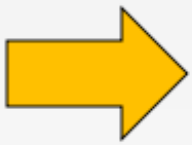
- $\prod_{A_1,A_2,\ldots,A_k}(r)$

_ En donde vamos a proyectar los atributos A1, A2, A_k (nombres de atributo) de la relación r (nombre de relación). Y el resultado se define como una relación de k columnas que se

obtiene eliminando las columnas que no están en la lista. Las filas duplicadas se eliminan del resultado, ya que las relaciones son conjuntos. Un ejemplo más a continuación:

$$\Pi_{situacion,\ ciudad}\ (proveedores)$$

S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES
S2	JAIMES	10	PARIS
S3	BERNAL	30	PARIS
S4	CORONA	20	LONDRES
S5	ALDANA	30	ATENAS



SITUACION	CIUDAD
20	LONDRES
10	PARIS
30	PARIS
30	ATENAS

Operador unión ∪: si tenemos una relación r y una relación s, se podría generar una nueva relación r ∪ s que va a tener los mismo atributos A y B, y la suma de todas las tuplas de r mas todas las tuplas de s, y si hubiera tuplas repetidas, estas van a aparecer una sola vez. Los nombres de los atributos pueden ser diferentes, pero los atributos tienen que ser del mismo tipo (letras con letras o números con números).

_ Tablas de las relación r:

A	**B**
α	1
α	2
β	1

_ Tabla de la relación s:

A	**B**
α	2
β	3

_ Tabla resultante:

A	B
α	1
α	2
β	1
β	3

_ Cuando usamos este operador si o si tiene que haber misma cantidad de atributos r y misma cantidad de atributos en s, y tienen que ser además del mismo tipo, o sea coincidentes. No necesariamente los nombre los atributos tienen que ser iguales. Y obtenemos como resultado una nueva relación y le podemos colocar cualquier nombre a la misma.

_ Técnicamente la unión se define como:

- $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$

_ En donde de la unión de r con s, el resultado es una nueva relación con todas las tuplas que pertenecen a r y/o además todas las tuplas que pertenecen a s. Para que esto funcione y sea válido, r y s deben tener la misma cardinalidad (mismo número de atributos), y además los dominios de los atributos deben ser compatibles, por ejemplo, la segunda columna de r trata el mismo tipo de valores que la segunda columna de s). Un atributo de una columna mezclado con otro de distinto nombre distinto se podría renombrar con cualquier nombre. Un ejemplo más a continuación:

$$\Pi_{snombre}(proveedores) \cup \Pi_{pnombre}(partes)$$

SNOMBRE
SALAZAR
JAIMES
BERNAL
CORONA
ALDANA
TUERCA
PERNO
BIRLO
LEVA
ENGRANAJE

Operador diferencia de conjuntos – : este operador hace la resta de r menos s, donde ambos son relaciones con sus atributos, y el resultado que obtenemos de r - s debe tener misma cantidad de atributos y misma características de estos (letras con letras o números con números), para poder hacer la diferencia de los conjuntos. Esto es similar a cuando restábamos atributos en Boyce-Codd:

_ Tabla de la relación r:

A	**B**
α	1
α	2
β	1

_ Tabla de la relación s:

A	**B**
α	2
β	3

_ Tabla resultante:

A	**B**
α	1
β	1

_ Entonces com resultado de la resta, en la nueva relación tenemos todas las tuplas de r que no se encuentran repetidas en s, en donde si aparecen en s otras tuplas que no están en r, se ignoran. Entonces técnicamente la definición seria:

- $r - s = \{t \mid t \in r \text{ y } t \notin s\}$

_ En donde el resultado son todas las tuplas que pertenecen a r y que la tupla no pertenezca a s. Las diferencias de conjuntos deben realizarse entre relaciones compatibles. Tanto r como s deben tener la misma cardinalidad, y los dominios de los atributos de r y s deben ser compatibles.

Operador producto cartesiano x: este es el más raro o difícil de entender. El producto cartesiano r x s, se produce con las tuplas de r más las tuplas de s, con los atributos de ambos, con todas las combinaciones de cada tupla de r combinada con todas las tuplas de s. Si tenemos dos atributos en r y tres en s, el resultado va a ser una relación de cinco atributos como vemos a continuación:

_ Tabla de la relación r:

A	**B**
α	1
β	2

_ Tabla de la relación s:

C	**D**	**E**
α	10	a
β	10	a
β	20	b
γ	10	b

_ Tabla resultante:

A	**B**	**C**	**D**	**E**
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

_ En el caso de haber un atributo r repetido en s, ninguna relación puede tener dos columnas con el mismo nombre, por ende si sucede que existe A en r como en s entonces es un conflicto, por lo tanto en la tabla resultante tendremos r.A y s.A, es decir, aclaramos el atributo A pero de la relación r y aclaramos el atributo A pero de la relación s, en donde se le coloca otro nombre a las columnas para que no haya problemas o conflictos.

_ La definición técnica es:

- $r \times s = \{t\, q \mid t \in r \textbf{ y } q \in s\}$

_ En donde el resultado son todas las tuplas t y q de manera tal que las tuplas de t pertenecen a r y las q a s. Suponemos que los atributos de r(R) y s(S) son disjuntos, o sea no puede haber dos que se llamen igual, es decir, $R \cap S = \varnothing$. Y si los atributos de r(R) y s(S) no son disjuntos, se debe utilizar el renombramiento en alguno de los dos atributos.

Operador de renombramiento ρ: este es un operador que le cambia el nombre a las cosas, es decir, nos permite nombrar y por lo tanto referirnos a los resultados de expresiones de álgebra relacional. Este le puede cambiar el nombre a una relación o a un atributo. Por ejemplo ρx (E) devuelve la expresión E bajo el nombre X.

Composición de operaciones: esto nos permite obtener expresiones que utilizan operaciones múltiples. Por ejemplo, podemos calcular el producto cartesiano de r con s, y podemos decir que solo nos sirve de toda la relación cuando A sea igual a C y eliminamos todas las otras tuplas. De esta manera podemos combinar todos los operadores para obtener nuevos resultados, siempre y cuando los atributos coincidan al momento de realizar la consulta:

- $\sigma A=C(r \times s)$

_ Tabla resultante de r x s:

A	**B**	**C**	**D**	**E**
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

_ Tabla resultante:

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Definición formal

_ En el algebra relacional tenemos que respetar la cardinalidad y el tipo de dato, ya que si no lo hacemos nos dará error. Nosotros podemos mezclar operaciones del algebra relacional con otras operaciones. Una expresión básica en el álgebra relacional se compone de una relación en la base de datos o una relación constante. Entonces sean E1 y E2 expresiones de álgebra relacional, todas las siguientes son expresiones del álgebra relacional:

- E1 $\cup$ E2
- E1 - E2
- E1 x E2
- σp (E1), P es un predicado de atributos de E1.
- $\prod$s (E1), S es una lista que se compone de alguno de los atributos de E1.
- ρx (E1), x es el nuevo nombre del resultado de E1.

_ El algebra relacional tiene una virtud gracias a la forma en la que ha sido construida, que es que siempre que aplicamos un operador cualquiera del algebra relacional a una relación r, el resultado que obtengo siempre es una nueva relación, aun si es un atributo con una sola tupla, por ejemplo:

$$\prod_{ciudad} (\sigma_{snombre="SALAZAR"}(\text{proveedores}))$$

CIUDAD
LONDRES

_ Lo bueno de esto es que no modificamos los atributos y los datos originales, y además con esto aplicamos recursividad, con el hecho de que a una relación le aplicamos un operador, y a la relación resultante le volvemos a aplicar otro, y asi sucesivamente, en donde la recursividad nos permitía resolver problemas complejos haciendo operaciones más simples en este caso.

Operaciones adicionales: definimos operaciones adicionales que no añaden potencia al álgebra relacional, pero que simplifican las consultas habituales. Estos son consecuencia de los operadores anteriores, es decir, se pueden ejecutar con los seis operadores básicos. Algunos son:

- Intersección de conjuntos: la intersección $r \cap s$, de r con s (ambos compatibles y con misma cardinalidad), se puede fabricar haciendo dos restas:
  - $r \cap s = r - (r - s)$

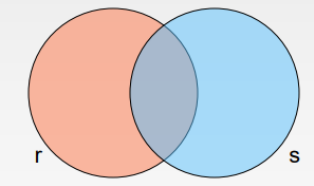


Tabla de la relación r:

A	**B**
α	1
α	2
β	1

Tabla de la relación r:

A	**B**
α	2
β	3

Tabla resultante:

A	**B**
α	2

- Reunión natural: este se construye de aplicar el producto cartesiano con algunas condiciones especiales.
- División
- Asignación

SQL

Definición del modelo

SQL: es un lenguaje de consulta de datos estructurado y que nace conceptualmente de la definiciones del algebra relacional. Partes de las condiciones que fijemos en una sentencia del lenguaje SQL tienen coincidencia con cuestiones que estuvimos viendo del algebra relacional.

_ El modelo de entidad-relación que vamos a usar para trabajar es el siguiente:

PROVEEDORES

S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES
S2	JAIMES	10	PARIS
S3	BERNAL	30	PARIS
S4	CORONA	20	LONDRES
S5	ALDANA	30	ATENAS

PARTES

P#	PNOMBRE	COLOR	PESO	CIUDAD
P1	TUERCA	ROJO	12	LONDRES
P2	PERNO	VERDE	17	PARIS
P3	BIRLO	AZUL	17	ROMA
P4	BIRLO	ROJO	14	LONDRES
P5	LEVA	AZUL	12	PARIS
P6	ENGRANAJE	ROJO	19	LONDRES

ENVIOS

S#	P#	CANT
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	400
S4	P2	200
S4	P4	300
S4	P5	400

_ A lo largo de la historia lo que se intentó mediante este lenguaje estándar, es que nosotros vamos a venir programando en cualquier lenguaje de programación, y al momento de hacerle consultas a la base de datos, nosotros ejecutamos una instrucción del formato de SQL, en donde esta instrucción le hace preguntas a la base de datos y obtiene una respuesta de la misma, y a partir de ahí seguimos operando con el lenguaje con el que se venía trabajando. Esta consulta en SQL, tiene la propiedad de que debe o debería ser exactamente igual para todos los motores de base de datos (Oracle, MySQL, SQL Server de Microsoft, etc) y la respuesta que se va a obtener es la misma. Entonces esto independiza a los lenguajes de programación de los motores de base de datos.

SQL separamos DDL y DML

_ En principio separamos nuestras sentencias en dos paquetes de sentencias grandes:

DDL (Lenguaje de Definición de Datos): este lenguaje permite crear, manipular, modificar, borrar, alterar, cambiar todas las estructuras de las tablas, índices, vistas con las que estamos operando, o sea hace a todo lo que es lenguaje de definición de datos, donde definimos la estructura de los datos. Tenemos CREATE TABLE, CREATE VIEW, CREATE INDEX, ALTER TABLE, DROP, TABLE, DROP VIEW, DROP INDEX, etc. Una vez que creamos la estructura, está por lo general se mantiene rígida. El DDL no sólo especifica información sobre la relación, sino sobre el conjunto de las relaciones, incluyendo:

- El esquema de cada relación.
- Los valores de dominio asociados a cada atributo.
- Integridad de los datos.
- Los índices que mantienen la relación entre las tablas.
- Información sobre la seguridad para el acceso a cada relación.
- La estructura física de cada relación en el disco.

DML (Lenguaje de Manipulación de Datos): son las sentencias de manipulación de datos, para cambiar las instancias, es decir, modifican la instancia de la base de datos pero no la estructura. Por ejemplo, podemos inyectar, registrar o eliminar datos en la base de datos. En este caso tenemos SELECT, UPDATE, DELETE, INSERT, etc.

_ Todo este conjunto compone el entorno total del SQL, en donde lo dividimos en dos grupos para acumular sentencias que nos sirven para definir las estructuras y sentencias que nos sirven para manipular los datos. Para que la base de datos exista, debemos usar las herramientas de DDL y asi construir la estructura (tablas e índices), y con las herramientas de DML insertamos, borramos o modificamos datos, y con la sentencia SELECT, por ejemplo, preguntamos cosas.

Tipos de dominio en SQL

- char(n): caracter de Longitud fija, especificada en la variable n.
- varchar(n): caracter de longitud variable, con un máximo especificado de longitud indicado en n.
- int: entero (un valor entero especificado que depende de la máquina).
- Smallint: entero pequeño (un valor entero especificado que depende de la máquina).
- numeric(p,d): número fijo con decimales, donde el usuario especifica la precisión de p dígitos, con n dígitos del punto decimal.
- real, double precision: número de punto flotante y de doble precisión, depende de la precisión de la máquina.

- float(n): número de punto flotante donde el usuario especifica la precisión de n dígitos.

Sentencia CREATE TABLE

_ Dado un esquema entidad-relación, mediante la sentencia CREATA TABLE podemos crear la tabla proveedores con cuatro atributos, cada uno con sus virtudes y donde algunos admiten datos de tipo null y otros no, también podemos añadir la clave primaria que en este caso va a ser snum y podemos obviar el not null. A continuación vemos:

```
CREATE TABLE proveedor(
        snum integer,
        snombre char(30) not null,
        situacion integer,
        ciudad char(30)
        primary key (snum)
)
```

_ Entonces la forma de crear una tabla es mediante la sentencia seguido del nombre de la relación r y entre paréntesis A_i que es el nombre del atributo en el esquema de la relación r, seguido de D_i es el dominio, valor y tipo de datos del atributo A_i (A_1 D_1 , A_2 D_2 , ... , A_n D_n), y podemos agregar a posterior algunas reglas de integridad para controlar algunas cuestiones del comportamiento de las tablas, como por ejemplo la clave primaria.

Estructura básica de las consultas

_ SQL está basado en operaciones relacionales y de conjunto con ciertas modificaciones y mejoras. De todas las sentencias nos enfocamos en la sentencia SELECT, esta es la que nos permite consultar información en la base de datos. Una consulta característica de SQL con SELECT tiene la forma:

SELECT A_1 , A_2 , ..., A_n
FROM r_1 , r_2 , ..., r_m
WHERE P
UNION
SELECT A_1 , A_2 **AS** B, ..., A_n
FROM
WHERE

_ Donde A_i representan los atributos, r_i representan las relaciones y P es un predicado. Esto está basado en las operaciones del algebra relacional, ya que la instrucción anterior del lenguaje SQL es equivalente a la siguiente sentencia del algebra relacional:

- $\prod A_1, A_2, ..., An(\sigma P (r_1 \times r_2 \times ... \times r_m))$

_ Esto sería la proyección de los atributos A_1 y A_2 de la selección del predicado p de las relaciones X_1, X_2, …, X_m. Un predicado p son condiciones o clausulas, que operan sobre los operadores and, or y not, y operan sobre los operadores matemáticos (<, >, =, !=, etc).

