UNA INTRODUCCIÓN GUIADA A LAS BASES DE DATOS RELACIONALES

Apuntes de la asignatura *Bases de Dades* del *Grau en Enginyeria en Informàtica* de la Universitat de Barcelona

Sergio Sayago
Departament de Matemàtiques i Informàtica
Universitat de Barcelona
Curso académico 2017/18

CONTENIDOS

1. Introducción

1.1 A la asignatura

Bases de Dades (de aquí en adelante, la asignatura) es una asignatura que se imparte en el segundo semestre del tercer año del *Grau en Enginyeria en Informàtica* de la Universitat de Barcelona.

La asignatura es importante porque las bases de datos tienen un papel destacado en nuestra vida diaria. Tal es su importancia que, algunas veces, se dice que los sistemas de bases de datos son críticos. ¿Porqué?

Es muy difícil imaginar la gran mayoría de los sistemas informáticos actuales, ya sean los relacionados con la banca[1] o con vuestra vida cotidiana (por ejemplo, redes sociales, repositorios de documentos, búsqueda y compra online, vuestros teléfonos móviles, el GPS del coche) sin un sistema de base de datos.

Otra razón de su importancia es que cuando la base de datos falla, el sistema, en general, para el usuario, también falla – aunque otras partes estén operativas – porque no podemos 'hacer nada'.

Por tanto, es importante que acabéis la carrera con conocimientos y habilidades que os permitan diseñar y analizar una base de datos, así como recuperar y almacenar datos en ella, de una manera relativamente rápida. Este es el principal objetivo de la asignatura.

1.2 A los Apuntes

Los Apuntes están dirigidos a los estudiantes de la asignatura. Escribir es un acto de comunicación. Cuando se escribe, normalmente se escribe 'algo' para 'alguien'. No obstante, lo anterior no necesariamente implica que otros estudiantes no puedan seguir / utilizar los Apuntes.

Los Apuntes muestran la estructura y principales contenidos de la parte de teoría de la asignatura. Los Apuntes se dividen en los siguientes temas: (a) bases de datos relacionales, (b) modelo relacional, (c) algebra relacional, (d) modelo Entidad-Relación, (e) transacciones y concurrencia, (f) normalización, (g) almacenamiento e indexación, e (h) introducción a bases de datos NoSQL. Los Apuntes se complementan con las presentaciones de teoría y con el Dossier de Prácticas.

Los Apuntes están inspirados, y siguen flexiblemente, todo un conjunto de libros de texto y cursos de bases de datos. Los libros seleccionados son un subconjunto de los recomendados en el plan de docente, más otros que se citan en las referencias de cursos de bases de datos que se imparten en universidades con gran prestigio a nivel internacional, como son Harvard[2], Cambridge[3],

[1] en los textos académicos, se ha convertido en una costumbre usar como ejemplo el caso de estudio de un banco (cuentas, clientes, préstamos…)
[2] http://sites.harvard.edu/~cscie66/
[3] https://www.cl.cam.ac.uk/teaching/1516/Databases/

Carnegie Mellon[4], y UCL (University College London)[5], para profundizar en los temas. Los Apuntes también están basados en mi propia formación y experiencia docente, especialmente la que adquirí como profesor de prácticas en la asignatura en el curso académico 2016/17.

Los Apuntes son una introducción guiada a las bases de datos relacionales. Son una *introducción* porque el campo de las bases de datos es muy amplio - resulta imprescindible hacer una selección de temas para tratarlos durante un semestre, y porque la asignatura es el primer (y, para algunos/as, el único) contacto que tenéis con las bases de datos en el grado. Son una introducción *guiada* porque he seleccionado aquellos temas que considero básicos e importantes para vuestra formación. Los Apuntes se centran en las bases de datos relacionales por su hegemonía en la literatura académica, en la que se citan como las más populares / usadas (por lo menos, por el momento).

Finalmente, por el propio carácter de las bases de datos relacionales, los Apuntes tienen 'punteros' a otras asignaturas, y complementan vuestra formación. Por ejemplo, el tema de álgebra relacional está relacionado con una parte de las matemáticas; el diseño Entidad – Relación tiene aspectos de Ingeniería del Software; la indexación en el tema del almacenamiento utiliza estructuras de datos avanzadas (árboles B+), y el concepto de bloqueo en las transacciones tiene cierta relación con la gestión de procesos de los sistemas operativos.

1.3 A los contenidos

El TEMA 1 es una visita guiada a las bases de datos. Define y discute aspectos tradicionales de las bases de datos relacionales y de su entorno, como la arquitectura de 3 niveles, los sistemas gestores de bases de datos, y su relación con los sistemas basados en ficheros, y también las sitúa dentro del contexto actual y de futuro. El TEMA 2 aborda aspectos clave del modelo relacional, como son las relaciones, claves, vistas, y restricciones de integridad, que es central en las bases de datos relacionales. El TEMA 3 se centra en el álgebra relacional, con el que tenemos un lenguaje formal de alto nivel para manipular las relaciones y construir nuevas. El álgebra relacional os debería ayudar a entender muchos aspectos de las consultas SQL (*Structured Query Language*) que veréis en la parte práctica. El TEMA 4 es sobre cómo representar una parte del mundo real en una estructura basada en entidades y relaciones (modelo Entidad-Relación) con la que podemos trabajar en una base de datos. El TEMA 5 aborda la normalización, que es una técnica de diseño para optimizar el diseño de una base de datos relacional. El TEMA 6 se centra en transacciones y concurrencia, que son aspectos muy relevantes para que los sistemas de bases de datos permitan a varios usuarios almacenar, recuperar y modificar datos sin hacer que la base de datos se transforme en un estado inconsistente. El TEMA 7 es sobre almacenamiento y, en especial, presenta los árboles

[4] http://15415.courses.cs.cmu.edu/fall2016/syllabus.html
[5] http://www.cs.ucl.ac.uk/current_students/syllabus/msccs/gc06_database_systems/

B+, una estructura de datos clave en los índices de las bases de datos. El TEMA 8 ofrece pinceladas de las bases de datos y el movimiento NoSQL.

Cada tema combina conceptos teóricos con ejemplos, que se combinan y amplían con el dossier de prácticas.

1.4 Bibliografía

- Connolly, T., Begg, C. (2005). *Database Systems. A Practical Approach to Design, Implementation, and Management*. Addison Wesley. 4th edition.
- Date, C (2001). *Introducción a los sistemas de bases de datos*. Pearson Educación.
- Elmasri, R., Navathe, S. (2016). *Fundamentals of Database Systems*. Pearson. 7th edition.
- Garcia-Molina, H., Ullman, J., Widom, J. (2009). *DATABASE SYSTEMS. The Complete Handbook*. Pearson. 2nd edition.
- Harrison, G. (2015). Next Generation Databases. NoSQL, NewSQL, and Big Data. Apress
- Kroenke, D., Auer, D. (2014). *Database Processing. Fundamentals, Design and Implementation*. Pearson. Hay una versión en castellano en la biblioteca.
- Mayer-Schönberger, V., Cukier, K. (2013). *BigData: a revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt.
- Ramakrishnan, R., Gehrke, J. (2003). *Database management systems*. McGraw- Hill. 3rd edition.
- Sadalage, P., Fowler, M. (2013). NoSQL Distilled. A brief guide to the emerging world of polyglto persistence. Addison-Wesley.
- Silberschatz, A., Korht, H., Sudarshan, S. (2002). *Fundamentos de bases de datos*. McGraw-Hill. 4ª edición.
- Sullivan, D. (2015). *NoSQL for Mere Mortals*. Addison-Wesley.

2. TEMA 1 - Una visita guiada a las bases de datos relacionales

Los objetivos de este tema son:

- Definir y entender conceptos claves para el resto de la asignatura.
- Entender las principales diferencias entre los sistemas basados en ficheros y en bases de datos.
- Conocer los principales componentes de los sistemas gestores de bases de datos (relacionales).
- Entender la arquitectura de tres niveles y la independencia de datos lógica y física.
- Conocer qué son los esquemas e instancias de una base de datos, y los mapeos.

2.1 Definiciones de conceptos clave

En los Apuntes y en la asignatura, las definiciones que utilizaremos de base de datos, Sistema Gestor de Base de Datos (SGBD) y sistema de base de datos son:

Una **base de datos** es un repositorio compartido de datos relacionados lógicamente y una descripción de esos datos (metadatos). Una base de datos se diseña para satisfacer las necesidades de información de una organización – en un sentido amplio.