_ Entonces, este es un lenguaje que se va a construir siguiendo la lógica del algebra relacional, en donde las seis operaciones básicas se van a convertir en instrucciones del SQL. Para entender la consulta anterior, hacemos analogía con las operaciones del algebra relacional:

- La instrucción selección de P, hace referencia a seleccionar un predicado (condición) de una relación r (proveedor por ejemplo), ahora en el SQL lo visualizamos como WHERE P, donde el predicado sea P y con FROM r hacemos referencia a la relación.
- La instrucción proyección, nos permite proyectar los atributos A de la relación r, es decir, muestra solo las columnas de los atributos de la relación r que pedimos. En el caso de SQL seria la instancia SELECT A_i de r y con FROM r hacemos referencia a la relación.
- Luego tenemos el producto cartesiano, que combina todas las tuplas de r_1 con todas las tuplas de r_2 por ejemplo, y esto lo expresamos en el FROM.
- La instrucción unión permite juntar tuplas de una relación con tuplas de otra relación, siempre y cuando sean misma cantidad de atributos y del mismo tipo. Entonces en el SQL esto va a ser mediante la sentencia UNION seguido de otro SELECT, FROM, WHERE, etc, en donde va a juntar todas las tuplas del primer resultado con las tuplas del segundo resultado.
- La instrucción renombrar la podemos reemplazar en SQL con la instrucción AS para dar un nuevo nombre.

DML (Lenguaje de Manipulación de Datos)

La cláusula SELECT, FROM y WHERE

_ A continuación, vemos la sentencia SELECT con todos sus componentes. Cuando las instrucciones se escriben entre corchetes significan que son opcionales, en donde puede ir o no. Por otro lado, es preferible escribir el código en tres renglones o más, ya que en un solo renglón es muy difícil su lectura.

SELECT: se utiliza para dar la relación de los atributos deseados en el resultado de una consulta. En el algebra relacional las tuplas repetidas desaparecen automáticamente, entonces el lenguaje SQL va a tener un comodín opcional que es el DISTINCT para que las tuplas repetidas aparezcan o desaparezcan. Pero SQL permite los duplicados en las relaciones además de en los resultados de la consulta. Y luego podemos escribir o uno o muchos elementos que son o los atributos de la tabla, puede ser un comodín como el *, o puede ir una fórmula matemática.

- SELECT [distinct] elemento(s)

FROM: hacemos referencia a desde una o muchas tablas, que hacen referencia en el modelo de entidad-relación a conjunto de entidades o conjunto de relaciones. Si hay una sola tabla va a ser la tabla resultante, y si hay varias la operación que vamos a estar haciendo es el producto cartesiano entre estas. Un SELECT con un FROM es lo más chica que puede ser una instrucción.

- FROM tabla(s)

WHERE: este es opcional. Las condiciones tenían una estructura donde podíamos hacer compasiones lógicas y aritméticas.

- [WHERE condición]

_ Otras sentencias opcionales son:

- [GROUP BY campo(s)]
- [HAVING condición]
- [ORDER BY campo(s)]

_ La cláusula SELECT se utiliza para dar la relación de los atributos deseados en el resultado de una consulta. Corresponde a la operación de proyección del álgebra. Por ejemplo, para obtener los nombres de todos los proveedores de la relación proveedor seria:

SELECT snombre
FROM proveedores

_ En donde la sintaxis del álgebra relacional “puro”, esta consulta debería ser:

- ∏snombre (proveedores)

_ Con estos obtenemos el nombre de los proveedores al azar. Ahora si queremos ver otros datos como S#, situacion y ciudad, es decir, una lista de todos los atributos de la tabla proveedores, seria:

SELECT snum, snombre, situacion, ciudad
FROM proveedores

_ Ahora, por ejemplo, mostramos únicamente todos los atributos de aquellos proveedores que vivan en Parias, para ello usamos la sentencia WHERE indicando que el atributo ciudad sea igual a Paris:

SELECT snum, snombre, situacion, ciudad
FROM proveedores
WHERE ciudad = “PARIS”

_ Por ejemplo, podemos mostrar datos de la tabla proveedores y partes unidos, en este caso el Pnombre y ciudad de las partes pero solo las que son de Paris (en la tabla aparecerían primero), y hacemos una unión para mostrar el snombre y ciudad de proveedores, pero de todas las ciudades (en la tabla aparecerían por debajo de las partes):

SELECT Pnombre, ciudad
FROM partes
WHERE ciudad = “PARIS”
UNION
SELECT snombre, ciudad
FROM proveedores

_ Esto es posible siempre y cuando juntemos misma cardinalidad de los atributos y mismo tipo de dato, ya que sino no funciona. Para el nombre de la columna, como sabemos, los motores pueden tener distintos criterios a la hora de colocárselos, muchos motores utilizan el nombre de la primera parte de la sentencia, otros les ponen un nombre cualquiera. Pero podemos asignar un nombre a un atributo o relación con la instrucción AS, por ejemplo:

SELECT Pnombre, ciudad, peso **AS** PesoSituacion
FROM partes **AS** p
WHERE ciudad = “PARIS”
UNION
SELECT snombre, ciudad, situacion
FROM proveedores

_ En este ejemplo usamos WHERE que es la proyección del algebra relacional, usamos FROM que es la producto cartesiano del algebra relacional, usamos WHERE que es la selección del algebra relacional que tiene un predicado, usamos UNION que justamente es la unión del algebra relacional y usamos AS que es el renombrar del algebra relacional.

_ Por ejemplo, queremos ver el peso en gramos de las partes que son de Paris, y para esto tenemos que multiplicar los pesos por 1000, y además mostramos los pesos originales en kilos:

SELECT Pnombre, ciudad, peso, peso * 1000 **AS** peso_en_gramos
FROM partes
WHERE ciudad = “PARIS”

_ Si nosotros ejecutamos SELECT *, agregando un asterisco, mostramos todos los atributos de una relación. Por ejemplo, si queremos mostrar todos los atributos de la tabla partes, que cumplan con la condición de que la ciudad sea igual a Paris sería:

SELECT *
FROM partes
WHERE ciudad = “PARIS”

_ Nosotros debemos especificar el orden en el que queremos que aparezcan los atributos en el SELECT, pero si ponemos el * aparecen todos los atributos.

_ Por ejemplo, podemos mostrar aquellas partes cuyo peso sea mayor a 13 kilos y la ciudad a la que pertenezcan sea Londres, para esto usamos la instrucción AND para aclarar las dos condiciones, y esto sería:

SELECT Pnum, Pnombre, peso * 1000 **AS** peso_en_gramo, peso
FROM partes
WHERE (peso > 13 **AND** ciudad = “LONDRES”)

_ A modo de ejemplo, a continuación mostramos solo los nombres de la tabla partes pero aplicando el DISTINTC para que desaparezcan las tuplas con nombres repetidos, esto sería:

SELECT DISTINTC Pnombre
FROM partes

_ Ahora, en el caso de que agreguemos en el ejemplo anterior mostrar además de los nombres, sino también Pnum y usando el comando DISTINTC, esto no servirá porque Pnum no tiene el mismo valor en los nombres repetidos, por ende el resultado que muestra es el mismo que no poner DISTINTC porque no hay dos tuplas iguales.

_ Con el siguiente ejemplo, tenemos como resultado la mezcla de todas las partes con todos los proveedores, es decir, el producto cartesiano en una sola tabla (multiplicación entre todas las tuplas de la tabla parte con todas las tuplas de la tabla proveedores):

SELECT *
FROM partes, proveedores

_ El sentido de hacer este producto cartesiano es poder mezclar información de dos partes o dos relaciones. Si las relaciones que mezclamos, tienen sentido que las mezclemos, podemos encontrar relaciones o información de utilidad o interés, es decir, que lo que igualemos dentro del WHERE tenga lógica y nos sirva para algo. Por ejemplo, en la siguiente consulta igualamos la ciudad de proveedores con la ciudad de las partes, y de esta manera obtenemos como resultado todos los proveedores que comparten las mismas ciudades en las que se fabrican las partes:

SELECT *
FROM partes, proveedores
WHERE proveedores.ciudad = partes.ciudad

_ El igualar dentro del WHERE, es lo que nos va a asegurar a nosotros que el producto cartesiano que estemos viendo empiece a tener lógica y sirva para algo.

_ Y asi podemos combinar, por ejemplo, de la tabla partes, todos los atributos y de la de proveedores, solo el nombre y la situación:

```
SELECT  partes.*, snombre, situacion
FROM partes, proveedores
```

_ También un ejemplo para combinar sería el siguiente, en donde el resultado que vamos a obtener son la cantidad total de envíos pero con los datos del proveedor que le corresponde ya que igualamos el snum de la tabla envíos con el de la tabla proveedores:

```
SELECT *
FROM proveedores, envíos
WHERE proveedores.snum = envios.snum
```

_ Ahora al número de envió lo mezclo con todas las partes, pero solo nos sirven cuando el envío Pnum coincida con la parte Pnum, con lo cual podemos asociar los 12 envíos con los datos de los proveedores y además las partes con sus datos:

```
SELECT *
FROM proveedores, envíos, partes
WHERE (proveedores.snum = envios.snum AND
        envios.Pnum = parte.Pnum)
```

_ En el producto cartesiano podemos multiplicar cualquier tabla con cualquier otra, probablemente sin ningún tipo de sentido, ahora si hay atributos coincidentes, el igualarlos hace que nos quedemos con las tuplas que sirven de esa mezcla que establecimos, pero solamente quedándome con tuplas que tienen coincidencia y con lo cual son realmente son útiles. Con el producto cartesiano mezclamos cualquier cosa con cualquier cosa, y después podemos poner un criterio de coincidencia para darle una lógica. Nosotros decidimos que hacer con los datos que estamos visualizando.

_ Si queremos que los proveedores se enfrenten de ida y vuelta todos contra todos hacemos el siguiente producto cartesiano en donde mezclamos dos veces en el FROM la tabla proveedores. Como primera medida, para esta ejecución de la consulta, los renombramos para que quede más claro. Luego en el WHERE colocamos el signo <> que es de distinto, y nos sirve para evitar que se enfrenta un proveedor contra sí mismo:

```
SELECT *
FROM proveedores AS pr1, proveedores AS pr2
WHERE pr1.snum <> pr2.snum
```

_ Ahora si queremos que los proveedores se enfrenten solamente de ida y no de vuelta (es decir una sola vez), como por ejemplo, Jaime contra Salazar pero no Salazar contra Jaime, para esto indicamos en el WHERE que el nombre de pr1 sea alfabéticamente menor que el de pr2 y no al revés, por ende seria (si sacamos el <> funcionaria igual):

```
SELECT *
FROM proveedores AS pr1, proveedores AS pr2
WHERE pr1.snum <> pr2.snum AND pr1.snombre < pr2.snombre
```

_ El motor de SQL es muy flexible y nos permite mezclar cualquier cosa con cualquier cosa, y por otro lado requiere de mucho ingenio y creatividad de nuestra parte para dar la solución que responda a una pregunta o problema determinado.

Operaciones con cadenas

_ Tenemos las instrucciones LIKE y BETWEEN. Para explicar estos, lo hacemos mediante el siguiente ejemplo, en donde el comparador LIKE nos muestra aquellos nombres que contengan las letras NA, no importan ni las primeras ni las ultimas letras, sino que quiero que en algún lado el o los nombres tengan una “n” y una “a” juntas. Y con el comparador BETWEEN indicamos que la situacion este entre los valores 15 y 40. Por ende vamos a obtener como resultado proveedores que cuya situacion esta entre 15 y 40 y además tienen la siglas NA en sus nombres:

```
SELECT snum, snombre, situacion
FROM proveedores
WHERE (snombre LIKE “%NA%” AND situacion BETWEEN 15 AND 40)
```

_ El comando LIKE es mas flexible que el parámetro igual (=), ya que con este último hay que ser especifico en la comparación.

Orden en la presentación de las tuplas

_ Con la instrucción ORDER BY ordenamos el resultado que vamos a obtener por algún criterio, campo o atributo. Por ejemplo, en este caso ordenamos alfabéticamente por el nombre del proveedor 1:

```
SELECT *
FROM proveedores AS pr1, proveedores AS pr2
WHERE pr1.snum <> pr2.snum AND pr1.snombre < pr2.snombre
ORDER BY pr1.snombre
```

_ No se puede ordenar todo a la vez, sino que cada sentencia da un orden distinto, ya que por ejemplo no podemos ordenar alfabéticamente por el nombre del proveedor 1 y 2 al mismo ya que estos no coinciden o no son iguales. Por otro lado, si al atributo en el ORDER BY no le escribimos nada al lado, automáticamente el orden en que lo muestra es ascendente, pero si le ponemos DESC ordena de forma descendente, y con ASC podemos aclarar que el orden sea ascendente. También podemos ordenar por dos criterios, por ejemplo por número de situacion en este caso, donde primero se ordena por pr1, y si se repiten los pr1 va a ordenar por pr2:

```
SELECT *
FROM proveedores AS pr1, proveedores AS pr2
WHERE pr1.snum <> pr2.snum AND pr1.snombre < pr2.snombre
ORDER BY pr1.situacion DESC, pr2.situacion DESC
```

Funciones de agregación

SUM: tenemos una función sumar, que sirve para sumar datos numéricos, como por ejemplo sumar todas las cantidades de la relación envíos:

```
SELECT  SUM( cant )
FROM envios
```

GROUP BY: esta sentencia es opcional y me permitiría, cuando sumamos las cantidades por ejemplo, ir agrupando por algún criterio. Por ejemplo, si agrupamos por número de proveedor y a su vez mostramos el número de proveedor en el SELECT, lo que obtenemos como resultado es la sumatoria de las cantidades pero agrupadas por cada proveedor:

```
SELECT  snum, SUM( cant )
FROM envios
GROUP BY snum ASC
```

_ Entonces obtenemos como un resumen donde muestra los snum de los proveedores de la tabla envió con la cantidad de unidades enviadas sin importar que producto mando, ordenado a su vez por snum de forma ascendente. El GROUP BY sirve para agrupar o compactar tuplas que tengan atributos que se repiten. Si en el ejemplo anterior agregamos el número de partes Pnum, lo que sucede es que pierde sentido al momento de haber compactado todo, por ende al número de parte lo pierdo y en algunos casos puede dar error. Por ende, la regla del SQL que opera para el GROUP BY dice que cuando yo agrupo por algún atributo, en el SELECT lo único que se puede visualizar es el mismo atributo por el que yo agrupe, y luego a continuación de este podemos mostrar funciones de agregación (suma, promedio o AVG, max, min, contar, etc). Cuando agrupamos por un determinado atributo, esa agrupación hace que todos los otros atributos, que no se agruparon, se rompan por más que haya coincidencia.

COUNT: con esta instrucción podemos contar tuplas, por ejemplo podemos mostrar agrupar los proveedores por ciudad para saber cuántos hay en cada una, en donde con el count * contamos directamente cuantas tuplas hay de esa ciudad, sin importar el resto de los datos:

```
SELECT  ciudad, count( * )
FROM proveedores
GROUP BY ciudad
```

_ Por ejemplo, para saber cuántas partes hay de cada color hacemos:

```
SELECT color, count( * )
FROM partes
GROUP BY color
```

_ Para hacer un ejemplo más complejo, mostramos el total de partes que se han enviado de cada una de ellas, el código y además el nombre de cada parte. Para ello mostramos el numero de la parte, el nombre y la cantidad total enviada, pero como el nombre de las partes no esta en la tabla envíos, sino que esta en la tabla partes, tenemos que hacer un producto cartesiano de las dos tablas en el FROM y en el WHERE aclaramos que el Pnum de la tabla envíos coincida con el de la tabla partes, luego agrupamos por Pnum para que los compacte y también agrupamos por nombre. Podemos agrupar por nombre y por número porque los atributos son los mismos en cada uno, es decir, todos los P1 tiene el mismo nombre y los mismos atributos, y asi con el resto.

```
SELECT  envios.Pnum, Pnombre, sum( cant ) AS total
FROM envios, partes
WHERE envios.Pnum = partes.Pnum
GROUP BY envios.Pnum, Pnombre
```

HAVING: este es un condicional exactamente igual que el WHERE pero funciona después de haber agrupado, es decir, funciona después del GROUP BY y el WHERE funciona antes del GROUP BY. Es simplemente a los fines de ordenar la sentencia. Por ejemplo, al ejemplo anterior le agregamos mediante el HAVING que la suma de las cantidades sea mayor a 450:

```
SELECT  envios.Pnum, Pnombre, sum( cant ) AS total
FROM envios, partes
WHERE envios.Pnum = partes.Pnum
GROUP BY envios.Pnum, Pnombre
HAVING sum( cant ) > 450
ORDER BY sum( cant ) DESC
```

_ No podemos colocar que la suma sea mayor a 450 en el WHERE porque en la parte del WHERE no se compacto la tabla, es decir, no se ordenó por algún criterio, por eso se coloca en el HAVING ya que previamente en el GROUP BY ya se compacto la tabla y ahora mostramos los datos que queramos con el HAVING de acuerdo a esa nueva tabla con sus nuevos datos.

_ Cuando el motor de consultas ejecuta la sentencia, lo primero que sale a buscar el motor es el FROM, es decir, busca que tablas vamos a necesitar para ejecutar la acción, lo segundo que hace el motor es aplicar el filtro del condicional WHERE, si tuviésemos GROPU BY es lo tercero que hace, luego una vez que agrupo hace el HAVING, después ordena con el ORDER BY, y por último al final de todo el motor decide que columnas son

las que va a mostrar, en el SELECT. Este es el orden en el que el motor ejecuta las sentencias.

_ Otro ejemplo, vamos a listar las partes enviadas y la cantidad total enviada de cada una:

```
SELECT p.Pnum, Pnombre, SUM( cant )
FROM partes AS p, envios AS e
WHERE p.Pnum = e.Pnum
GROUP BY p.Pnum, Pnombre
```

_ Siempre que nosotros preguntamos algo a la base de datos con la sentencia SELECT, el resultado que obtenemos es una nueva tabla que no existe físicamente en la base de datos, sino que es temporal, y nosotros a su vez podemos realizarle consultas a esa nueva tabla, anidando consultas. Si las preguntas son muy difíciles, nosotros podemos resolver esta con muchas preguntas más sencillas para luego conformar la pregunta original. Por ejemplo, armamos una pregunta más pequeña donde solamente nos interesa saber los totales de la tabla envíos, agrupados por número de partes (compactamos la tabla envíos):

```
SELECT Pnum, SUM( cant ) AS total
FROM envios AS e
WHERE p.Pnum = e.Pnum
GROUP BY p.Pnum
```

_ Ahora, al ejemplo anterior le hacemos otra pregunta, en donde a la tabla en la que tenemos agrupados los envíos, le ponemos un nuevo nombre “Tot” porque son el total de los envíos, y hacemos el producto cartesiano de Tot con las partes, igualando los números de partes, y listamos *:

```
SELECT *
FROM ( SELECT Pnum, SUM( cant ) AS total
        FROM envios AS e
        WHERE p.Pnum = e.Pnum
        GROUP BY p.Pnum ) AS Tot, partes AS p
WHERE Tot.Pnum = p.Pnum
```

_ Entonces, obtenemos como resultado todas las partes mezcladas con el agrupamiento de las cantidades enviadas.

AVG: con esta instrucción podemos calcular el promedio. Y como ejemplo a continuación, calculamos la cantidad promedio de envíos (valor del envío promedio):

```
SELECT AVG( cant )
FROM envios
```

_ Teniendo en cuenta este resultado, ahora listamos las partes enviadas cuyo envió ha sido superior al envío promedio, es decir, debemos listar todos los envíos que se hayan hecho siempre y cuando la cantidad supere el valor del envío promedio, entonces ponemos el SELECT anterior como parte de la nueva consulta:

```
SELECT *
FROM envios
WHERE cant > ( SELECT AVG( cant )
               FROM envios )
```

_ Lo que aparece como concepto es el hecho de poder enganchar o anidar sentencias SELECT una con otras, para poder construir respuestas que pueden ser muy complejas pero que las podemos construir como muchas pequeñas respuestas más sencillas.

Subconsultas anidadas

IN / NOT IN: a modo de ejemplo pensamos en listar las partes que no han sido enviadas por ningún proveedor. Como primera medida listamos todas las partes (con el número de partes) que si han sido enviadas por algún proveedor:

```
SELECT DISTINCT Pnum
FROM envios
```

_ Por ende, ahora mostramos todo el listado de partes, siempre y cuando no esté dentro de la lista de las partes que si han sido enviadas y lo hacemos mediante el NOT IN, por lo que obtendremos como resultado aquel número de parte que no está en la lista de las que si han sido enviadas:

```
SELECT *
FROM partes
WHERE Pnum NOT IN ( SELECT DISTINCT Pnum
                    FROM envios )
```

_ Además podemos usar el IN, y en ese caso mostramos todas las partes de la lista que si han sido enviadas. Tenemos otros comodines como SOME, en donde si igualamos el Pnum del WHERE a SOME, vamos a listar algunas de las partes que están en la lista o tabla nueva. Y también tenemos el ALL, en la que siguiendo la lógica anterior podemos mostrar por ejemplo aquella parte que sea mayor a todas las que están en la nueva lista, como vemos a continuación:

```
SELECT *
FROM partes
WHERE Pnum > ALL ( SELECT DISTINCT Pnum
                   FROM envios )
```

Valores nulos

NULL: cuando hablamos de los datos de tipo NULL o nulos, estos no se consideran al momento de sumar, ni al momento del mínimo, ni el máximo, etc, ya que significa “nose”. Por ejemplo, si tenemos algunas partes en las que en el valor del peso es NULL y ejecutamos la siguiente consulta, al momento de hacer todos los cálculos, los NULL se ignoran:

SELECT SUM(peso), **MIN**(peso), **MAX**(peso), **AVG**(peso), **COUNT**(peso), **COUNT**(*)
FROM partes

_ En el caso del COUNT(peso), no es lo mismo que el COUNT(*), ya que el COUNT(peso) cuenta las tuplas que tienen un valor de peso asignado, ignorando los que tienen NULL, y el COUNT(*) cuenta todas las tuplas de la tabla partes.

IS / IS NOT: para usar estos comodines para comparar. Podemos mostrar por ejemplo aquellas partes donde el color no sea NULL:

SELECT *
FROM partes
WHERE color **IS NOT** NULL

_ También podemos preguntar, por ejemplo, cuantas partes hay de cada color en la tabla de partes:

SELECT color, **COUNT**(color)
FROM partes
WHERE color **IS NOT** NULL
GROUP BY color

_ Haciendo la consulta anterior un poco más compleja, además queremos saber cuántas partes con color NULL hay en la tabla partes, por ende unimos el resultado de la consulta anterior con la nueva, y esto lo hacemos porque tenemos la misma cardinalidad y mismo tipo de dato:

(**SELECT** color, **COUNT**(color)
FROM partes
WHERE color **IS NOT** NULL
GROUP BY color)
UNION
(**SELECT** “No se el color”, **COUNT**(*)
FROM partes
WHERE color **IS** NULL)

_ Cuando colocamos “No se el color”, estamos modificando el nombre del atributo, es decir, unimos con los colores establecidos del primer SELECT un nuevo nombre como

atributo, pero si lo colocamos asi nomas y sin ningún valor, aparte de tomar a este como nombre en el encabezado de la tabla resultante, va a aparecer como valor de todas las tuplas. Con ingenio podemos fabricar la sentencia que nos haga falta.

Modificación de la base de datos

INSERT: con esta sentencia agregamos datos nuevos a la tabla. Por ejemplo, insertamos una parte nueva dentro de la tabla partes, en donde seguido del INTO asignamos los campos o los atributos (los campos son opcionales, por lo que si no los escribimos, el sistema automáticamente va a asumir todos los campos de la tabla y el orden en el que se van a ir asignando va a coincidir con el orden en el que se encuentran en la tabla originalmente) y con VALUES asignamos los valores que van a tener:

INSERT INTO partes
(Pnum, Pnombre, color, peso, ciudad)
VALUES (“P7”, “TORNILLO”, “NARANJA”, NULL, “CORDOBA”)

_ Si agregamos en el INTO un atributo que no existe en la tabla, saltara error. Por lo que para añadir un nuevo atributo tenemos que usar la sentencia ALTER TABLE que modifica la tabla, y esta sentencia como vimos pertenece al DDL para definir las estructuras. Además, podemos hacer un INSERT de un SELECT y añadir toda una nueva tabla de un solo tiro.

DELETE: con esta sentencia eliminamos en la tabla datos o tuplas que cumplen con una cierta condición. Como por ejemplo, borramos todos los proveedores de la ciudad de Londres, y de esta forma no aparecerán más:

DELETE FROM proveedores
WHERE ciudad = “LONDRES”

_ Si en la consulta anterior quitamos el WHERE, vamos a borrar todos los proveedores de la tabla. También como ejemplo podemos borrar de la tabla envíos, todos aquellos envíos cuyo número de envió este en todos los envíos que sean de los proveedores que están en la ciudad de parís, y asi armamos una consulta compleja y especifica:

DELETE FROM envios
WHERE snum **IN** (**SELECT** snum
FROM proveedores
WHERE ciudad = “PARIS”)

UPDATE: esta sentencia permite actualizar datos de la tabla. Por ejemplo, aquellas partes cuyo color sea rojo se van a actualizar, cambiando el nombre de su ciudad actual por la de Buenos Aires:

UPDATE partes
SET ciudad = “BUENOS AIRES”
WHERE color = “ROJO”

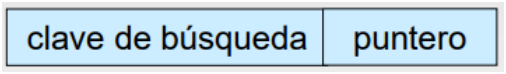
Indexación y asociación

Conceptos básicos

_ Las bases de datos son archivos donde cada registro esta enlazado con el otro, en una especie de lista doblemente enlazada, pero los indicies son el concepto central que dan agilidad y dan velocidad al movimiento. Es por eso que nosotros planteamos que el archivo en si no está ordenado por ningún criterio sino que cada registro que se va agregando es un nuevo registro enlazado en la lista de registros, y luego se trabaja con los índices. Los índices son los que nos van a dar a nosotros la visualización de que los datos de la tabla se encuentran ordenados por alguna clave primaria. La forma de indexado se realiza con los conceptos de árbol binario o hashing estático o dinámico. Cuando nosotros creemos una tabla con sus atributos, y cuando referenciemos que índices va a tener esa tabla, en la referenciación de los índices justamente es donde vamos a construir el árbol binario o la tabla de hash que nos van a agilizar al momento de buscar un determinado registro. Existen diferentes mecanismos de indexación empleados para acelerar el acceso a los datos deseados. Por ejemplo, la clave UCC de la facultad.

Clave de búsqueda: es un atributo, del conjunto de atributos, empleado para buscar registros en un archivo.

Archivo de índices: consta de registros, llamados entradas de índice, de la forma:



_ Los archivos de índices generalmente son más pequeños que el archivo original. Tenemos dos clases básicas de índices:

- Índices ordenados: las claves de búsqueda se almacenan de forma ordenada.
- Índices asociativos: las claves de búsqueda están distribuidas uniformemente en “cajones”, empleando una “función de asociación”.

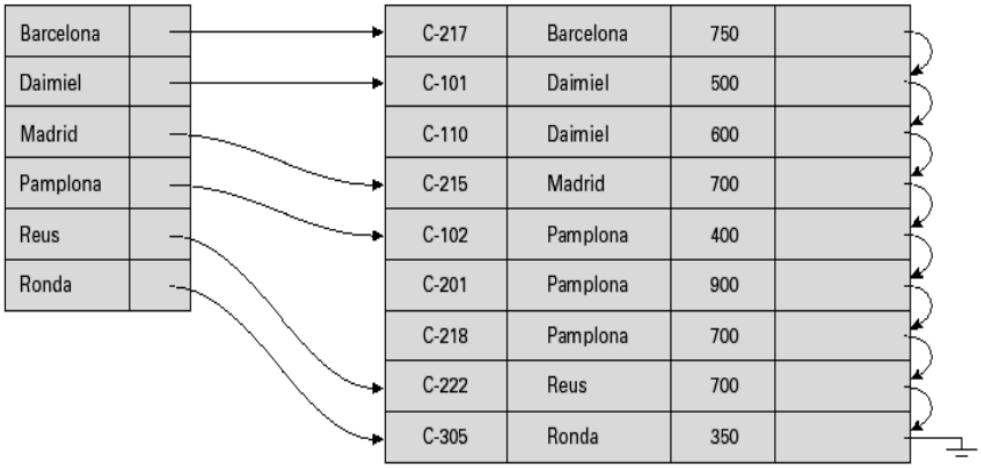
Métricas de evaluación de índices

_ Las métricas para saber que índice es el correcto a la hora de implementarlo son:

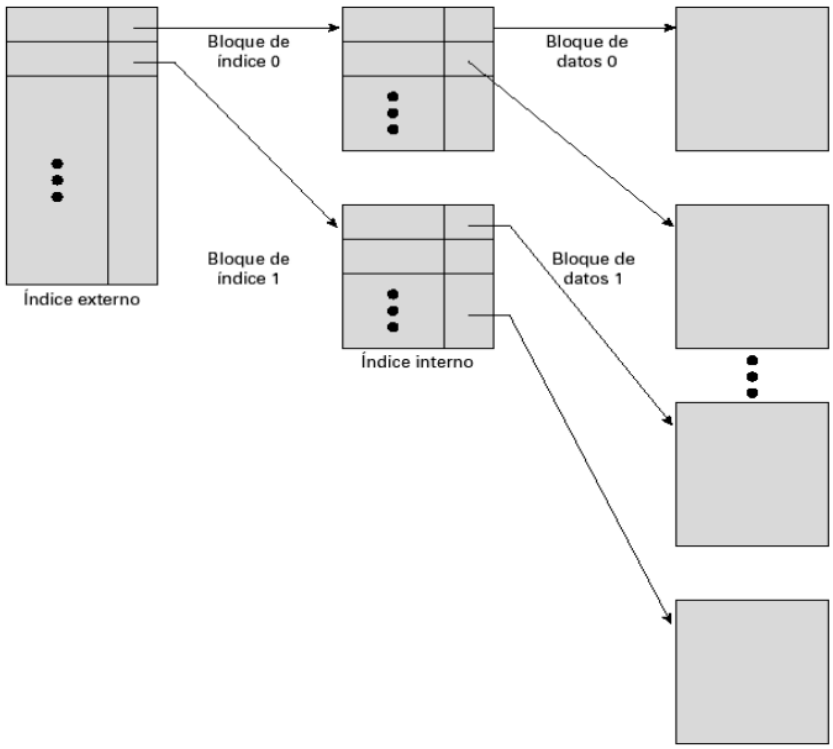
- Tipos de acceso soportados eficientemente, como por ejemplo, registros con un valor concreto en el atributo o registros con un valor de atributo que se encuentra en un determinado rango de valores.
- Tiempo de acceso
- Tiempo de inserción
- Tiempo de borrado