Un **Sistema Gestor de Bases de Datos** es un software que nos permite definir, crear, mantener y controlar el acceso a una base de datos. Un SGBD relacional traduce sentencias SQL a acciones a realizar en la base de datos.

Un **programa de aplicación** es un intermediario entre el usuario y el SGBD. Recupera y modifica datos de la base de datos mediante sentencias SQL que se procesan en el SGBD.

Un **sistema de base de datos** es una colección de programas de aplicación que interactúan con el SGBD y con la base de datos.

La Figura 1 muestra una representación esquemática de un sistema de base de datos teniendo en cuenta las definiciones anteriores.

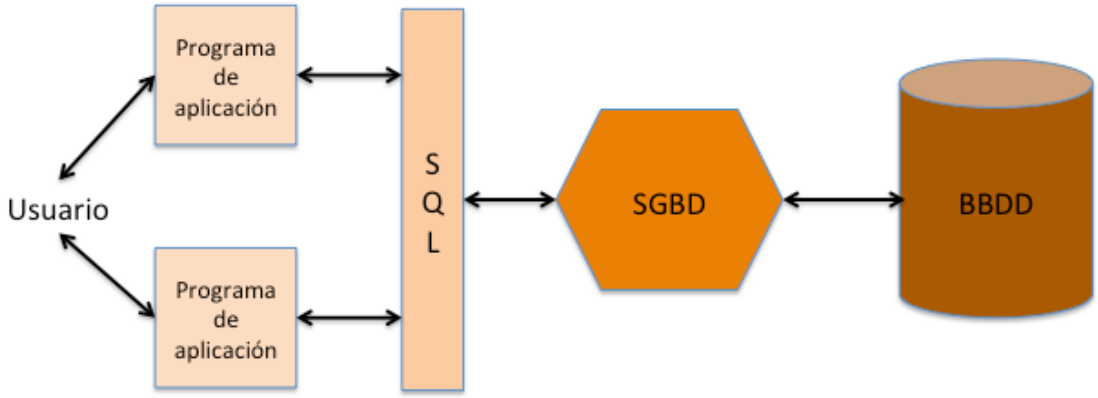


Figura 1: Sistema de bases de datos

2.2 Sistemas basados en ficheros versus bases de datos

Antes de comenzar a ver detalles de las bases de datos, creo que te resultará útil considerar su predecesor más inmediato - los sistemas basados en ficheros. De hecho, se ha convertido en una tradición en los textos académicos de bases de datos hablar de su predecesor, y en los Apuntes continuamos con esta tradición, porque te ayudará a entender mejor las bases de datos.

2.2.1 Sistemas basados en ficheros

Las bases de datos como las conocemos hoy (digitales) no siempre han existido. Antes de que llegaran los ordenadores, los datos se organizaban en ficheros (documentos) en papel. Como te puedes imaginar, consultar, recuperar, y almacenar datos en documentos en papel podía convertirse en una tarea bastante tediosa.

Con la llegada del ordenador, pasamos de ficheros manuales a ficheros digitales. Esto significó un cambio (muy) importante – tanto, que algunos pensaban que el papel desaparecía de las oficinas! Sin embargo, el sistema basado en ficheros trata de modelar el sistema de fichero manual en el ordenador, y esta decisión de diseño tiene una serie de limitaciones, especialmente desde el punto de vista de la asignatura. Veámoslo.

Imagínate que una empresa de alquiler de viviendas tiene cuatro ficheros en papel (clientes, propiedades en alquiler, oficinas, y trabajadores). Cada fichero es gestionado por un departamento. La empresa decide cambiar los ficheros en papel por ficheros digitales, y cada departamento, como parece normal, trabaja con su propio fichero. Además, como también parece lógico, cada departamento define su propio programa de aplicación para acceder y gestionar su propio fichero. ¿Se te ocurren algunos problemas derivados o limitaciones de este diseño?

La **integridad de los datos se puede violar**. Si se realiza algún cambio en el fichero de clientes, es probable que se tenga que actualizar el fichero de propiedades en alquiler, ya que son los clientes

los que las alquilan. Como cada departamento trabaja con su propio fichero, y con su propia aplicación, esta actualización no es nada trivial. Recuerda que la solución está descentralizada: cada departamento trabaja únicamente con su fichero, y lo hace, 'a su manera'.

La **realización de búsquedas es muy limitada**. Si queremos saber, por ejemplo, qué trabajador trabaja en qué oficina, parece necesario disponer de dos ficheros, el de oficinas, y el de trabajadores. Seguramente se te ocurra otra idea, que los datos de los trabajadores estén en el fichero de las oficinas, y viceversa. De esta manera, trabajamos con un solo fichero. Es una solución, pero genera, al menos, un problema, la duplicación de datos.

La **definición de los datos no es independiente de la aplicación**. Si la estructura del fichero (de cualquiera) cambia, por requerimientos de la empresa, la aplicación que interactúa con dicho fichero también se tiene que modificar, para que pueda realizar operaciones de lectura y escritura siguiendo el formato del fichero. La estructura del fichero está incrustada en el programa de aplicación.

2.2.2 Bases de datos

Si desgranas la anterior definición de una base de datos, verás que soluciona los principales problemas o limitaciones de los sistemas basados en ficheros.

Según la definición, una base de datos es un **repositorio**, no muchos ficheros separados. Los datos están representados de manera independiente del programa de aplicación. Una base de datos es un repositorio **compartido**, por varios usuarios. Por tanto, el departamento de clientes y el de gestión de propiedades en alquiler tienen acceso a los datos de los clientes y de las propiedades.

Una base de datos es una colección compartida de **datos relacionados lógicamente**. En otras palabras, no es una caja en la que dejamos objetos de cualquier manera. Los datos tienen una relación, y dichas relaciones también se guardan en la base de datos. Además de guardar datos, las bases de datos guardan **datos sobre los datos** (metadatos)[6]. Estos metadatos son muy importantes. Ejemplos de metadatos son las claves primarias y las foráneas, que veremos más adelante.

Una base de datos se diseña para **satisfacer las necesidades de información** de una organización. Las bases de datos pueden ser pequeñas o grandes, pero habitualmente se construyen con el objetivo de poder almacenar, recuperar y gestionar datos (clientes, productos, canciones de música,…) de manera eficiente y segura.

Las bases de datos también plantean retos – no iba a ser todo 'bueno' y fácil. Por ejemplo, uno de los que veremos en la asignatura es modelar el mundo real en un conjunto de tablas y relaciones, cosa que parece una simplificación brutal. Otro reto es ser capaz de controlar el acceso compartido a los mismos datos de manera que múltiples usuarios accedan a ellos de manera consistente.

[6] Las bases de datos también guardan otros elementos. Por ejemplo, índices, que son estructuras de datos que nos permiten ordenar y buscar información eficientemente; *triggers* o disparadores, que nos permiten forzar restricciones, …

2.3 Bases de datos relacionales a grandes rasgos

Los datos se guardan en tablas. Cada tabla tiene filas y columnas. Cada fila se corresponde con una instancia de un objeto o cosa de interés. Las columnas (o campos) son atributos de los objetos / cosas.

Las relaciones entre las tablas son muy importantes porque nos ayudan a modelar el problema, y también se guardan – en la misma estructura de las tablas, como verás más adelante - en las bases de datos relacionales. Por ejemplo, suponemos que tenemos tres tablas: ESTUDIANTE, ASIGNATURA y NOTA. Si la nota no estuviera relacionada con estudiante/s y asignatura/s, nos sería de muy poca utilidad. Además, normalmente queremos saber qué nota tiene un estudiante en una asignatura en concreto, un estudiante cursa más de una asignatura, etc. Las relaciones son importantes.

2.4 Sistema Gestor de Bases de Datos

Un SGBD nos permite definir, crear, mantener y controlar el acceso a una base de datos. Un SGBD interactúa con los programas de aplicación. Ejemplos de SGBD (Relacionales) son Microsoft Access, MySQL, y PostgreSQL.

2.4.1 Lenguaje de Definición y Manipulación de Datos

UN SGBD nos proporciona un Lenguaje de Definición de Datos (LDD) y un Lenguaje de Manipulación de Datos (LMD).

El LDD se utiliza para especificar / modificar el esquema (definición) de una base de datos. Con el LDD también podemos actualizar el diccionario de datos (o metadatos) de la base de datos. El resultado de utilizar el LDD es, simplificando, un conjunto de tablas.