- Costes de espacio

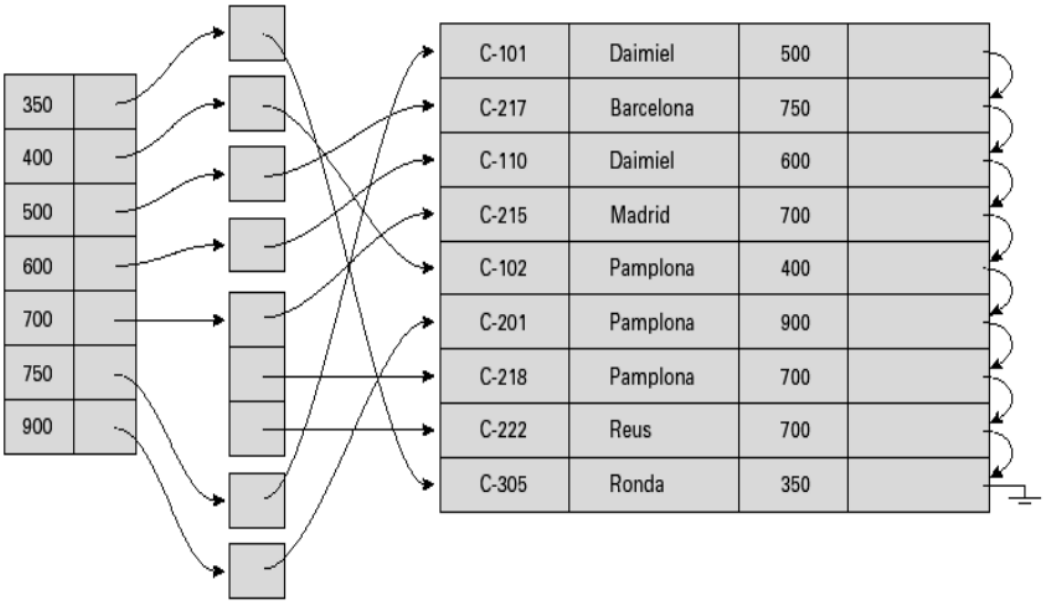
_ Todos estos tiempos se modifican porque son impactados por los índices con los que nosotros trabajamos. Cuando queremos acceder a un determinado dato, no vamos a ir a la lista enlazada y buscarlo uno por uno, sino que nos movemos a través de un árbol binario de índices que ágilmente nos a llevar a una determinada posición de la lista enlazada. Como vemos a continuación, los registros de la tabla están encadenados unos con los otros y a medida que agregamos mas continuamos con el encadenamiento, pero por otro lado tenemos un árbol de punteros que nos lleva directamente a una determinada posición.



_ Al crecer la cantidad de bloques de registros, luego creen los bloques de los índices y asi también el bloque del índice externo principal. Entonces, hacemos una búsqueda desde el índice externo, este nos lleva a otro bloque de subíndices hasta llegar al bloque que tiene específicamente los datos. No buscamos en la base de datos uno por uno los datos, sino que buscamos a través de índices, y como vemos se crea una estructura de tipo árbol.



_ Cuando nosotros creamos la tabla, en el mismo momento vamos a plantear la creación de un índice o múltiples índices. Esa creación de los índices es lo que va a hacer de que cada vez que demos de alta un registro nuevo en la zona de datos, tengamos que actualizar la estructura de arboles para que podamos encontrar ese registro rápidamente con el buscador de índices. Cada agregado, borrado o modificación en la tabla no solo tiene un impacto en la zona de datos, sino que también impacta en tener que corregir todos los índices que se hayan construido para luego moverse más ágilmente en la base de datos.



_ No tiene sentido, por ejemplo, crear un índice llamado barrio de la entidad alumno, ya que nunca lo vamos a buscar por este. Y al agregar un alumno nuevo tenemos que modificar o actualizar para que queden bien organizados los índices de barrio, y si no usamos nunca esta clave, el mantenimiento de esta es innecesario porque nunca lo usamos para buscar. Siempre deberíamos crear índices que sean los elementos de búsqueda para localizar un alumno en este caso, como por ejemplo la clave UCC, ya que este da agilidad para encontrar cosas como nombres, notas, teléfonos, mails, direcciones, etc. Deberíamos analizar cuales son los índices que son óptimos para el funcionamiento del sistema, ya que la creación de índices consume tiempo y esfuerzo por ende debemos crear los que sean realmente útiles o necesarios. Luego, estos índices se van a convertir técnicamente, en el motor de base de datos, en un árbol binario o en una tabla de hash (si el archivo es chico) con un acceso directo, para poder encontrar rápidamente un atributo en la base de datos. Cuando los archivos son chicos, el manejo de índices conviene que sea a través de tablas de hash, ahora cuando los archivos son grandes conviene utilizar un árbol binario.

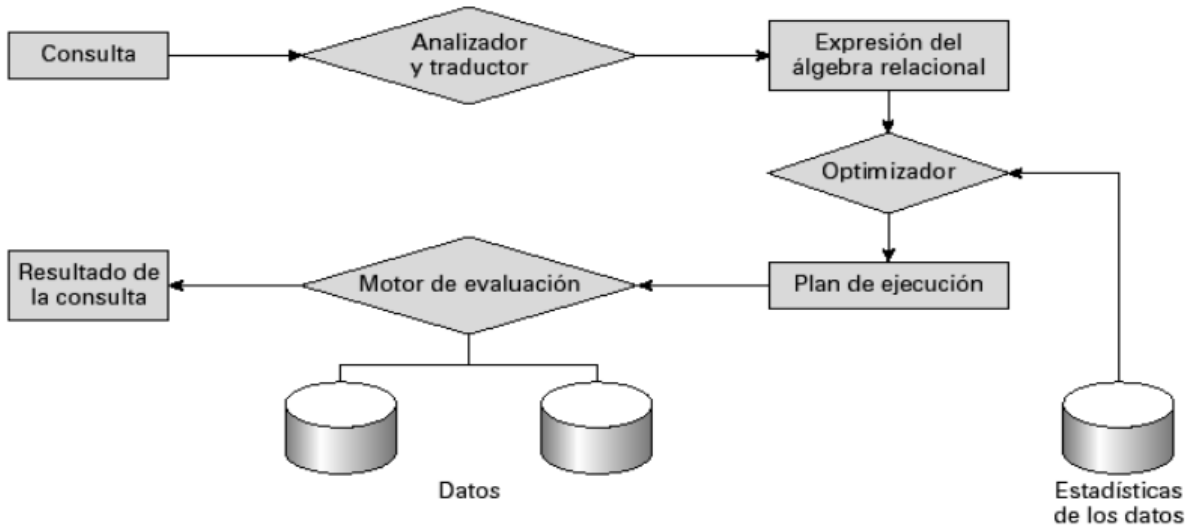
_ Además, cuando se actualiza un índice cuando insertamos, borramos o modificamos los registros, esto es parecido a lo que vimos en estructura de datos, como agregar, borrar o

modificar un nodo en el árbol binario, solo que esta direccionado al concepto de base de datos, pero técnicamente es lo mismo, tanto para claves primarias como para claves secundarias.

Procesamiento de consultas

Pasos básicos en el procesamiento de consultas

_ A continuación analizamos como opera el gestor de base de datos al momento de ejecutar una consulta. Cuando escribimos una sentencia SELECT, FROM, WHERE, etc, el gestor de base de datos, al momento de ejecutar la consulta, lo primero que hace es analizar la consulta que hacemos (que su sintaxis sea correcta) y realizar una traducción de la misma a una expresión del algebra relacional, luego una vez que obtiene la expresión, el gestor pasa por una herramienta que se llama optimizador que va a intentar ver si la sentencia que se escribió, se puede mejorar de alguna manera, por ejemplo, si en un FROM estamos planteando un producto cartesiano de alumnos con notas, el optimizador intenta verificar porque nos conviene filtrar, por aquellos que tengan nota mayor a 4, etc, por ende en el WHERE podemos tener varios condicionantes y el optimizador va a analizar cual de estos conviene aplicar primero. Todo esto se hace en función a una pequeña base de datos con estadísticas, que el mismo gestor de base de datos va produciendo con las preguntas que habitualmente se le hacen, como por ejemplo cada vez que buscamos una nota, esta consulta se guarda, y asi sucesivamente, ayudando al gestor a poder detectar que condicional es más conveniente a la hora de hacer consultas.



_ El optimizador entonces se alimenta de estadísticas que el mismo gestor de base de datos va construyendo. Esto surge de ir almacenando estadísticas de prueba y error, e ir generando información que agiliza el motor de base de datos a armar un plan de

ejecución lo más rápido posible, y evaluar realmente la base de datos para obtener el resultado de la consulta, pero mirando su propia estadística de información.

Optimización

_ Una expresión del álgebra relacional puede tener muchas expresiones equivalentes, como por ejemplo, las siguientes expresiones equivalentes cuyo resultado es el mismo:

- σsaldo$<$2500($\prod$saldo(cuenta))
- $\prod$saldo(σsaldo$<$2500(cuenta))

_ Cada operación de álgebra relacional se puede evaluar empleando uno de los diferentes algoritmos. Igualmente, una expresión del álgebra relacional se puede evaluar de muchas maneras.

Plan de evaluación: es la expresión comentada que especifica la estrategia de evaluación detallada. El motor podría tener un plan de evaluación y por otro lado podría proponer creaciones de índices que agilicen la búsqueda. Por ejemplo, puede usar un índice sobre saldo para encontrar las cuentas con saldo < 2.500, o puede llevar a cabo el rastreo de la relación completa y eliminar las cuentas con saldo $\geq$ 2.500.

Análisis de costo de consultas: hace referencia a cuánto cuesta una consulta en SQL y como optimizar el tiempo de esa consulta a buscar la instancia más optima de ejecución de la consulta. Entre todos los planes de evaluación equivalentes, el motor puede seleccionar el de menor coste. El coste se estima empleando información estadística del catálogo de la base de datos, como por ejemplo, el número de tuplas de cada relación, el tamaño de la tuplas, etc.

- El coste se mide generalmente como el tiempo total transcurrido para responder la consulta. Hay numerosos factores que contribuyen al coste del tiempo como accesos al disco, CPU, o incluso las comunicaciones de la red.

Transacciones

Concepto de transacción

_ Cuando nosotros hacemos una transacción bancaria, como humanos vemos este proceso como una sola actividad, pero a nivel de base de datos ocurren múltiples operaciones.

Transacción: la definimos como un conjunto de operaciones que forman una única lógica de trabajo, es decir, es una unidad de ejecución de programa que puede acceder, y posiblemente actualizar, uno o varios elementos de la base de datos. Cuando alguien va a pagar un ticket en el supermercado, quien hace la operación del ticket registra todos los productos que pasa por la caja y luego va a cerrar la operación con un pago con tarjeta de crédito en tantas cuotas por ejemplo, por ende se denomina la unidad de transacción en el momento en el que esa operación esta terminando y en un solo bloque el sistema va a dar de baja del stock los productos que se vendieron, además va a registrar la operación de la tarjeta de pago, y entre otras cosas, en donde todo esto tiene que cerrar como un bloque único, ya que no hay chances que se de baja el stock y nos olvidemos de registrar la operación con la tarjeta de pago, ya que todo tiene que suceder en el mismo momento.

_ Otro ejemplo de transacción es cuando vamos al cajero y decimos que queremos extraer tal cantidad de monto, entonces el cajero tiene que verificar que tiene dinero para entregarnos, tiene que verificar que tenemos saldo en nuestra cuenta, y si todo eso sucede, en un solo bloque de transacción, el cajero nos entrega el dinero, registra contablemente, registra en el banco la transacción, da de baja el dinero, y todo esto sucede como un solo bloque de acciones juntas, ya que no hay forma de que ocurra una sin que suceda el resto.

_ El sistema gestor de base de datos se tiene que asegurar que esta transacción, como una unidad lógica, se produzca de manera adecuada. La transacción o se ejecuta completa o no se ejecuta. Por otro lado debe permitir que se ejecuten múltiples transacciones y que se pueda gestionar la concurrencia de las mismas. A las transacciones las podemos ver como un programa o código dentro de la ejecución, además estos programas pueden hacer operaciones pertinentes en la base de datos, y estas operaciones se pueden identificar y delimitar por las llamadas a la función de estas transacciones. Para poder asegurar la integridad de las operaciones que se realizan, se establecen distintos puntos de falla dentro de las transacciones para poder volver a tras en caso de que se produzcan. En el caso de que la transacción se haya realizado con éxito se hace un commit, y de esta manera se cumple de que una transacción es una operación que no debería depender de otra para asegurar la integridad de la base de datos, sino que como paquete se ejecuta y la base de datos se mantiene de manera consistente. Entonces consideramos que:

- Una transacción debe ver una base de datos consistente.
- Durante la ejecución de la transacción la base de datos puede ser inconsistente.
- Cuando se compromete una transacción la base de datos deber ser consistente.
- Se pueden ejecutar múltiples transacciones en paralelo.
- Dos enfoques principales a tener en cuenta son los fallos de varias clases, tales como fallos de hardware y caídas del sistema, y el otro es la ejecución concurrente de múltiples transacciones.

Propiedades ACID

_ Se estableció como regla general, de que se mantengan las propiedades ACID para todas las transacciones y asi se pueda mantener la consistencia en la base de datos. Estas propiedades contemplan:

Atomicidad: hace referencia a que o todas las operaciones de la transacción se realizan y se reflejan correctamente en la base de datos, o no se realiza ninguna, y esto es asi para mantener la consistencia, ya que no puede haber una operación en el bloque de la transacción que no se ejecute.

Consistencia: la ejecución de una transacción en aislamiento preserva la consistencia de la base de datos. Esto hace referencia a que una transacción tiene que ser una unidad lógica entera de principio a fin, en donde por ejemplo no podemos hacer un transacción al mismo tiempo que sea descantar plata de mi cuenta y agregarle a otra.

Aislamiento: una base de datos debe permitir que se generen varias transacciones y que se ejecuten concurrentemente a la vez en la misma, y cada transacción debe ignorar lo que sucede o lo que se ejecute en las otras transacciones. Los resultados de las transacciones intermedias deben ocultarse de otras transacciones ejecutadas concurrentemente, es decir, las operaciones se realizan intermedio dentro de un programa y recién se impactan todos los cambios al final de la transacción una vez que se hayan realizado todas las operaciones de manera correcta en caso de que no se produzco un error.

- Por cada par de transacciones T_i y T_j, le parece a T_i, que, o bien T_j ha terminado su ejecución antes de que comience T_i , o que T_j ha comenzado su ejecución después de que Ti terminara.

Durabilidad: hace referencia a que tras la finalización con éxito de una transacción, los cambios que se realizaron en la misma, impacten en la base de datos permaneciendo en ella, incluso si hay fallos en el sistema.

_ Cuando hablamos de durabilidad, hay una garantía en la ejecución de quien registra la transacción, de que la información se ha grabado en el disco duro, con lo cual ya no hay posibilidades de pérdida de la información.