El LMD nos proporciona operaciones para manipular los datos de una base de datos. Las operaciones de manipulación suelen consistir en operaciones de inserción, modificación, recuperación y eliminación de datos.

2.4.2 Algunas de las principales funciones de un SGBD

- Proporcionar funcionalidades para almacenar, recuperar y actualizar datos.
- Proporcionar soporte a las transacciones. Una transacción es una serie de acciones que se ejecutan en un bloque de manera atómica (a la vez, y todas o ninguna) y acceden a, o modifican, los contenidos de la base de datos. Añadir un nuevo cliente y modificar el salario de un trabajador son ejemplos de transacciones.
- Proporcionar servicios para controlar la concurrencia. Piensa que varios usuarios pueden acceder a una base de datos a la vez.
- Ser capaz de recuperarse tras un fallo, proporcionar servicios de autorización, de integridad de los datos (por ejemplo, podemos establecer restricciones: ningún trabajador puede gestionar

todas las propiedades) y de utilidad (cargar datos desde ficheros, análisis estadístico…), entre otros.

2.4.3 Arquitectura de tres niveles / capas

Uno de los principales objetivos de un sistema de bases de datos es proporcionar una vista abstracta de los datos a los usuarios, ocultando ciertos detalles de cómo los datos se almacenan y manipulan. La arquitectura de muchos SGBD está basada en la definida por el ANSI-SPARC (American Standards Institute – ANSI – Standard Planning and Requirements Committee – SPARC).

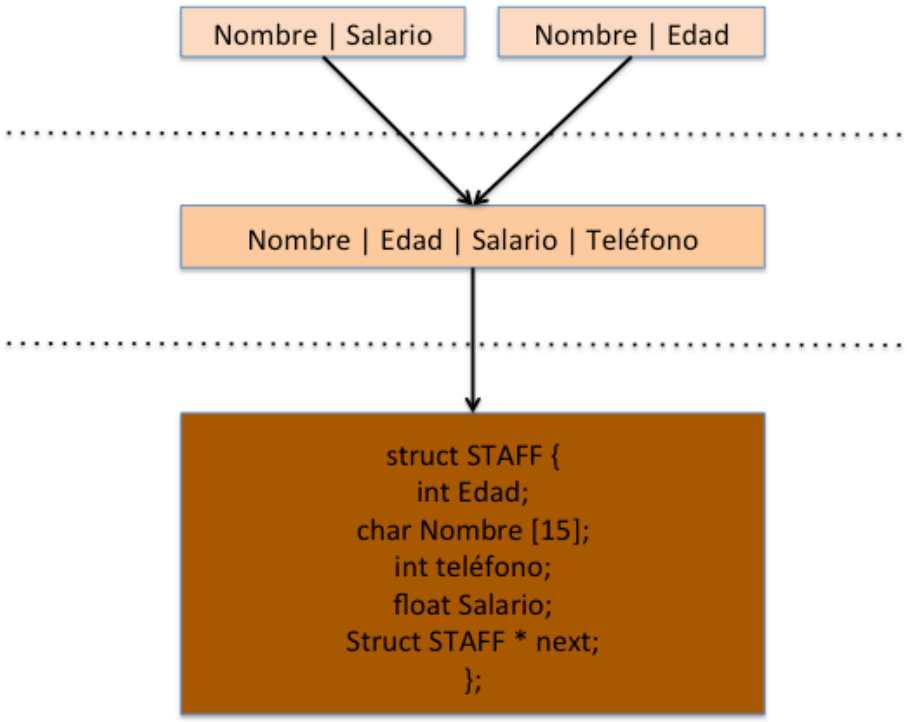


Figura 2: 3 niveles, 3 esquemas, independencia física y lógica

- El **nivel externo** se corresponde con la presentación de los datos al usuario. Consiste, o puede consistir, de diferentes vistas. Cada vista describe la parte de la base de datos que es relevante para un tipo de usuario. Una base de datos puede tener diferentes vistas.
- El **nivel conceptual / lógico** representa qué datos y relaciones están almacenados en la base de datos. Este nivel representa, en las bases de datos relacionales, entidades, atributos y relaciones, y restricciones, entre otros aspectos.
- El **nivel interno o físico** describe cómo están físicamente guardados los datos. Incluye estructuras de datos e interactúa con el sistema operativo.

2.4.4 Esquemas, instancias y mapeos

Los tres niveles dan pié a tres esquemas (o descripciones) de una base de datos (ver Figura 2).

El **esquema externo** se corresponde con las vistas y los usuarios. El **esquema conceptual** es el más importante para la asignatura, ya que define las entidades, atributos y relaciones. El **esquema interno** se corresponde con las estructuras de datos del nivel físico.

El esquema conceptual suele ser único y no suele cambiar, ya que define la lógica de la base de datos. Lo mismo para el esquema interno, aunque los cambios en este nivel sí pueden ser más frecuentes – por ejemplo, para intentar mejorar la eficiencia, se realizan mejoras / cambios en las estructuras de datos.

Podemos tener más de un esquema externo, ya que describe las vistas, y diferentes grupos de usuarios pueden tener vistas diferentes de la misma base de datos.

El SGBD es el responsable de realizar los mapeos entre los esquemas.

Una **instancia** (o ejemplar) de una base de datos se refiere a una "foto" de la base de datos en un momento concreto. Una base de datos tiene múltiples instancias, que se corresponden al mismo esquema.

2.4.5 Independencia de datos

La arquitectura de tres capas o niveles ayuda a proporcionar independencia de datos. Esto significa que niveles externos no se ven afectados por cambios en niveles internos.

Se distingue entre la independencia de datos lógica y física (ver Figura 2).

- La **independencia de datos lógica** se refiere a la inmunidad del esquema externo a cambios en el esquema conceptual. Claramente los usuarios para los que se ha hecho el cambio notarán los cambios, pero los otros usuarios no, necesariamente, ya que tenemos las vistas.
- La **independencia de datos física** se refiere a la inmunidad del esquema conceptual a cambios en el esquema interno. Cambios en el esquema interno son cambios en las estructuras de datos del sistema de ficheros (árboles, tablas de *hash*…). Para el resto de niveles superiores, cambios en el nivel físico deberían ser invisibles[7].

2.4.6 Estructura principal

Los sistemas de bases de datos los podemos dividir en dos componentes funcionales:

- El **gestor de almacenamiento** proporciona la interfaz entre los datos de bajo nivel y los programas de aplicación. Es el responsable de la interacción con el gestor de archivos y traduce las instrucciones LMD a órdenes del sistema de archivos.
- El **gestor de consultas** incluye el intérprete del LDD, el compilador (y optimizador) del LMD, y un motor de evaluación de consultas que ejecuta las instrucciones generadas por el compilador.

[7] Aunque el usuario puede percibir que el sistema va más rápido (o lento).

2.5 Programa de aplicación

Es un programa (software) con el que interactúan los usuarios. A través de estos programas, los usuarios acceden a la base de datos, y realizan operaciones (entre las principales) de almacenamiento, recuperación y consulta de datos. En la parte práctica del curso veremos un ejemplo de programa de aplicación con JAVA y JDBC. Pero seguro que puedes pensar en otros programas de aplicación - piensa en una web de compra y venta, o un buscador, en las app...

Los SGBD suelen trabajar con un gran volumen de datos, y no siempre todos los usuarios tienen que ver todos los datos. Algunas veces nos interesa poder proporcionar diferentes vistas de la misma base de datos. Las vistas se pueden crear con instrucciones SQL (CREATE VIEW) y visualizarse en los programas de aplicación.

2.6 Sistema de bases de datos

En resumen, un sistema de bases de datos está formado por **diferentes componentes**; algunos son hardware – los discos en los que los datos se guardan - otros son software, como programas; y otros componentes son los mismos datos, procedimientos, y las personas. Podemos tener diferentes perfiles de personas: los administradores, los diseñadores, los desarrolladores, y los usuarios finales.

2.7 Referencias

- Capítulo 1 de Silberschatz, Korth & Sudarshan
- Capítulo 1 y 2 de Connolly & Begg
- Capítulo 1 de Kroenke & Auer
- Capítulo 1 de García-Molina, Ullman & Widowm

3. TEMA 2 – El modelo relacional

Los objetivos de este tema son:

- Conocer qué es un modelo de datos.
- Conocer el modelo de datos relacional.
- Entender la estructura de datos relacional y su terminología.
- Conocer algunas propiedades importantes de las relaciones.
- Conocer y ser capaz de identificar superclaves, claves candidatas, claves primarias y foráneas.
- Conocer y entender las restricciones de entidad y referencial.
- Conocer aspectos básicos de las vistas.