Ejemplo de transferencia de fondos

_ Suponemos que queremos transferir cierto monto desde una cuenta “A” a una cuenta “B” mediante los siguientes pasos:

- Paso 1: leer(A)
- Paso 2: A := A - 50
- Paso 3: escribir(A)
- Paso 4: leer(B)
- Paso 5: B := B + 50
- Paso 6: escribir(B)

_ La transacción informa que leemos cuánto dinero hay en la cuenta de “A”, luego restamos 50 pesos de la cuenta de “A”, grabamos ese dato, después leemos cuánto dinero hay en la cuenta de “B”, le sumamos 50 pesos a la cuenta de “B”, y por último guardamos el dato. Este bloque completo es en sí la transferencia. Pero para esto tenemos que cumplir ciertos requisitos:

Requisito de atomicidad: si hay un error en la transacción después del paso 3, es decir, después de escribir en la cuenta “A” y antes del paso 6, es decir, antes de escribir en la cuenta “B”, nosotros y el sistema deberíamos asegurar que la base de datos no se actualice o sus actualizaciones no se reflejan en la misma, asi evitamos inconsistencias.

Requisito de consistencia: al estar aislada la ejecución de la transacción, la suma de “A” y “B” no se altera por la ejecución de la misma.

Requisito de aislamiento: si entre los pasos 3 y 6 se permite acceder a otra transacción a la base de datos parcialmente actualizada, veríamos una base de datos inconsistente, por ejemplo que la suma de A + B sea menor de lo que debería. El aislamiento se puede asegurar de forma trivial ejecutando las transacciones secuencialmente, es decir, una detrás de la otra.

Requisito de durabilidad: desde que se notifica al usuario que se ha completado la transacción correctamente (es decir, que ha tenido lugar la transferencia de 50), las actualizaciones de la base de datos producidas por la transacción deben permanecer, a pesar de los fallos que hayan ocurrido.

Estados de la transacción

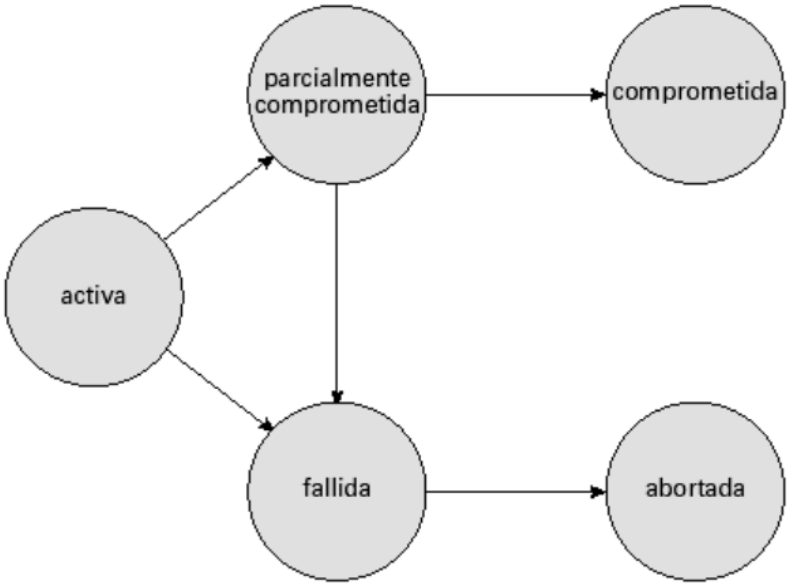
Activa: es el estado inicial de la transacción, y la misma siempre permanece en este estado mientras se está ejecutando. De este estado puede pasar a fallida o parcialmente comprometida

Parcialmente comprometida: la transacción toma este estado después que se ha ejecutado la instrucción final y a partir de acá puede pasar a estar en un estado fallida o comprometida.

Fallida: sucede después de descubrir que la ejecución normal ya no puede llevarse a cabo por algún error.

Abortada: en donde después de que la transacción se ha retrocedido y la base de datos restauró a su estado anterior al inicio de la transacción (estado inicial), aquí tenemos dos opciones después de que haya abortado, una es cancelar directamente la transacción, y la otra es reiniciar la transacción y volver a hacerla sólo si no hay errores lógicos internos.

Comprometida: es cunado la transacción se ha terminado con éxito.



Implementación de atomicidad y durabilidad

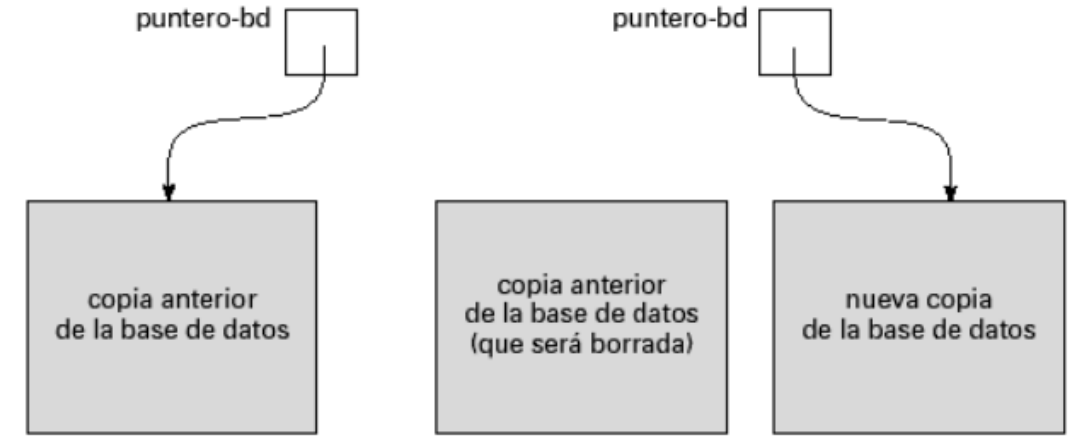
_ Esto básicamente está ligado a la gestión de recuperación de un sistema de base de datos, fijándose en tres de las propiedades ACID que serían la atomicidad, la consistencia y la durabilidad. El caso del aislamiento esta queda en manos del protocolo de control de concurrencias.

Gestión de recuperación: esta se encarga del manejo de errores. Los errores de transacción que podemos tener son los siguientes:

- Errores lógicos o internos (de programación).
- Errores de sistema, ya sean de hardware o software.
- Errores de almacenamiento (no se tienen en cuenta porque no hay ningún sistema gestor de base de datos que recupere los datos a no ser que tenga un backup).

_ Existen distintos mecanismos para lograr esta gestión, los más conocidos son:

- Shadow paging o esquema de la base de datos en la sombra: en este esquema se asume que hay una sola transacción corriendo al mismo tiempo y que únicamente, cuando a la transacción se la toma como correcta, la pagina es visible a los otros. Esto se logra a partir de punteros. Vamos a tener un puntero llamado puntero-bd que siempre apunta a la última copia consistente de la base de datos, entonces realizamos una copia de esta y sobre esta copia es donde nosotros vamos a aplicar las distintas acciones de la transacción, como modificar algún valor o algo, y una vez que se haya alcanzado un compromiso parcial y todas las páginas se hayan desviado al disco, ahí si el puntero pasa a apuntar a esta nueva copia con las modificaciones de la transacción. Y en caso de que haya un error, al puntero se lo puede apuntar a la copia anterior de la base de datos para no perder la consistencia.



(a) Antes de la actualización (b) Después de la actualización

Este no es factible de implementar, ya que en la práctica no es viable al tener que hacer una copia de toda la base de datos y además esto es caro. Al hacer modificaciones y redireccionar los punteros, los campos anteriores quedan no asignados, por lo que las reasignaciones se fragmenta el disco. También la información no queda ordenada, provocando un bajo rendimiento para el gestor de base de datos a la hora de acceder a la información. Por último, solo se soporta una transacción a la vez.

- Write-Ahead Logging (WAL): este mecanismo consta en registrar todas las acciones en disco y en memoria para deshacerlas en el caso de error, es decir, que nosotros vamos a tener un registro lógico, que es una porción del disco separada de la base de datos, y se llama asi porque guarda todas las operaciones lógicas que hace la transacción. De cada transacción se va a guardar su ID, el objeto que se modificó, el valor anterior que nos va a permitir deshacer la acción, y un valor posterior que nos va a permitir rehacer la acción. Con los procesos de deshacer y rehacer lo que vamos a hacer es recuperar la consistencia de la base de datos. Este método se debe asegurar de que los camios se escriban sobre el registro antes de realizar la escritura del disco. Se asume que el registro esta almacenado en un lugar estable.

Ejecuciones concurrentes

_ L a ejecución concurrente nos permite ejecutar múltiples o varias transacciones al mismo tiempo, y esto nos va a dar algunas ventajas como:

- Aumento de la utilización del procesador y del disco, conduciendo a mejorar la productividad de la transacción. Por ejemplo, podemos tener una transacción que está utilizando la CPU, mientras tenemos otra que está leyendo o escribiendo en el disco, logrando una mayor eficacia a la hora de ejecutar transacciones.
- Reducción del tiempo medio de respuesta de la transacciones, en donde por ejemplo las transacciones pequeñas no tienen necesidad de esperar detrás de transacciones las grandes.

_ El sistema gestor de base de datos logra la concurrencia a partir del solapamiento de las acciones de las transacciones. Es decir, el sistema gestor de base de datos a partir del protocolo de control de concurrencias, nos lleva a un método en el cual este sistema gestor va viendo la mejor forma de ir ordenando las operaciones para que no haya conflicto. Existen dos tipos de categorías del protocolo:

- Pesimistas: serian en los cuales se considera en los que si o si van a haber fallas, por lo que se va haciendo un registro constante de cada transacción cosa de que cuando haya fallas se puedan ejecutar las operaciones de hacer o rehacer para llevarlo a un estado consistente.

- Optimista: acá se considera que los errores son muy poco probables, por lo que se lleva a la operación a un punto antes de hacer la transacción, y ahí es donde se revisa si hubo algún tipo de conflicto, en donde si lo hay se hace un rollback llevándolo a un estado de consistencia.

Planificaciones

_ Las planificaciones son secuencias o cadenas que indican un orden cronológico en el que se van ejecutando las instrucciones de las transacciones concurrentes, se decir, las transacciones que están en simultaneo. Una planificación para un conjunto de transacciones debe constar de todas las instrucciones de las mismas, y además se debe preservar el orden en el que aparecen las instrucciones en cada transacción individual. Y como dijimos antes, si una transacción en su ejecución tiene alguna falla, esta va a tener como última instancia una recuperación y sino puede ser abortada. Caso contrario si una transacción se completa correctamente, tendrá como última sentencia instrucciones de compromiso.

Planificación secuencial: a modo de ejemplo vemos una planificación de dos transacciones, en donde la primer transacción T_1 se encarga de hacer una transferencia de 50 pesos desde "A" a "B", y la segunda transferencia T_2 realiza una transferencia del 10% del saldo desde "A" a "B". Como podemos ver esta es una planificación secuencial en la que primero se realizan todas las instrucciones de T_1 y después todas las de T_2. Hasta que no termina la transacción 1 no se realiza ninguna acción de la transacción 2. Luego tenemos otras transacciones, en donde en este caso comienza T_2 y luego T_1. Estas planificaciones son secuenciales pero no son equivalentes por lo que los resultados no son los mismos.

T_1	T_2
leer(A)	
$A := A$ - 50	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$temp := A * 0,1$
	$A := A$ - $temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)

T_1	T_2
	leer(A)
	$temp := A * 0,1$
	$A := A$ - $temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)
leer(A)	
$A := A$ - 50	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	

Planificación equivalente: en el ejemplo que presentamos no vemos una planificación secuencial ya que las instrucciones en las transacciones se van intercalando, pero de todas maneras esta transacción es equivalente con el primer ejemplo de planificación secuencial , ya que tanto en la planificación secuencial 1 anterior como en esta, se preserva la suma A + B, es decir, producen el mismo resultado final.

T_1	T_2
leer(A)	
$A := A$ - 50	
escribir(A)	
	leer(A)
	$temp := A * 0,1$
	$A := A$ - $temp$
	escribir(A)
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(B)
	$B := B + temp$
	escribir(B)

Secuencialidad

_ Nosotros empezamos de la suposición de que cada transacción nos va a preservar la consistencia de la base de datos, y esto es por los principios que ya nombramos. Y también es importante destacar esto porque cuando uno hace una ejecución secuencial, como primero se realiza una transacción, y cada transacción o se aborta o se compromete, siempre la base de datos esta en un estado consistente, entonces si se ejecutan de forma secuencial las transacciones, nos aseguramos de que la base de datos siempre va a estar consistente porque o se aborta o se compromete la misma.

_ Siguiendo con los ejemplos de la planificación, a veces si uno hace una ejecución concurrente, la base de datos puede quedar en un estado no consistente, en donde nos muestre que se haya generado dinero de la nada o haya desaparecido el sistema bancario por una mala planificación, por eso es importante generar bien las planificaciones.

_ Una ejecución secuencial preserva la consistencia de la base de datos, y las planificaciones concurrentes nos interesan mucho porque son mas eficientes que una planificación secuencial, pero tenemos que tener cuidado de que una planificación concurrente mal hecha pueda romper la base de datos, aunque cada transacción de por si este bien hecha.

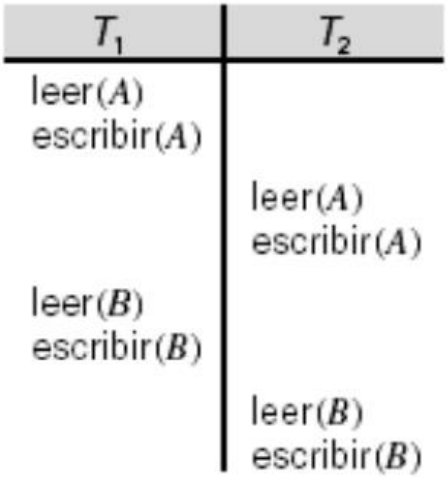
_ A la hora del análisis de las transacciones y la planificación, solamente vamos a tener en cuenta las operaciones de lectura y escritura sobre un dato y no todo lo que se haga en el medio, porque nosotros lo tomamos como cálculos aleatorios, pero lo que seguro sucede es lectura y escritura, y aparte esto es lo que impacta en la base de datos

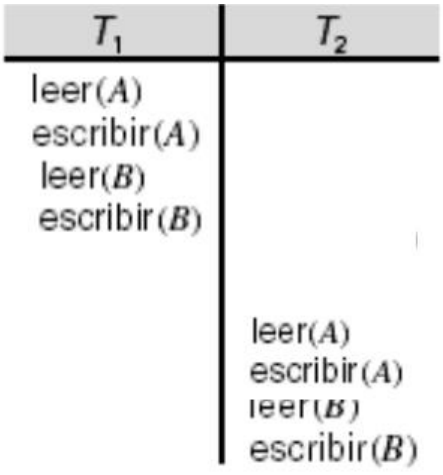
Secuencialidad en cuanto a conflictos: las instrucciones l_i y l_j , de las transacciones T_i y T_j respectivamente, entran en conflicto si y sólo si existe algún elemento Q accedido por ambas l_i y l_j, y al menos una de estas instrucciones grabó Q. Por ejemplo, si miramos de la base de datos de alumnos la nota que se saco cada uno, y otra persona esta modificando las asistencias, no pasa nada porque no tienen nada que ver las notas con las asistencias. Pero si todos estamos tratando de leer las asistencias y alguien las esta modificando, acá entramos en conflicto. Entonces, hay conflicto cuando alguien esta leyendo y otro escribiendo, viceversa, o ambos están tratan de escribir, pero siempre y cuando sea el mismo dato el cual esta siendo impactado, ya que si los datos son diferentes, podrían trabajar todos con distintas transacciones al mismo tiempo sin tener impacto.