3.1 El modelo relacional

El modelo relacional es un modelo de datos basado en registros (si quieres, piensa en registros como si fueran filas de una tabla).

Un **modelo de datos** es una colección integrada de los conceptos para describir y manipular datos, relaciones entre ellos, y las restricciones. Un modelo de datos **basado en registros** consiste en registros, que tienen un número fijo de campos, y de una longitud fija.

El **modelo relacional**[8] está basado en el concepto matemático de relación. El modelo relacional fue propuesto por Codd en el 1970. En el modelo relacional, todos los datos están lógicamente estructurados en relaciones, que son tablas bidimensionales (filas, columnas). Las tablas están relacionadas unas con otras. Cada relación tiene un nombre y se compone de atributos. El nombre de la relación y su conjunto de atributos forman el **esquema de la relación**, y el conjunto de esquemas relacionales es el esquema de la base de datos.

Un ejemplo de esquema de relación es el siguiente:

```
ESTUDIANTE (identificación, nombre, apellido, correo-e).
```

En mayúsculas indicaré el nombre de la relación (tabla). Entre paréntesis, indicaré los atributos.

El modelo relacional se mapea en el nivel externo y en el conceptual, pero no en el nivel lógico – en el que tenemos estructuras de datos que pueden tomar diferentes formas (arborescente, tabular…).

3.2 Estructura de datos relacional: terminología

Una **relación** es una tabla con filas y columnas. Una relación guarda información de los objetos a ser representados en la base de datos.

Un **atributo** es una columna de una relación. Los atributos describen propiedades o características de los objetos representados en la relación.

[8] Los antecesores del modelo relacional son el modelo de red y el modelo jerárquico, ambos modelos de datos basados en registros. Si quieres saber más sobre estos modelos, puedes consultar el capítulo 2 de Conolly & Begg.

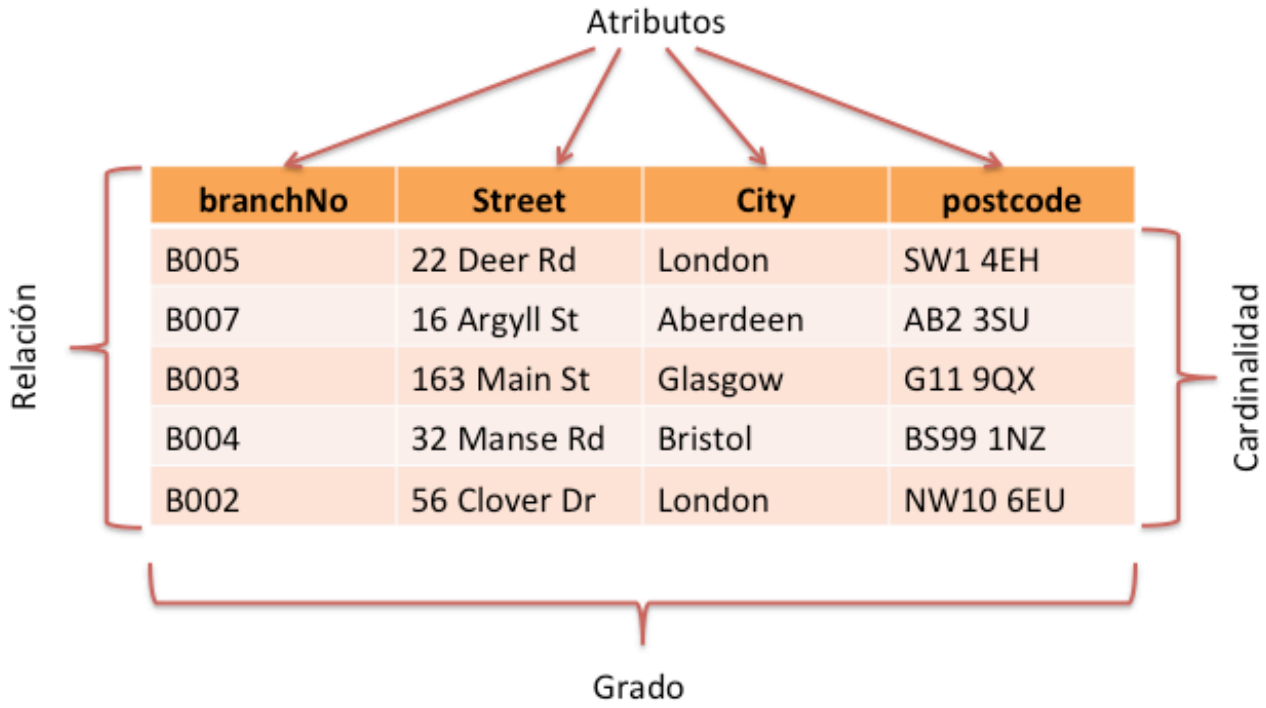
El **dominio** es el conjunto de posibles valores que puede tener un atributo o un conjunto de atributos. Cada atributo en una relación se define en un dominio. Un dominio puede ser compartido por diferentes atributos. Por ejemplo, el nombre de un empleado y el nombre de un cliente pueden pertenecer al dominio de nombres de personas, que son cadenas de caracteres de longitud variable.

Una **tupla** es una fila de una relación.

El **grado** de una relación es el número de atributos de una relación. Una relación de grado 1 tiene un atributo. Una relación binaria tiene 2 atributos.

La **cardinalidad** es el número de filas (tuplas) de una relación.

Veamos un ejemplo. Considera una base de datos de una empresa de gestión de propiedades de alquiler. Una de las relaciones en la base de datos es BRANCH – relación con datos de las oficinas (BRANCH).



branchNo	Street	City	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Figura 3: Relación BRANCH

La Figura 3 muestra la relación BRANCH. El grado de la relación es 4 porque tiene 4 atributos (el código de la oficina, la calle, la ciudad y el código postal). La cardinalidad de la relación – concretamente, de la instancia que se nos muestra en la figura – es 5, porque hay 5 filas / tuplas.

3.3 Algunas propiedades (importantes) de las relaciones

- Cada relación tiene un nombre que es diferente al nombre del resto de relaciones. No podemos tener dos relaciones con el mismo nombre.
- Cada fila es única, no hay filas duplicadas en una relación. En la Figura 3, cada fila representa a una oficina. No podemos tener la misma oficina duplicada.
- Las filas tienen datos de una entidad. En nuestro caso, cada fila tiene datos de una oficina.

- El orden de los atributos no tiene ningún significado.
- Los valores de una columna son siempre del mismo tipo / dominio. En la Figura 3, los valores de la columna ciudad no pueden ser, por ejemplo, números enteros.
- Cada atributo tiene un nombre único en cada relación. No obstante, varias relaciones pueden tener atributos con el mismo nombre. Por ejemplo, siguiendo con el ejemplo de la Figura 3, en la misma base de datos podemos tener una relación PROPIEDAD, con un atributo ciudad. Más adelante en la asignatura verás como se pueden distinguir atributos con el mismo nombre en varias relaciones.
- Cada valor de una celda tiene exactamente un único valor (se dice que son atómicos). No se permiten estructuras de datos como listas o vectores en las celdas.

Estas propiedades saldrán de manera implícita o explícita a lo largo de la asignatura. Te recomiendo que las tengas presentes.

3.4 Claves

Las claves son muy importantes en el modelo de datos relacional. Una de las propiedades de las relaciones es que no tenemos filas duplicadas. Por tanto, necesitamos ser capaces de identificar cada fila con un identificador único. En el mundo de las bases de datos, hablamos de **superclaves**, **claves candidatas**, **claves primarias** y **claves foráneas**.

3.4.1 Superclave

Una superclave es un atributo, o conjunto de atributos, que identifica unívocamente (y como mucho) a una fila. Una superclave puede tener atributos que no sean realmente necesarios. Por tanto, nos interesa identificar superclaves con el mínimo número de atributos.

Por ejemplo, considera la siguiente relación:

```
BRANCH (branchNo, street, city, postcode)
```

Una superclave para la relación BRANCH sería (branchNo, street). El conjunto de atributos formado por el identificador de cada oficina y la calle en la que se encuentra la oficina nos permite identificar unívocamente (y como mucho) a una oficina. Sin embargo, esta superclave tiene atributos que no son realmente necesarios. Si te fijas, con el identificador de la oficina ya tenemos suficiente para identificar unívocamente a cada fila en la relación.