- l_i = leer(Q), l_j = leer(Q), l_i y l_j no están en conflicto.
- l_i = leer(Q), l_j = escribir(Q), hay conflicto.
- l_i = escribir(Q), l_j = leer(Q), hay conflicto.
- l_i =escribir(Q), l_j = escribir(Q), hay conflicto.

_ Las planificaciones secuenciales nos garantizan que la base de datos se mantenga consistente, pero tienen la desventaja de ser lentas, por lo tanto tienden a ser inviables, entonces lo que hacemos es que si tenemos una planificación P, esta se pude transformar en otra P' por medio de una serie de intercambios de instrucciones no conflictivas, y se dice que P y P' son equivalentes en cuanto a conflictos. Ahora, se dice que la planificación P es secuenciable en cuanto a conflictos si es equivalente en cuanto a conflictos a la planificación secuencial P'.

_ A continuación tenemos un ejemplo, en donde por un lado tenemos una planificación concurrente y otra secuencial, pero por otro lado vemos que la planificación concurrente a su vez es secuenciable en cuanto a conflictos, ya que en la transacción 1 tenemos la escritura de "A" y en la transacción 2 tenemos la lectura de "A", luego escribimos "A" pero en la transacción 1 se lee "B" y acá no hay conflicto ya que los datos son distintos.





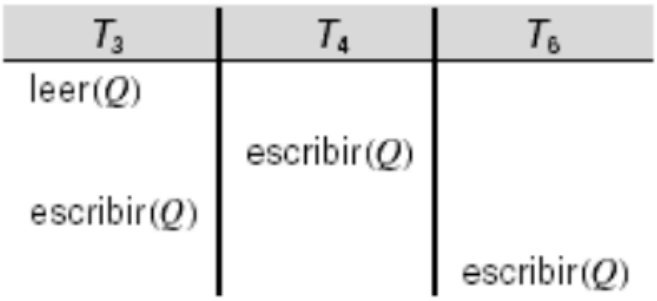
_ Entonces mediante intercambio de instrucciones que no están en conflicto podemos pasar de la ejecución concurrente a la secuencial, y esto nos indica que la planificación concurrente está bien hecha y que no va a generar inconsistencia a la hora de realizar las transacciones.

Secuencialidad en cuanto a vistas: si tenemos dos planificaciones S y S' que tengan el mismo conjunto de transacciones. Para que estas dos sean equivalentes en cuanto a vistas si deben cumplir tres condiciones:

- Por cada elemento de datos Q, si la transacción T_i lee el valor inicial de Q en la planificación S, entonces la transacción Ti debe, en la planificación S', leer también el valor inicial de Q. Es decir, la primera vez que se lee un valor o dato Q, debe ser ejecutado por la misma transacción.
- Por cada elemento de datos Q, si la transacción T_i ejecuta leer(Q) en la planificación S y ese valor fue producido por la transacción T_j, entonces la transacción T_i debe leer también, en la planificación S', el valor de Q que fue producido por la transacción T_j . Es decir, cuando leemos un valor en una transacción, tiene que haber terminado de ser escrito por la misma transacción en ambas planificaciones
- Por cada elemento de daros Q, la transacción que lleva a cabo la operación final escribir(Q) en la planificación S, debe realizar la operación final escribir(Q) en la planificación S'. Esto es similar al primero solo que la ultima vez que se escribe un valor, debe ser por la misma transacción.

_ Para estas condiciones se cumplan solo nos basamos en lecturas y escrituras.

_ Para que una planificación S sea secuenciable en cuanto a vistas, tiene que ser equivalente en cuanto a vistas a una planificación secuencial, y para esto debe cumplir con las condiciones anteriores. Cada planificación secuenciable en cuanto a conflictos es también secuenciable en cuanto a vistas. Pero si es secuenciable en cuanto a vistas no necesariamente tiene que ser en cuanto a conflictos, como vemos a continuación, en donde a T_6 no le importa el valor que haya antes ya que va a sobrescribir lo que hay en Q:



_ Una planificación equivalente a la anterior podría ser si ejecutamos primero T_3 luego T_4 y T_6, ya que nuestro primer leer siempre va a estar en T_3 y asi se cumpla la primer condición, luego para que se cumpla la tercera, el escribir siempre lo va a ejecutar T_6, y por mas de que se cambie el orden en el escribir de T_3 y T_4, esto no afecta la segunda condición ya que no hay ninguna otra instrucción de leer.

_ En resumen, estamos intentando planificar secuencias que nos permitan ejecutar varias transacciones al mismo tiempo y no que sean una secuencial de otra, sino que estamos intentando generar planificaciones que nos permitan ejecutar transacciones todas al mismo tiempo. Vamos a intentar resolver esto con dos lógicas, donde una es control en cuanto a conflictos en la que si se produce un conflicto significa que no puedo planificar de esta manera, o sea, los conflictos son los que frenan la planificación. La otra lógica nos permite la secuencialidad, no mirando los conflictos, sino con un criterio de vistas, en el cual si alguien miro un dato en una planificación, luego debería mirarlo en tal momento en la próxima planificación para que todo funcione.

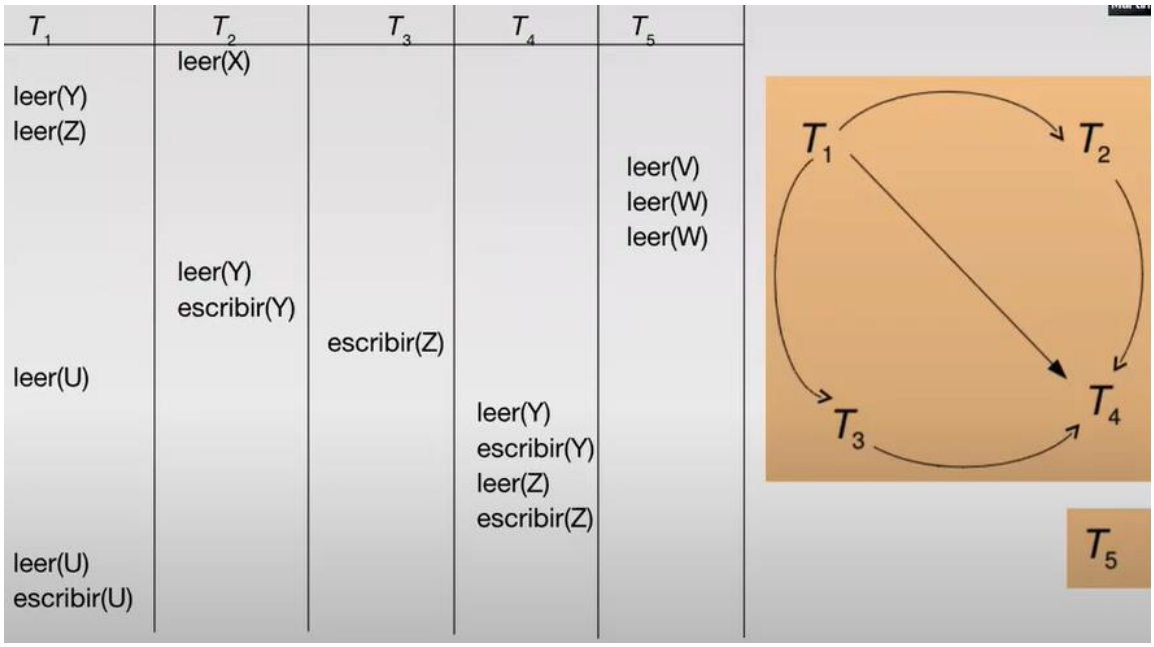
Prueba de secuencialidad

_ Para las pruebas de secuencialidad vamos a considerar alguna planificación de un conjunto de transacciones T_1, T_2, …, T_n.

Grafo de precedencia: es un grafo directo en donde los vértices son las transacciones. Teniendo en cuenta a este, lo que hacemos ahora es dibujar un arco sobre las transacciones que estén en conflicto, en este caso desde T_i hasta T_j si las dos transacciones están en conflicto, y T_i accedió al elemento de datos sobre el que el conflicto surgió antes. Es decir, el arco va a salir de la transacción que primero hay accedido al elemento de datos sobre el cual tenemos el conflicto, y se va a dirigir a la transacción que después haya accedido a ese elemento. A ese arco o arista lo identificamos por el elemento o dato en si al que se accedió.

_ A continuación analizamos el siguiente ejemplo, en donde en primer lugar tenemos la transacción T2 que utiliza el dato X, y que si vemos bien, a este dato no lo utiliza ninguna transacción, entonces esta no va a entrar en conflicto. La transacción T1 usa las variables Z e Y, que después van a ser utilizadas por la transacción T2, T3 y T4, entonces lo que hacemos lo que hacemos es dibujar un grafo, que desde T1 sale un arco dirigido a T2, T3 y T4. Por otro lado, vemos que la transacción T2 tiene el dato Y, que además es utilizado por la transacción 4, por ende hay un conflicto, entonces en el grafo sacamos un arco dirigido

desde T2 a T4, y de igual forma de T3 a T4 ya que la transacción T3 utiliza el dato Z y T4 también. En el caso de la transacción 5, como esta utiliza datos que no son utilizados por el resto de las transacciones, significa que no va a entrar en conflicto, entonces lo señalamos como un nodo aislado sin arcos con otras transacciones.



Prueba para la secuencialidad en cuanto a conflictos: una planificación es secuenciable en cuanto a conflictos si y sólo si su grafo de precedencia es acíclico. Existen algoritmos de detección de ciclos que toman nota n^2 veces, donde n es el número de vértices del grafo. Entonces, si el grafo de precedencia es acíclico, el orden de secuencialidad se puede obtener por una ordenación topológica del grafo, en donde hay una orden lineal consistente con el orden parcial del grafo. Por ejemplo, teniendo en cuenta el grafo anterior, un orden de secuencialidad para la planificación sería:

- $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$
- $T_5 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$

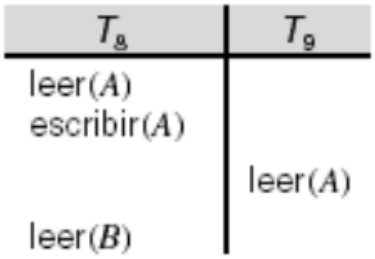
Prueba para la secuencialidad en cuanto a vistas: tomamos el grafo de precedencia que ya haya pasado por la prueba de secuencialidad en cuanto a conflictos, y lo vamos a modificar. Hacer esta prueba es improbable.

Recuperabilidad

_ La recuperabilidad es la necesidad de dirigir el efecto de los fallos en una transacción, en transacciones que se ejecutan concurrentemente.

Planificación recuperable: si una transacción T_j lee un elemento de datos previamente escrito por una transacción T_i, la operación comprometer de T_i aparece antes que la operación comprometer de T_j. Es decir, una planificación es recuperable cuando los fallos en la transacción no afectan a las transacciones concurrentes.

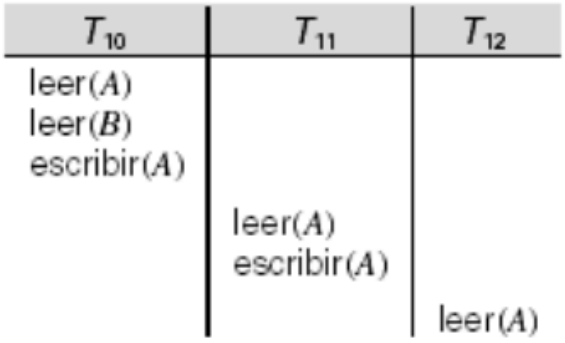
_ Hemos logrado que las transacciones se ejecuten de forma secuencial, una a continuación de la otra y esperar hasta que una termine, luego con el sistema de planificación logramos armar planificaciones que permiten que se ejecuten varias transacciones a la vez, si estas no tienen impacto con las otras o si los impactos pueden ser resueltos o considerados de alguna manera. En este caso, una vez que logramos armar la planificación y vamos ejecutando transacciones en paralelo para ir ganando tiempo, pero puede ser que alguna transacción de la secuencia falle y se tenga que volver para atrás, es decir, no termina con un commit sino con un rollback y la transacción se vuelve para atrás. Entonces, a modo de ejemplo, tenemos la siguiente planificación con las transacciones T_8 y T_9, en donde esta planificación no es recuperable si T_9 se compromete inmediatamente después de la lectura, porque si comprometo, después de leer A, a la transacción 9, nosotros estamos guardando los cambios y a su vez se están superponiendo con información de la transacción 8.



_ Entonces, una forma correcta para que la planificación sea recuperable, tenemos que guardar los cambios de la transacción 8, o sea comprometerlos después de escribir A, y de esta manera se guardan correctamente y pasamos a la transacción 9 a comprometerlo después de leer A. de esta manera tenemos las transacciones recuperables y no se mezcla una información con la otra. Básicamente lo que hacemos es asegurar la ubicación de las operaciones comprometidas.

Retroceso en cascada: cuando un solo fallo en la transacción conduce a un conjunto de retrocesos de la transacción. Acá si va a ser recuperable, pero si tenemos un fallo en la transacción 10, por ejemplo, vamos a perder todo el trabajo que hicimos para volver a ese fallo. En la siguiente planificación, ninguna de las transacciones se ha comprometido aún

(de tal manera que la planificación es recuperable), pero si T_{10} falla, T_{11} y T_{12} también deben ser retrocedidas, porque tenemos que volver a encontrar el error, y esto puede conducir a la pérdida significativa de trabajo.



Planificaciones sin cascada: seria lo opuesto a lo anterior, en donde acá lo que hacemos es comprometer cada transacción, en donde por ejemplo luego de comprometer la transacción 10, el trabajo se va a guardar, y pasamos a la transacción 11, y si tenemos un fallo en la transacción 10, volvemos a esta, la modificamos y no perdemos todo el trabajo.

_ Hay que tratar de evitar el retroceso en cascada ya que todo el proceso que uno hace se pierde al momento de encontrar un fallo.

Control de concurrencia

_ Un sistema gestor de base de datos debe proporcionar un mecanismo que asegure que todas las planificaciones generadas son aceptables, es decir que la base de datos sea consistente y los cambios sean reversibles. Estos mecanismos se llaman esquema de control de concurrencia, y hay muchos tipos que varían según el grado de concurrencia que permiten y el coste que incurren, como por ejemplo:

- Esquema de control: en donde las transacciones, pertenecientes a la planificación, bloquean completamente a la base de datos, de tal manera que ninguna otra transacción puede acceder a los datos de la base de datos mientras se mantenga dicho bloqueo. Y este bloqueo solamente se libera cuando la transacción se compromete. Todas las planificaciones generadas bajo este esquema son secuenciales, y por lo tanto son recuperables y sin cascada. Pero asi como son secuenciales, tienen un bajo rendimiento y un grado de concurrencia muy pobre.

_ El objetivo principal de los esquemas de control de concurrencia son:

- Otorgar un alto nivel de concurrencia en las planificaciones.

- Asegurar que las planificaciones generadas sean secuenciales en cuanto a conflicto o vistas.
- Asegurar que las planificaciones sean recuperables y sin cascada.