3.4.2 Clave candidata

Una clave candidata es una superclave mínima. Con mínima me refiero a que tiene el mínimo número de atributos para identificar unívocamente, y como mucho, a una fila de la relación. Una clave candidata es irreducible. Es decir, un subconjunto suyo no tiene la propiedad de unicidad.

Por ejemplo, considera la siguiente relación:

```
BRANCH (branchNo, street, city, postcode)
```

El conjunto de atributos (branchNo, street) es superclave, pero no clave candidata, porque *branchNO* es clave candidata. Un subconjunto suyo (branchNo) tiene la propiedad de unicidad.

Veamos otro ejemplo. Considera la relación VIEWING:

`VIEWING (clientNo, propertyNo, viewData, comment)`

La relación VIEWING guarda las visitas que los clientes han hecho a las propiedades en alquiler. Para cada visita, guardamos la fecha y los comentarios del cliente.

El atributo *clientNo* no puede ser clave candidata, porque es razonable pensar que un mismo cliente pueda visitar más de una propiedad. Por tanto, *clientNo* no me permite identificar unívocamente, y como mucho, a una fila de la relación.

El atributo *propertyNo* no puede ser clave candidata, porque la misma propiedad puede ser visitada por más de un cliente. Por tanto, *propertyNo* no me permite identificar unívocamente, y como mucho, a una fila de la relación.

La combinación (clientNo, propertyNo) sí es clave candidata. ¿Puede haber otra? Si consideramos que un mismo cliente puede visitar la misma propiedad en diferentes fechas, necesitamos añadir otro atributo, y la clave candidata sería (clientNo, propertyNo, viewDate). Podemos tener más de una clave candidata.

3.4.3 Clave primaria

La clave primaria es la clave candidata seleccionada para la relación. La selección es del / de la diseñador/a - programador/a de la base de datos. No hay una receta, porque depende de los requerimientos y/o necesidades de información de la organización. Un ejemplo lo tienes en el caso anterior de las visitas a las propiedades en alquiler.

Una relación siempre tiene una (y solo una) clave primaria. ¿Porqué? Porque no podemos tener filas duplicadas, por definición. En el peor caso, la clave primaria de una relación está formada por el conjunto de todos sus atributos.

3.4.4 Clave foránea

Anteriormente hemos comentado que las tablas están relacionadas y que las relaciones se guardan. ¿Cómo? Las claves foráneas son atributos de una relación que se mapean en la clave primaria de otra relación. Cuando el mismo atributo aparece en más de una relación, recuerda que esto se puede hacer, normalmente representa una relación entre las relaciones.

Por ejemplo, considera las dos relaciones siguientes (concretamente, sus esquemas):

`BRANCH (branchNo, street, city, postcode)`

`STAFF (staffNo, name, surname, position, sex, DOB, salary, branchNo)`

En la relación BRANCH, el atributo *branchNo* es la clave primaria. Fíjate que está subrayada.

En la relación STAFF, el atributo *staffNo* es la clave primaria. Fíjate que está subrayada. El atributo *branchNo* está en cursiva. ¿Qué indica? Hay una relación entre oficinas y trabajadores. El

nexo, tal y como se ha diseñado la base de datos, es el identificador de oficina, que es clave foránea en la relación STAFF.

¿Porqué no está *staffNo* como clave foránea en BRANCH? Piensa un poco antes de seguir leyendo, por favor. Por una cuestión de diseño. En una oficina trabajan N trabajadores. Los trabajadores son entidades mucho más dinámicas (altas, bajas, cambios…) que las oficinas. Podemos tener un empleado sin oficina asignada; pero una oficina sin trabajadores…

3.5 Restricciones de integridad

Las claves primarias y foráneas tienen un papel muy importante en las relaciones, y se les imponen restricciones para garantizar su correcto diseño y funcionamiento. Estas restricciones reciben el nombre de restricciones de integridad, y son de dos tipos: de **entidad** y **referencial**. Un concepto importante para entender estas dos restricciones es el concepto de nulo.

3.5.1 Nulo

Nulo representa un valor para un atributo que es desconocido o no aplicable para la fila. Nulo NO es lo mismo que 0 o una cadena de caracteres vacía o con caracteres en blanco. Los ceros y los espacios en blanco son valores, y **nulo representa la ausencia de un valor**. Tiene un toque semántico. En la relación VIEWING, el campo de comentario puede ser nulo. Si no fuera así, nos forzaría a entrar un comentario – seguramente, falso o con poco sentido conceptual, cuando el cliente no hubiera hecho ningún comentario en la visita.

3.5.2 Integridad de entidad

Las claves primarias no pueden ser nulas. Si no fuera así, sería una contradicción con la propia definición de clave primaria. ¿A qué fila identificaría una clave primaria con valor NULL?

3.5.3 Integridad referencial

Si existe una clave foránea, su valor es nulo o el valor (tipo de datos incluido) de la clave primaria de la relación a la que hace referencia. El nombre de la clave foránea puede ser diferente al nombre de la clave primaria.

Si consideramos las relaciones STAFF y BRANCH, no tiene mucho sentido crear un trabajador y asignarle una oficina que no exista. Pero si que tiene más sentido (porque puede pasar) crear un trabajador y no asignarle una oficina – valor NULL, porque todavía no la tiene.

3.6 Vistas

En el modelo relacional, una vista es una relación virtual o derivada de relaciones bases**. Las vistas son relaciones que no existen** – no están en el modelo relacional - pero las podemos crear dinámicamente, y se guardan en los SGBD. Las vistas proporcionan **acceso personalizado y flexible**. Al limitar los datos con los que trabajan los usuarios, podemos controlar el acceso a los

datos y de esta manera, mejorar la seguridad de los mismos. Las vistas las veremos en mayor profundidad más adelante, especialmente en las prácticas.

3.7 Referencias

- Capítulo 3 de Conolly & Begg
- Capítulo 3 de Ramakrishnan & Gehrke
- Capítulo 3 de Garcia-Molina, Ullman & Widom
- Capítulo 2 de Silberschatz, Korht & Sudarshan

4. TEMA 3 – Álgebra Relacional

Los objetivos del tema son:

- Conocer las cinco principales operaciones de álgebra relacional: selección, proyección, unión, diferencia y producto cartesiano.
- Conocer otras operaciones de álgebra relacional que son muy útiles en las bases de datos relacionales: intersección, división y JOINS.
- Aprender cómo escribir expresiones en álgebra relacional para realizar consultas.

4.1 Introducción

En este tema comenzarás a manipular datos. Lo harás a través del álgebra relacional, que la aproximaremos como si fuera un lenguaje de alto nivel. El álgebra relacional nos permite construir nuevas relaciones a partir de las que ya existen.

El álgebra relacional nos proporciona las operaciones básicas que cualquier LMD necesita. SQL incorpora álgebra relacional, y muchas sentencias SQL las podemos ver como versiones '*user-friendly*' de expresiones de álgebra relacional.

Entonces, te puedes preguntar, ¿porqué tengo que estudiar álgebra relacional en la asignatura, si puedo hacer consultas directamente con SQL? Porque pienso que si eres capaz de formular las consultas en álgebra relacional con cierta soltura, entenderás mucho mejor las consultas SQL, y las sabrás formular con más facilidad y precisión. Tendrás una base y sabrás lo que está pasando '*behind the scene'*. Además, las expresiones en álgebra relacional nos permiten abstraernos de otras cosas y centrarnos en manipular las relaciones.

4.2 Álgebra relacional

El álgebra relacional es un lenguaje teórico con operaciones que trabajan sobre una relación o más de una para definir otra relación sin cambiar / modificar la/s relación/es base.

El resultado de una operación de álgebra relacional es una relación. Por tanto, el resultado de una operación puede ser la entrada de otra operación. Recuerda esto, porque será importante.

Las principales operaciones en álgebra relacional (ver Figura 4) son: **Selección**, **Proyección**, **Producto Cartesiano**, **Unión** y **Diferencia**. Además, también tenemos los **JOIN**, la **Intersección** y la **División**, que se pueden expresar en términos de las cinco operaciones básicas.

Las operaciones de Selección y Proyección son operaciones unarias, ya que operan sobre una única relación. Las otras operaciones son binarias.