Niveles de aislamiento

_ Dejando de lado las planificaciones y volviendo a las transacciones en sí, una transacción define un nivel de aislamiento que especifica el grado en el que se debe aislar una transacción, de las modificaciones de las demás transacciones. Estos niveles se describen en base a los niveles permitidos en una aplicación escrita en un lenguaje de alto nivel. En términos generales:

Nivel de aislamiento alto: nos otorga:

- Menos efectos adversos.
- Menos simultaneidad para acceder a los datos.

Nivel de aislamiento alto: nos otorga:

- Más efectos adversos.
- Más simultaneidad para acceder a los datos.

_ El nivel apropiado para nuestra aplicación depende del equilibrio entre la integridad de los datos y la sobrecarga de recursos que nuestra aplicación necesita.

Definición de transacción en MySQL

_ Un sistema gestor de base de datos nos debe de proveer con ciertas palabras reservadas que nos permitan definir una transacción explícitamente. En MySQL, estas palabras son las siguientes:

START TRANSACTION: no permite iniciar la transacción.

COMMIT: termina la transacción, guarda los datos en la base de datos, y si hay algún tipo de bloqueo en la misma, lo libera.

ROLLBACK: termina la transacción, reinvierte los cambios en vez de guardarlos, y si hay algún tipo de bloqueo en la misma, lo libera.

SET TRANSACTION ISOLATION LEVEL: nos permite cambiar el nivel de aislamiento de la transacción.

_ En MySQL cada vez que realizamos una instrucción DML, como un INSERT, UPDATE o DELETE, por debajo se ejecuta un commit implícito por defecto, por lo que cada instancia DML es una transacción. Y mediante un rollback podemos volver atrás si nos equivocamos para dejar todo como estaba.

Control de concurrencia

Protocolos basados en bloqueos

_ Un bloqueo es un mecanismo para controlar el acceso concurrente a un elemento de datos. Los elementos de datos se pueden bloquear de dos maneras:

Modo exclusivo (X): en este caso el elemento de datos además de leerse se puede escribir. Un bloqueo de este tipo se solicita con la instrucción bloquear-X.

Modo compartido (S): en donde los elementos de datos sólo se pueden leer. Un bloqueo de este tipo se solicita con la instrucción bloquear-S.

_ Las solicitudes de bloqueo se dirigen al gestor de control de concurrencia, en donde la transacción puede realizar la operación sólo después de que se conceda la solicitud.

_ Entonces, por un lado esta el concepto de transacción, en el que por ejemplo en el caso de una transacción de dinero, la transacción busca asegurar de que de la salida y llegada a destino del dinero, o sucede todo, o no sucede nada, es decir, no se puede quedar en el medio la operación, no se puede sacar el dinero y que no llegue a destino. Otro tema que tenemos es el hecho de como bloqueamos por un pequeño instante la cuenta de la que sacamos el dinero, para que nadie modifique los importes de las cuentas mientras estamos haciendo una operación, y otra persona está haciendo otra distinta en otro lugar pero tratando de operar sobre la misma cuenta. Entonces acá surge la concurrencia, es decir, mucha gente tratando de acceder al mismo tiempo, apareciendo el tema de los bloqueos para que nadie pueda tocar lo que no le corresponde.

Matriz de compatibilidad de bloqueos: a una transacción se le puede garantizar un bloqueo en un elemento, si el bloqueo solicitado es compatible con los bloqueos que ya tengan otras transacciones sobre ese mismo elemento. Cualquier número de transacciones puede tener bloqueos compartidos sobre un elemento, pero si una de ellas tiene un bloqueo exclusivo sobre un determinado elemento, ninguna otra puede tener ningún otro bloqueo sobre dicho elemento. Y como vemos en el esquema, cuando son dos bloqueos compartidos se marca como “cierto” y si es un bloqueo exclusivo se marca como “falso”:

	C	X
C	cierto	falso
X	falso	falso

_ Si no se puede garantizar un bloqueo, la transacción que lo solicita tiene que esperar hasta que los bloqueos incompatibles que tienen otras transacciones se hayan liberado, y a continuación se autoriza el bloqueo.

Protocolo de bloqueo: es un conjunto de reglas que siguen todas las transacciones cuando se solicitan o se liberan bloqueos. Los protocolos de bloqueo restringen el número de planificaciones posibles.

Fallos de los protocolos basados en bloqueos

_ Analizando el siguiente esquema, tenemos que ni T_3 ni T_4 pueden progresar ya que la ejecución de bloquear-S(B) ocasiona que T_4 espere a que T_3 libere su bloqueo sobre B, mientras que la ejecución de bloquear-X(A) ocasiona que T_3 espera a que T_4 libere su bloqueo sobre A. Esta situación se denomina interbloqueo, y para solucionarlo, uno de los dos, T_3 o T_4 , debe retroceder y liberar sus bloqueos.

T_3	T_4
bloquear-X(B)	
leer(B)	
$B := B - 50$	
escribir(B)	
	bloquear-C(A)
	leer(A)
	bloquear-C(B)
bloquear-X(A)	

_ En la mayoría de los protocolos de bloqueo se pude producir un interbloqueo potencial. Y también es posible que se produzca la inanición si el gestor de control de concurrencia se diseñó defectuosamente, y esto causa por ejemplo, que puede que una transacción esté esperando un bloqueo exclusivo sobre un elemento determinado mientras que una secuencia de transacciones diferentes solicita y obtienen la autorización de bloqueo compartido sobre el mismo elemento, entonces la misma transacción retrocede repetidamente debido a los interbloqueos. Por ende, el gestor de control de concurrencia se puede diseñar para impedir que se produzca la inanición.

Protocolos de bloqueos de dos fases

_ Este protocolo asegura la secuencialidad, y lo que hace es exigirles a las transacciones que cumplan con las siguientes fases:

- Fase de crecimiento: en donde las transacciones pueden conseguir bloqueos, y las mismas no pueden liberar los bloqueos.

- Fase de decrecimiento: en donde las transacciones pueden liberar bloqueos, pero las mismas no pueden conseguir bloqueos.

_ Con este protocolo se puede probar que las transacciones se pueden secuenciar en el orden de sus puntos de bloqueo, es decir, el punto de la planificación en el cual la transacción obtiene su bloqueo final (último bloqueo al que va a llamar la transacción antes de comenzar el decrecimiento).

_ Este protocolo no asegura la ausencia de interbloqueos. Al momento de hacer una transferencia bancaria por ejemplo, antes se hacen muchas preguntas para determinar la operación, como por ejemplo, uno indica que quiere hacer una transacción, de esta cuenta a tal otra, de tal cantidad o monto, indicamos para cuando queremos la transferencia, etc, y cuando todos estos detalles o preguntas se hayan hecho y tengamos que confirmar que queremos realizar la transacción, al momento de confirmar se acabo el tiempo de preguntarle cosas al usuario y se bloquea la base de datos. Si en el medio de que bloqueamos la base de datos, nos ponemos a preguntarle cosas al usuario y este demora en responderme, vamos a estar con bloqueos eternos de la base. Entonces, primero preguntamos todo lo que haya que preguntar, y una vez que se recaudó toda la información, se bloquea la base de dato, se realizan las operaciones y luego se desbloquea, finalizando la operación, para que esto pase en el menor tiempo posible. Y ahí es donde aparecen los siguientes bloqueos:

- Protocolo de bloqueo estricto de dos fases: en él, una transacción debe mantener todos sus bloqueos exclusivos hasta que se complete o aborte.
- Protocolo de bloqueo riguroso de dos fases: es más estricto que el anterior, en donde en él, todos los bloqueos se mantienen hasta que se complete o aborte. En este protocolo las transacciones se pueden secuenciar en el orden en que se comprometen.

_ Pueden existir planificaciones secuenciable en cuanto a conflictos que no se pueden obtener si se utiliza el bloqueo en dos fases. Sin embargo, si no hay información adicional de las transacciones, si es necesario que se dé el bloqueo en dos fases para conseguir la secuencialidad en cuanto a conflictos. Esto se da de la siguiente manera, dada una transacción T_i que no sigue el bloqueo en dos fases, se puede encontrar otra transacción T_j que si lo utilice, tal que exista una planificación para T_i y T_j que no sea secuenciable en cuanto a conflictos.

<u>Conversión de bloqueo:</u> este procedimiento se conoce como subir el bloqueo, que se da en la fase de crecimiento, o bajar el bloqueo, que se da en la fase de decrecimiento. Esto nos va a permitir favorecer a la concurrencia. Lo que se hace es subir o bajar los bloqueos para permitir en ciertos momentos que varias transacciones utilicen esos elementos y puedan bloquearlos para que no haya inconsistencia cuando se quiera hacer una modificación de los datos.

Adquisición automática de bloqueos: las transacciones cuando llaman a una instrucción ya de por si hacen un bloqueo, tanto para las operaciones de leer como de escribir. Suponemos que el gerente del banco quiere saber la cantidad de saldo en todas las cuentas, entonces se genera un reporte indicando el saldo de cada persona. Ahora, mientras se genera el reporte, en el medio podría haber personas que estén sacando plata, modificando de esta forma el reporte que vería el cliente, de ahí la necesidad de estos bloqueos para permitir que continue la secuencia independientemente de que se esté generando un reporte.

Implementación de bloqueos

Gestor de bloqueos: se puede implementar como un proceso independiente, en el que las transacciones le envían los bloqueos y desbloqueos requeridos. Entonces, el gestor de bloqueos responde a un bloqueo solicitado, enviando un mensaje de concesión de bloqueo, o un mensaje preguntando por la transacción a retroceder, en el caso de un interbloqueo, luego la transacción solicitada espera hasta que se atiende su solicitud. El gestor de bloqueos mantiene una estructura de datos, denominada tabla de bloqueo.

Tabla de bloqueo: la utiliza el gestor para registrar los bloqueos concedidos y las solicitudes pendientes. La tabla de bloqueos generalmente se implementa como una tabla asociativa en memoria, indexada de lista, sobre el nombre del elemento de datos que se está bloqueando.

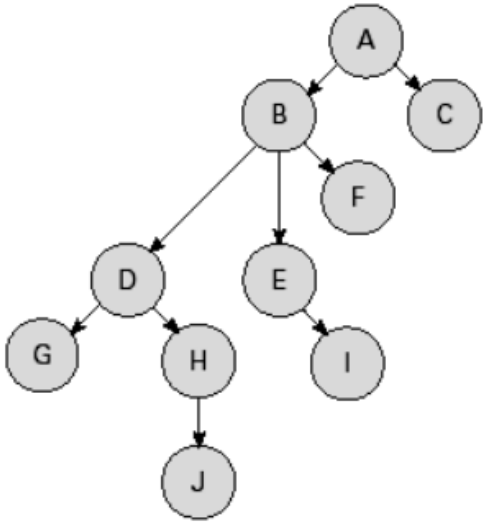
_ El motor de la base de datos no solamente esta accediendo a la información, sino que también tiene sus propias herramientas para controlarla. Tiene sus propias tablas donde va registrando que cosas esta bloqueando o desbloqueando en tal momento. Esto ayuda a que la base de datos sea más consistente y más sólida.

Alternativas de protocolos

_ Cuando el protocolo de bloqueo de dos fases no responde a la necesidad, tenemos las siguientes alternativas:

Protocolo basado en grafos: lo único que se necesita es un conocimiento previo a cerca del orden en el que se accede a los elementos. Este protocolo impone un orden parcial sobre un conjunto D que contiene todos los elementos, en donde va a avanzar de dato en dato hasta llegar al último. Para entender mejor, analizamos un protocolo árbol, en donde en este protocolo sólo se permiten los bloqueos exclusivos y los compartidos se dejan de lado. El primer bloqueo de cualquier transacción puede ser sobre cualquier elemento de datos. Posteriormente, una transacción T_i puede bloquear un elemento de datos Q sólo si T_i está bloqueando actualmente al padre de Q, es decir, solamente se puede bloquear un elemento después del primero si la transacción esta en ese momento bloqueando al padre, por ejemplo, si D esta bloqueado y queremos bloquear J, tenemos que bloquear primero a H y después a J, ya que no se puede directamente a J. También, los elementos

de datos se pueden desbloquear en cualquier momento, sin embargo una vez que una transacción desbloqueo alguno de los datos, ya no los puede tocar más.



_ Este protocolo asegura la secuencialidad en cuanto a conflictos y también asegura la ausencia de interbloqueos (por ende no se necesita retroceder), porque nunca va a haber un caso en el que todas las transacciones vayan a quedar en espera, ya que al ser acíclico la transacción en algún momento se va a quedar sin datos para poder bloquear. Tiempos de espera menores y a hay un aumento de la concurrencia, al ser de una sola fase por lo que los desbloqueos se pueden producir antes. Pero este protocolo no garantiza la recuperabilidad y la ausencia de cascadas, porque si una transacción que escribió sobre un dato se compromete y una que había leído antes se aborta, el dato ya quedo modificado permanentemente.

Protocolo basado en marcas temporales: en vez de establecer un orden entre los elementos, se establece un orden entre las transacciones. Para que este funcione, hace que cada transacción lleve una marca temporal que se asigna cuando se introduce la transacción en se introduce en el sistema. Si a la transacción T_i se le ha asignado la marca temporal $MT(T_i)$ y una nueva transacción T_j entra en el sistema, entonces $MT(T_i) < MT(T_j)$. El protocolo maneja la ejecución concurrente de modo que las marcas temporales de las transacciones determinan el orden de secuencia. Para asegurar dicho comportamiento, el protocolo mantiene por cada elemento de datos Q dos valores de marca temporal:

- marca_temporal-E(Q): va a tomar la transacción más grande que escribió sobre Q.
- marca_temporal-L(Q): va a tomar la transacción más grande que escribió leyó Q.

_ El protocolo de ordenación por marcas temporales tiene las siguientes características:

- Asegura que todas las operaciones de leer y escribir conflictivas se ejecuten en el orden de las marcas temporales.
- asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar.
- Existe una posibilidad de inanición de las transacciones largas si una secuencia de transacciones cortas conflictivas produce reinicios repetidos de la transacción larga.
- Pero puede que la planificación no esté libre de cascadas e, incluso, es posible que no sea recuperable.

Regla de escritura de Thomas: es una versión modificada del protocolo de ordenación por marcas temporales, que permite una mayor concurrencia potencial ya que las operaciones escribir obsoletas se pueden ignorar bajo determinadas circunstancias. Por ejemplo, cuando la transacción T_i intenta escribir el elemento de datos Q, si la marca temporal $MT(T_i) <$ marca_temporal-E(Q), entonces T_i está intentando escribir un valor obsoleto de {Q}. Por lo tanto, en vez de retroceder la transacción T_i como lo habría hecho el protocolo de ordenación por marcas temporales, esta operación "escribir" se puede ignorar. La regla de escritura de Thomas permite una mayor concurrencia potencial, y además permite algunas planificaciones secuenciables en cuanto a vistas que no son secuenciables en cuanto a conflictos.

Protocolos basados en validación: es un esquema que evita automáticamente los retrocesos en cascada mediante la ejecución de las transacciones en tres fases:

- Fase de lectura: todas las operaciones de escribir se realizan sobre variables locales temporales sin actualizar la base de datos actual.
- Fase de validación: la transacción lleva a cabo una "prueba de validación" para determinar si se puede escribir en las variables locales sin violar la secuencialidad.
- Fase de escritura: si la fase de validación tiene éxito, las actualizaciones se aplican a las bases de datos; en otro caso, la transacción retrocede.

_ Las tres fases de las transacciones que se ejecutan concurrentemente se pueden intercalar, pero cada transacción debe seguir las tres fases en ese orden. Por ejemplo, suponemos que la fase de validación y la de escritura se producen a la vez, atómica y serialmente, es decir, sólo se ejecuta una transacción de validación/escritura a la vez. A este protocolo se lo denomina también control de concurrencia optimista ya que la transacción se ejecuta por completo con la esperanza de que todo irá bien durante la validación.

_ Se determina el orden de secuencialidad a través de las marcas temporales dadas en el momento de la validación, para aumentar la concurrencia. De este modo a la marca temporal MT(T_i) se le da el valor de Validación(T_i). Este protocolo es útil y proporciona un alto grado de concurrencia si la probabilidad de conflictos es baja, y esto se debe a que el orden de secuencialidad no está predecidido, y en el caso de que no funcione, solo un número inferior de transacciones tendrán que retroceder.

Planificación producida por validación: en el siguiente ejemplo de planificación se consideran dos transacciones, en donde en la transacción 14 se levanta el dato de cuanto tenía la cuenta de B, mientras en la transacción 15, alguien le saco 50 a la cuenta B y se los paso a la cuenta A. Luego en la transacción 14 se lee cuanto tiene la cuenta de A, y cuando sumamos las cuentas, el resultado va a ser mayor de lo que hay en el banco.

T_{14}	T_{15}
leer(*B*)	
	leer(*B*) *B:- B-50* **leer**(*A*) *A:- A+50*
leer(*A*) (*validar*) **visualizar** (*A+B*)	
	(*validar*) **escribir** (*B*) **escribir**(*A*)

Granularidad múltiple

_ En los esquemas de control de concurrencia que se describieron antes, se ha tomado cada elemento de datos individual como la unidad sobre la cual se produzca la sincronización. Hay circunstancias en la que puede ser conveniente agrupar los elementos de datos y tratarlos como una unidad individual de sincronización. Por ejemplo, si la transacción tiene que acceder a toda la base de datos, entonces la transacción debe bloquear cada elemento de la base de datos y ejecutar estos bloqueos produce una pérdida de tiempo, porque seria mejor que la transacción pueda realizar una única petición de bloqueo para bloquear toda la base de datos. Por otro lado si la transacción necesita acceder solo a unos cuantos datos, no seria necesario bloquear toda la base de datos ya que en este caso se perdería la concurrencia, por lo que es necesario un mecanismo que permita al sistema definir diferentes niveles de granularidad.

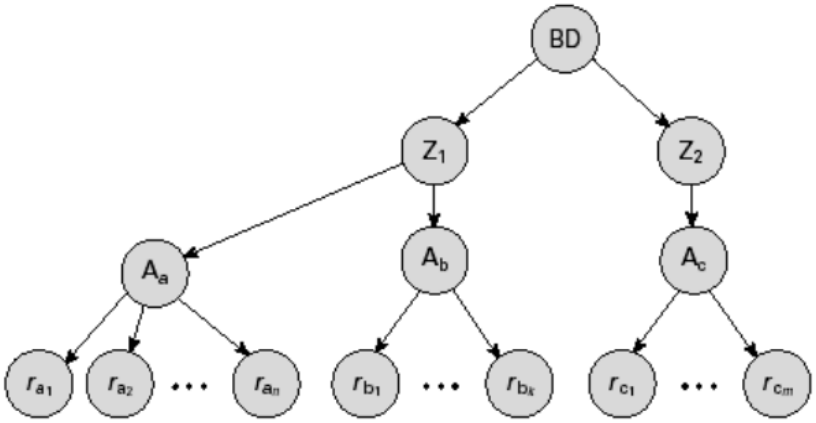
_ El sistema de granularidad múltiple permite que los elementos de datos sean de diferentes tamaños y permite definir una jerarquía de granularidad de datos, donde las granularidades pequeñas están anidadas en otras más grandes. Se puede representar

gráficamente como un árbol (aunque no debe confundirse con el protocolo de bloqueo de árbol). Cuando una transacción bloquea un nodo del árbol explícitamente, implícitamente bloquea todos los descendientes del nodo de la misma manera.

Granularidad de bloqueo: se refiere al nivel del árbol en el que se realiza el bloqueo:

- Granularidad fina: es en los niveles inferiores del árbol, la cual tiene alta concurrencia y alta sobrecarga de bloqueo.
- Granularidad gruesa: es en los niveles superiores del árbol, la cual tiene baja sobrecarga de bloqueo y baja concurrencia.

_ A continuación, vemos un diagrama en donde se refleja la jerarquía de granularidad, en donde la raíz del árbol seria la base de datos de por ejemplo el banco, los Z representan un área o sucursal del banco, los A representan archivos y podrían ser los clientes del banco, y los R representan los archivos que podrían ser los datos de los clientes.



Modos de bloqueo intencional: Además de los modos de bloqueo compartido y exclusivo, existen tres modos de bloqueo adicionales con granularidad múltiple:

- Intencional-compartido (IC): indica un bloqueo explícito en un nivel inferior del árbol, pero sólo con bloqueos en modo compartido.
- Intencional-exclusivo (IX): indica un bloqueo explícito en un nivel inferior del árbol, pero sólo con bloqueos en modo exclusivo o compartido.
- Intencional-exclusivo y compartido (IXC): el subárbol cuya raíz es ese nodo se bloquea explícitamente en modo compartido, y dicho bloqueo explícito se produce en un nivel inferior con bloqueos en modo exclusivo.

_ los bloqueos intencionales permiten que se bloquee un nodo de nivel superior en modo C o X sin tener que comprobar todos los nodos descendientes.

_ A veces necesitamos realizar alguna operación muy grande en la base de datos, como por ejemplo, reconstruir un índice en la base de datos porque algo se ha dañado, hacer alguna revisión del contenido de la base de datos por alguna cuestión en especial como un cierre de balance del banco, en donde los saldos que están en ese momento no pueden sufrir ningún cambio ni desorden. Podemos hacer un bloqueo en la base de datos, pidiendo un bloqueo exclusivo, en donde si hay gente que la esta usando, esperamos a que la termine de usar, si hay gente que pide otros accesos no se los damos inicialmente, y a medida que van terminando todos los que la estaban usando, se van liberando y tomamos el control. Ahora, realizamos la acción que queramos o necesitemos en la base de datos, como por ejemplo para obtener un determinado reporte, obtener un balance o hacer un cambio grande en la base de datos, y luego la libero. Son muy pocos los momentos en los que esto se produce, ya que esto molesta a los usuarios impidiéndoles retirar plata en ese momento, ver su saldo o pagar impuestos, por ende estas acciones grandes se hacen en horarios específicos y fuera de lo operativo del banco, y se genera un bloqueo, general, en una sucursal, etc. Son operaciones muy especificas que quieren capturar toda la base o un bloque muy grande de la base.

Esquemas multiversión

_ Estos esquemas consisten en crear varias versiones de los datos y mantener cada una de estas en memoria a lo largo del tiempo. Podemos tener tanto la ordenación por marcas temporales multiversión como el bloqueo de dos fases multiversión. La manera en la que se crean estos datos es mediante operaciones de escribir, y a cada una de estas versiones es etiquetarlas con marcas temporales. Cuando se quiere hacer una operación leer(Q), seleccionamos la versión apropiada de Q basándonos en las marcas temporales de la transacción, y se devuelve el valor de la versión elegida. Las operaciones leer no tienen que esperar nunca ya que se devuelve la versión apropiada inmediatamente.

Ordenación por marcas temporales multiversión: empezamos asumiendo que cada dato Q va a tener Q_k versiones, en donde a cada versión k, se le indica que tiene tres campos:

- Dato o contenido
- marca_temporal-E(Q_k): es la marca temporal de la transacción que haya creado o escrito la versión.
- marca_temporal-L(Q_k): es la mayor marca temporal de todas las transacciones que hayan leído con éxito la versión.

_ Cuando una transacción crea una versión nueva Q_k de Q,, la marca_temporal-E y la marca_temporal-L de Qk se inician con la marca temporal de la transacción que la escribió. Y la marca temporal se actualiza cada vez que una transacción con una mayor marca temporal la lee.

Bloqueo de dos fases multiversión: vamos a distinguir entre dos tipos de transacciones:

- Transacciones de actualización: adquieren bloqueos de lectura y escritura, y mantienen todos los bloqueos hasta el final de la transacción, en donde se dice que estas transacciones siguen un bloqueo de dos fases riguroso. Cuando se escribe un dato, siempre se va a generar una nueva versión y a esta versión se le va a agregar una marca temporal, dependiente de un contador llamado contador-mt que se incrementa durante el procesamiento del compromiso.
- Transacción de sólo lectura: siguen el protocolo de ordenación por marcas temporales multiversión para llevar a cabo las lecturas. En este caso se les asigna una marca temporal leyendo el valor actual del contador-mt, antes de que inicien la ejecución.

Tratamiento de interbloqueos

_ Decimos que un sistema está interbloqueado si existe un conjunto de transacciones tales que cada una de ellas están esperando a otra del mismo conjunto. Para prevenir estos interbloqueos tenemos los protocolos de prevención de interbloqueos, en donde aseguran que el sistema nunca llega a un estado de interbloqueo. Algunas estrategias de prevención son:

- Predeclaración: consiste en hacer que cada transacción bloquee todos sus elementos de datos antes de que comience a ejecutarse, por lo que no se va a poder ejecutar hasta que no tenga todos los datos.
- Protocolo basado en grafos: permite que una transacción solo pueda ser bloquear en un orden previamente especificado.

_ Por mas que alguien haya bloqueado, se deja una marca del estado anterior que tenia por ejemplo una cuenta bancaria, y se lo añadimos como dato para no depender de los impactos que estamos teniendo. Tratamos de habilitar para el que quiera leer, pueda hacerlo sin sentir el efecto de que se este produciendo algún cambio. Esto permite que muchos usuarios sean concurrentes a la base de datos, siempre y cuando no quieran modificarla. Los saldos parciales pueden tener algún error ya que fueron cambiando mientras se estaban leyendo, pero no perdemos tiempo o no nos demoramos.

Otras estrategias de prevención de interbloqueos

Esquema esperar-morir: en este caso la transacción más antigua puede esperar a la nueva para liberar el elemento de datos. Por otro lado, las transacciones más nuevas nunca esperan por las más antiguas, en vez de ellos se retroceden. Una transacción puede morir varias veces antes de que adquiera el elemento de datos necesario.

Esquema herir-esperar: en este caso la transacción más antigua hiere o fuerza el retroceso de la nueva en vez de esperarla. Las transacciones más nuevas esperan a que las más

antiguas terminen. Puede que se produzcan menos retrocesos que en el esquema anterior.

_ En los dos esquemas anteriores, las transacciones retrocedidas se reinician con su marca temporal original. De este modo, las transacciones más antiguas tienen precedencia sobre las más nuevas y, por lo tanto, se impide que se produzca la inanición (asfixia).

Esquemas basados en limite de tiempo: una transacción espera un bloqueo sólo durante un período de tiempo, en donde al pasar este tiempo, la espera expira y se retrocede la transacción. Por lo que los interbloqueos no son posibles. Este esquema es fácil de implementar, pero existe la posibilidad de que aparezca la inanición. También es complicado determinar un buen valor para el intervalo de límite de tiempo.

_ Muchas veces podemos asignar en los motores de base de datos, el tiempo de demora de una transacción, en donde estos limites de tiempo son opcionales. Si pasa algo en el medio de la transacción que produzca demora o nos pasamos del tiempo, retrocedemos.

Detección de interbloqueos:

_ Los interbloqueos se pueden describir como un grafo de espera, que consta de un par:

- G = (V,A)

_ Donde V es un conjunto de vértices, es decir, todas las transacción del sistema, y A es el conjunto de arcos, donde cada elemento es un par ordenado $T_i \rightarrow T_j$. Existe un interbloqueo en el sistema si y sólo si el grafo de espera contiene un ciclo. Se debe invocar periódicamente a un algoritmo de detección de interbloqueos que busque un ciclo en el grafo.

Recuperación de interbloqueos:

_ Cuando se detecta un interbloqueo, tendrán que retroceder algunas transacciones para romper el interbloqueo y asi las otras puedan continuar, y para esto se debe seleccionar como víctima (la que retroceda) aquella transacción que incurra en un coste mínimo. El retroceso puede ser:

- Total: se aborta la transacción y luego vuelve a comenzar.
- Solamente retroceder la transacción lo necesario para romper el interbloqueo.

_ Si siempre se elige a la misma transacción como víctima, se produce la inanición, y para evitar esto se debe incluir en el factor de coste el número de retrocesos. Por ejemplo, a las tantas veces que retrocedimos, elegimos otra transacción.

Operaciones insertar y borrar

_ Cuando hacemos modificaciones, tenemos dadas de alta o eliminaciones, a veces hay conflicto y se utiliza el bloqueo de dos fases:

- Una operación borrar sólo se puede llevar a cabo, si la transacción que borra la tupla, tiene un bloqueo exclusivo sobre la tupla que se está borrando. Si alguien había leído o tiene un bloqueo anterior , no podemos eliminar una tupla hasta que no terminen las transacciones anteriores.
- A una transacción que inserta una tupla nueva en la base de datos se le proporciona el bloqueo exclusivo de dicha tupla, es decir, insertamos pero no tiene impacto.

_ Las inserciones y borrados pueden llevar al fenómeno fantasma, en donde si intentamos eliminar una tupla como el saldo de una cuenta bancaria, se produciría un fallo de una suma o un saldo que ya revisamos. Esto mismo pasa con una insersión, en donde si agregamos un nuevo saldo de una persona que acaba de ingresar al banco, el sistema no lo tiene en cuenta ya que es el más difícil de detectar, porque si estamos revisando el saldo de una cuenta bancaria, y nunca voy a haber mirado una cuenta que no existía si esta antes justamente no existía, ya que se esta cargando una nueva tupla que no fue considerada en la lectura.

_ La transacción que rastrea la relación lee la información que indica qué tuplas contiene la relación, al mismo tiempo que la transacción que inserta una tupla actualiza la misma información. Entonces la información debería bloquearse. Una solución a esto sería asociar un elemento de datos con la relación, para representar la información acerca de qué tuplas contiene la relación, luego las transacciones que rastrean la relación adquieren un bloqueo compartido en el elemento de datos, y por ultimo las transacciones que insertan o borran una tupla adquieren un bloqueo exclusivo en el elemento de datos. Esto quiere decir que al momento de insertar o borrar una tupla, tenemos que pedir un bloqueo mas fuerte para que no me lo otorguen si alguien había estado leyendo o no esos datos. Mientras estemos borrando e insertando elementos va a haber menos chances de concurrencia, es decir, poca gente va a estar accediendo a la base de datos en ese momento.

Protocolo de bloqueo de índices: lo que hace el motor de base de datos es, en lugar de controlar o bloquear las tuplas o nuevos registros que agregamos en la base de datos, lo que trabajamos acá es un bloqueo en el índice que nos lleva a esas nuevas tuplas. Si bloqueamos un nodo de un árbol y alguien quiere agregar o borrar un nodo debajo de este, no podría hacerlo porque esta bloqueado. Entonces, juega con el índice para que no se produzca el fenómeno fantasma, de aparición o desaparición de tuplas que no fueron consideradas en los bloqueos por la concurrencia con otros procesos.