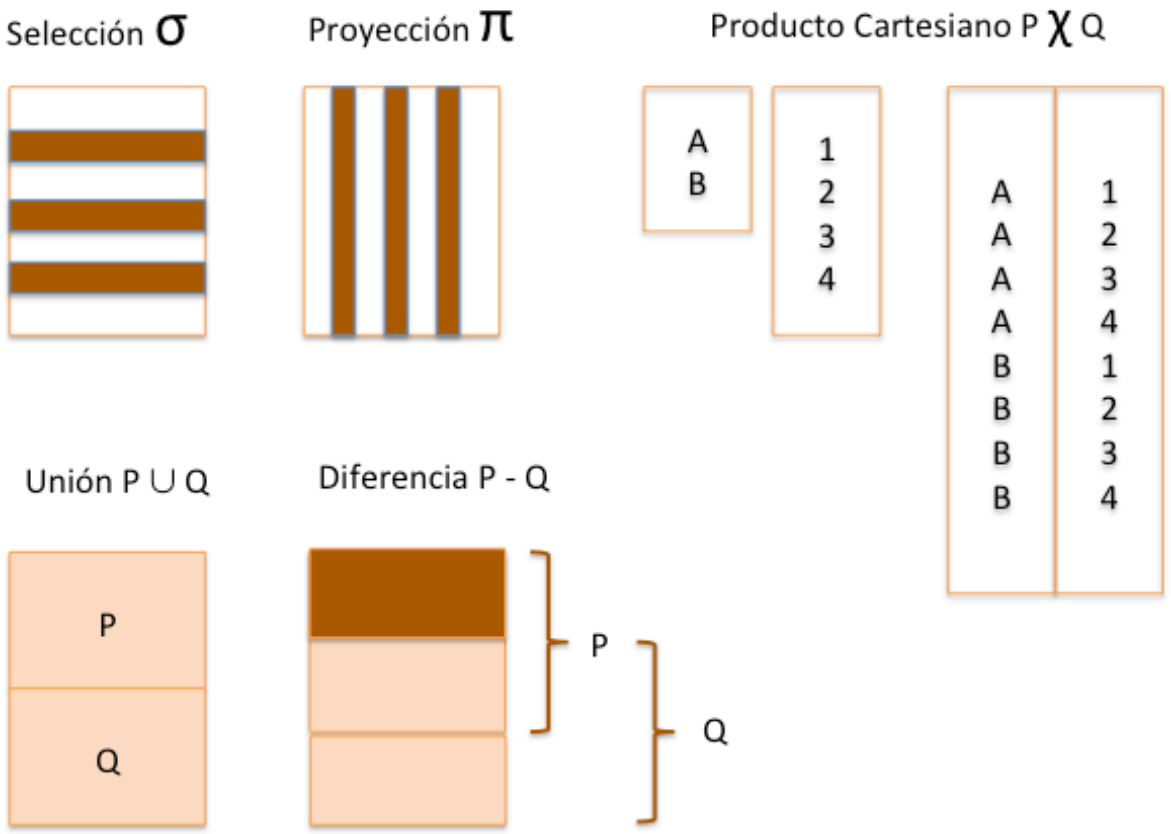


Figura 4: Operaciones básicas álgebra relacional

4.3 Principales operaciones

La Figura 4 muestra las operaciones básicas del álgebra relacional. Ahora veremos cada una de ellas en detalle, combinando definiciones con ejemplos.

4.3.1 Selección

$\sigma_{\text{predicado}}(R)$	La operación **selección** trabaja sobre una relación R y define una relación que contiene únicamente aquellas filas de R que satisfacen la condición del predicado

1) Para la relación STAFF que se muestra a continuación, escribe una operación de álgebra relacional que muestre todos los empleados con un salario ≥ 10.000.

staffNo	name	surname	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-6	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Relación 1: STAFF

La expresión en álgebra relacional es la siguiente:

$$\sigma_{\text{salary} \geq 10000}(\text{STAFF})$$

Te dejo como ejercicio construir la relación resultante de aplicar la operación de selección. Considera ahora la relación MOVIES:

title	year	length	genre	studioname
Star wards	1977	124	sciFi	Fox
Galaxy Quest	1999	104	comedy	DreamWorks
Wayne's World	1992	95	comedy	Paramount

Relación 2: MOVIES

2) Escribe una operación en álgebra relacional para mostrar las películas que tienen una duración de más de 100 minutos.

$$\sigma_{\text{length} > 100}(\text{MOVIES})$$

3) Escribe una operación en álgebra relacional para mostrar las películas que tienen una duración de más de 100 minutos y que sean de la Fox.

$$\sigma_{\text{salary} > 100 \text{ AND studioname='Fox'}}(\text{MOVIES})$$

4.3.2 Proyección

$\pi_{c1,c2,\ldots cn}(R)$	La operación **proyección** trabaja sobre una relación R y define una relación que contiene un subconjunto vertical de R, con los valores de los atributos que se especifican en la proyección, y elimina duplicados.

1) Escribe una operación de álgebra relacional que muestre el número de trabajador, nombre, apellido, y salario de todos los trabajadores. La relación es STAFF (Relación 1).

$$\pi_{\text{staffNo, name, surname, salary}}(\text{STAFF})$$

2) Escribe una operación de álgebra relacional que muestre el título, año y duración de las películas. La relación es MOVIES (Relación 2).

$$\pi_{\text{title, year, length}}(\text{MOVIES})$$

3) Escribe una operación de álgebra relacional que muestre el género de las películas. La relación es MOVIES (Relación 2).

$$\pi_{\text{genre}}(\text{MOVIES})$$

Por cierto, ¿cuántas filas tiene la relación resultante de aplicar esta proyección? Recuerda que la operación proyección elimina duplicados.

4.3.3 Unión

$R \cup S$	La operación **unión** de las relaciones R y S define una relación que contiene las filas de R, o de S, o de ambas, R y S. Las filas duplicadas se eliminan. R y S tienen que ser compatibles.

Fíjate que la unión es una operación binaria. La selección y proyección no.

La **compatibilidad** significa que los esquemas de R y S tienen el mismo número y tipo de atributos. Esto no implica que R y S tengan que ser iguales! R y S pueden ser el resultado de otras operaciones de álgebra relacional.

1) Escribe una operación en álgebra relacional que muestre todas las ciudades en las que haya bien una oficina o una propiedad para alquilar. Las relaciones son BRANCH y PROPERTY4RENT.

branchNo	*Street*	*City*	*Postcode*
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Relación 3: Relación BRANCH

propertyNo	*Street*	*City*	*Postcode*	*T*	*Rooms*	*Rent*	*ownerNo*	*staffNo*	*branchNo*
PA14	16 Holhead	Aberdeen	AB7 5SU	H	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	F	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	F	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	F	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	H	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	F	4	450	CO93	SG14	B003

Relación 4: Relación PROPERTY4RENT

$$\pi_{\text{city}}(\text{BRANCH}) \cup \pi_{\text{city}}(\text{PROPERTY4RENT})$$

4.3.4 Diferencia (R – S)

R - S	La operación **diferencia** define una relación que consiste de las filas que están en R pero no están en S. R y S deben ser compatibles. Nota: R – S no es lo mismo que S – R.

1) Escribe una operación en álgebra relacional que muestre todas las ciudades en las que hay una oficina pero no propiedades para alquilar.

$$\pi_{\text{city}}(\text{BRANCH}) - \pi_{\text{city}}(\text{PROPERTY4RENT})$$

4.3.5 Producto Cartesiano

R × S	El **producto cartesiano** define una relación formada por la concatenación de cada fila de R con cada fila de S.

La relación resultante tiene todos los posibles pares (R, S), formados por coger, en el mismo orden en el que aparecen en R y en S, como primer elemento del par, una fila de R, y como segundo elemento, una fila de S. ¿Se entiende? No te preocupes, con los ejemplos quedará más claro. Por tanto, si R tiene A filas, y S tiene B filas, el producto cartesiano produce una relación de A*B filas.

El esquema de la relación resultante es la unión de los esquemas de R y S. Si los esquemas tienen atributos con los mismos nombres, los desambiguamos precediendo el atributo con el nombre de la relación (por ejemplo, R.a, S.b).

1) Calcula el producto cartesiano de R y S:

R

A	B
1	2
3	4

S

B	C	D
2	5	6
4	7	8
9	10	11

R X S

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Figura 5: Producto cartesiano de R x S

2) Escribe una operación en álgebra relacional que muestre los nombres y comentarios de todos los clientes (y su código) que han visto una propiedad para alquilar. El código de la propiedad también se tiene que mostrar. Las relaciones son CLIENT y VIEWING.

clientNo	Name	Surname	telNo	prefType	maxRent
CR76	John	Kay	0207-774-1122	Flat	425
CR56	Aline	Stewart	01481-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

Relación 5: Relación CLIENT

clientNo	propertyNo	viewDate	Comment
CR56	PA14	24-May-04	Too small
CR76	PG4	20-Apr-04	Too remote
CR56	PG4	26-May-04	
CR62	PA14	14-May-04	No dinning room
CR56	PG36	28-Apr-04	

Relación 6: Relación VIEWING

La solución no es tan directa como en los casos anteriores. Veámoslo, de dentro a fuera:

$$\sigma_{\text{Client.clientNo = Viewing.clientNo}} (\pi_{\text{clientNo, name, surname (CLIENT)}} \times \pi_{\text{clientNo, propertyNo, comment (VIEWING)}})$$

Paso 1: hacemos una proyección sobre la relación CLIENT y VIEWING para obtener las columnas que queremos mostrar

Paso 2: realizamos el producto cartesiano de las relaciones resultantes de hacer las proyecciones

Paso 3: el resultado del paso 2 es una relación con más datos de los que necesitamos. Si te fijas, hay filas con diferentes códigos de clientes. Necesitamos hacer un filtro, una selección, con la condición de que el identificador del cliente esté en las dos relaciones.

4.4 Joins

Los JOINS son una operación fundamental en el álgebra relacional. Es una operación equivalente a la selección sobre el producto cartesiano.

4.4.1 Theta Join y Equijoin

$R \bowtie_F S$	El **theta join** define una relación con las filas que satisfacen el predicado F del producto cartesiano de R y S. Si F contiene =, entonces hacemos un **equijoin**.

Se calcula en dos pasos:

1) Producto cartesiano de R y S

2) Seleccionar en el producto cartesiano aquellas filas que cumplan la condición F

Como sucedía en el producto cartesiano, los atributos van precedidos por el nombre de la relación para desambiguar. Si es un equijoin, nos quedamos con uno de los atributos.

1) Escribe una operación en álgebra relacional que muestre todos los nombres (y código) y comentarios de todos los clientes que han visitado una propiedad para alquilar. Las relaciones son CLIENT y VIEWING.

$$R = \pi_{\text{clientNo, name, surname}}(\text{CLIENT})$$

$$S = \pi_{\text{clientNo, propertyNo, comment}}(\text{VIEWING})$$

$$R \bowtie_{\text{Client.clientNo = Viewing.clientNo}} S$$

2) Calcula el theta join de R con S, con la condición de A < D.

A	B	C
1	2	3
6	7	8
9	7	8

Relación 7: R

B	C	D
2	3	4
2	3	5
7	8	10

Relación 8: S

A	R.B	R.C	S.B	S.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

Relación 9: $R \bowtie_{A < D} S$

4.4.2 Natural Join

$R \bowtie S$	El **natural join** es un equijoin de R y S sobre todos los atributos comunes. Una ocurrencia de cada atributo común se elimina del resultado. En otras palabras, en la relación resultado, no tenemos dos campos con el mismo nombre – ni tampoco precedidos por el nombre de la relación.

Un natural join lo podemos descomponer en tres pasos

1)Producto cartesiano de R y S

2)Selección sobre los atributos del producto cartesiano con el mismo valor

3)Proyección

1) Escribe una operación en álgebra relacional que muestre todos los nombres y comentarios de todos los clientes que han visitado una propiedad para alquilar. Las relaciones son CLIENT y VIEWING.

$$\pi_{\text{clientNo, name, surname}}(\text{CLIENT}) \bowtie \pi_{\text{clientNo, propertyNo, comment}}(\text{VIEWING})$$

Fíjate que la solución es la misma que la te he mostrado en la sección 4.4.1 – la del theta join. La principal diferencia es que he omitido en el predicado (F) la condición de igualdad, porque hago un Natural Join – si no hay F, entonces se hace un Natural Join.

2) Calcula el Natural Join de R y S

A	B
1	2
3	4

Relación 10: R

B	C	D
2	5	6
4	7	8
9	10	11

Relación 11: S

A	B	C	D
1	2	5	6
3	4	7	8

Relación 12: R ⋈ S

El único atributo común entre R y S es B. Por tanto, en la relación resultante del Natural Join entre R y S, tenemos un único campo B. Al tratarse de un Natural Join, seleccionamos las filas en las que el valor de B sea el mismo en R y S, y proyectamos.

4.4.3 Outer Join

$R \bowtie S$	A menudo, cuando hacemos un JOIN, una fila en una relación no tiene una fila equivalente en la otra relación. Si queremos mostrar las filas que no tienen pareja, lo podemos hacer con un OUTER JOIN. Los valores que faltan en la segunda relación se ponen a NULL.

Se habla de LEFT OUTER JOIN y de RIGHT OUTER JOIN. En la definición, he utilizado el símbolo de LEFT OUTER JOIN. Entonces, las filas de R que no tienen pareja en S son las que se muestran, y los valores a NULL se colocan en el lado de S – porque no tienen pareja.

1) Escribe una operación en álgebra relacional que muestre un listado de las propiedades que (no) se han visitado. Las relaciones son PROPERTY4RENT (Relación 4) y VIEWING (Relación 6).

Fíjate que te piden todas las propiedades, tanto si se han visitado como si no. Por tanto, tienes que mostrar todas las propiedades, y añadir aquellos datos de la relación VIEWING si los hay. La solución es hacer un (LEFT) OUTER JOIN, de la siguiente manera:

$$\Pi_{\text{propertyNo, street, city}}(\text{PROPERTY4RENT}) \bowtie \text{VIEWING}$$

Para mostrar un subconjunto de los valores de PROPERTY4RENT he hecho una proyección.

4.5 Otras operaciones

4.5.1 Intersección

$R \cap S$	La **intersección** define una relación que consiste en el conjunto de filas que están tanto en R como en S. R y S tienen que ser compatibles.

1) Escribe una operación en álgebra relacional que muestre todas las ciudades en las que hayan oficinas y por lo menos una propiedad para alquilar. Las relaciones son BRANCH y PROPERTY4RENT.

$$\Pi_{\text{city}}(\text{BRANCH}) \cap \Pi_{\text{city}}(\text{PROPERTY4RENT})$$

Fíjate que podemos expresar la intersección en términos de la operación diferencia:

$$R \cap S = R - (R-S)$$

Te dejo como ejercicio que demuestres esta igualdad. Lo puedes hacer con diagramas.

4.5.2 División

R ÷ S	Considera dos relaciones, R y S, en la que R tiene dos campos (*x*, *y*) y S tiene uno, *y*, del mismo dominio que el de R. Definimos R ÷ S como un conjunto de valores x (en la forma de filas unitarias) tal que para cada valor de *y* en S, existe una fila (*x*, *y*) en R.

1) Considera las relaciones A, B1, B2, B3 de la Figura 6. Muestra el resultado de realizar la operación de división en álgebra relacional para los siguientes casos. Fíjate que la división busca *todas* las x (s1, s2…) en A que estén acompañadas de p2 (caso de B1), de p2 & p4 (caso de B2), y de p1 & p2 & p4 (caso de B3):

- A/B1
- A/B2
- A/B3

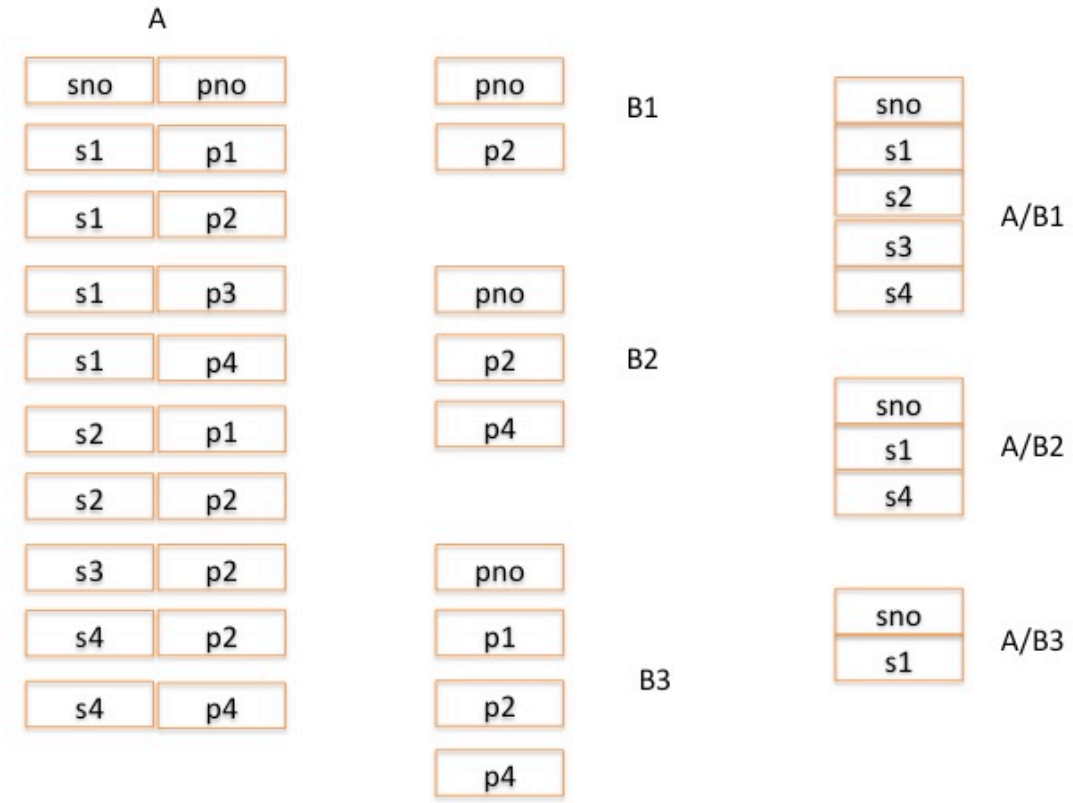


Figura 6: División

4.5.3 Renombramiento

A veces puede resultar útil – especialmente desde un punto de vista de comprensión – dividir las expresiones de álgebra relacional en expresiones más pequeñas. Para ello, tenemos el operador de renombramiento, ρ.

ρ (R, E), toma como entrada una expresión en álgebra relacional, E, y el resultado lo devuelve en R.

4.6 Combinar operaciones para construir consultas

Hasta el momento hemos realizado ejercicios para practicar las operaciones de álgebra relacional por separado. Ahora lo haremos en consultas – si quieres, ejercicios un poco más interesantes y difíciles.

1) Supongamos que queremos saber el título y el año de aquellas películas de la FOX que tienen una duración de por lo menos 100 minutos. Escribe la expresión en álgebra relacional para esta consulta. La relación es MOVIES.

Paso a paso.

1)Seleccionamos las películas que tienen una *length* ≥ 100

2)Seleccionamos las películas que tienen como *studioName* = 'Fox'

3)Realizamos la intersección de 1 y 2

4)Para mostrar el resultado final, hacemos una proyección sobre el resultado de 3 con los atributos *title* y *year*.

$$\Pi_{\text{title,year}} (\sigma_{\text{length}\geq 100} (\text{MOVIES}) \cap \sigma_{\text{studioName}=\text{'Fox'}} (\text{MOVIES}))$$

2) Tenemos las siguientes relaciones:

```
SAILOR (sid, sname, rating, age)
BOOKING (sid, bid, day)
BOAT (bid, bname, color)
```

Escribe la expresión en álgebra relacional, con y sin renombramiento, que muestre el listado de navegantes que han reservado la embarcación 103.

Paso a paso

1)Seleccionamos el barco con bid=103 de la relación BOOKING.

2)Realizamos un (NATURAL) JOIN con el resultado de 1 y SAILORS.

3)Con el resultado de 2, proyectamos sname.

$$\Pi_{sname}((\sigma_{bid=103}{}^{(BOOKING)}) \bowtie SAILOR))$$

Con renombramiento:

$$\rho(Temp1, \sigma_{bid=103}{}^{(BOOKING)})$$

$$\rho(Temp2, Temp1 \bowtie SAILOR)$$

$$\Pi_{sname}(Temp2)$$

4.7 Referencias

- Capítulo 4 de Connolly & Begg
- Capítulo 2 de Garcia-Molina, Ullman, & Widom
- Capítulo 4 de Ramakrishnan & Gehrke

5. TEMA 4- Entidad-Relación[9]

Los objetivos de este tema son:

- Conocer el modelo entidad-relación.
- Conocer qué es una entidad y un conjunto de entidades.
- Identificar atributos de las entidades y conocer los principales tipos de atributos.
- Conocer qué es una relación y un conjunto de relaciones.
- Conocer y saber identificar las restricciones de cardinalidad y de participación.
- Construir las claves primarias para entidades débiles y conjuntos de relaciones teniendo en cuenta restricciones de cardinalidad.
- Saber tomar decisiones de diseño en el momento de reducir un problema a un conjunto de entidades y relaciones.
- Conocer la representación tabular de los conjuntos de entidades y relación en un diagrama E-R.

5.1 Conceptos

El modelo **entidad-relación** (E-R) está basado en una percepción del mundo real en la que todo se reduce a objetos básicos (entidades) y a relaciones entre las entidades. Aunque es una simplificación considerable, el modelo ER es muy potente. Veamos ahora un poco de terminología y conceptos.

5.1.1 Entidad y conjunto de entidades

Una **entidad**:

- es una cosa u objeto del mundo real que es distinguible de todos los demás objetos. Una entidad puede ser un coche, una persona, un libro…No hay una receta; depende de la situación que estamos modelando,
- tiene un conjunto de propiedades (atributos) con sus respectivos valores,
- puede ser concreta o abstracta.

Un conjunto de entidades es un conjunto de entidades del mismo tipo que comparten las mismas propiedades o atributos.

Te has encontrado con entidades y conjuntos de entidades antes en los Apuntes. Si recuperamos, por ejemplo, la Relación 1: STAFF, podemos decir que STAFF es un conjunto de entidades, y que cada fila del conjunto, es una entidad.

[9] La herencia en E-R se trata en el Dosier de Prácticas.

staffNo	name	surname	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-6	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

5.1.2 Atributos y tipos

Una entidad se representa mediante un conjunto de atributos. Los atributos describen propiedades que cada miembro de un conjunto de entidades posee. Los atributos son las columnas en las relaciones.

Para cada atributo, hay un conjunto de valores permitidos: el **dominio**. Por ejemplo, el dominio del atributo nombre podría ser el conjunto de todas las cadenas de caracteres de una cierta longitud.

Los principales tipos de atributos son:

- **Simples y compuestos**. Los atributos simples no están divididos en partes. Los compuestos sí que lo están. El atributo dirección de un conjunto de entidades persona es un ejemplo de atributo compuesto (los componentes son nombre de la calle, piso, puerta…)
- **Monovalorados y multivalorados**. Los monovalorados sólo pueden tener un valor, mientras que los multivalorados, pueden tener un valor dentro de un conjunto de valores. Por ejemplo, una persona puede tener ninguno, uno o más de un número de teléfono[10].
- **Derivados.** El valor de este tipo de atributo se puede derivar de los valores de otros atributos. Por ejemplo, el número de préstamos se puede derivar del número de entidades préstamos asociadas a un cliente. Otro ejemplo de atributo derivado es la edad. Si tenemos un atributo que sea la fecha de nacimiento de una persona, podemos calcular su edad[11].

5.1.3 Relación

Una **relación** es una asociación entre diferentes entidades. Un conjunto de relaciones es un conjunto de relaciones del mismo tipo.

Podemos tener relaciones **recursivas** en la que se necesita hacer explícito el papel de cada ejemplar del conjunto de entidades que participa en la relación. Por ejemplo, los miembros del PDI

[10] Si leer esta definición te ha provocado cierta preocupación, es que seguramente la has relacionado con las propiedades (importantes) de las relaciones que se mencionan en el tema del Modelo Relacional. Un atributo multivalorado no es una celda en una tabla con más de un valor, porque incumpliría la propiedad de atomicidad del modelo relacional. Como verás más adelante, hay una estrategia para modelar atributos multivalorados – se convierten en relaciones / tablas.

[11] Tener un atributo edad no es buena decisión – porque lo tendremos que actualizar cada año – y porque la podemos calcular con la fecha de nacimiento y la fecha actual.

(Personal Docente Investigador) de una universidad pueden jugar varios papeles: supervisor, supervisado. La Figura 8 muestra un ejemplo.

La mayoría del conjunto de relaciones en un sistema de base de datos son binarias – participan dos entidades. Sin embargo, podemos encontrarnos con relaciones que involucran más entidades (si son 3, son relaciones ternarias…). Las relaciones m-arias se pueden transformar en relaciones binarias – lo verás un poco más adelante.

Se dice que las entidades **participan** en el conjunto de relaciones R.

Una relación también puede tener – de hecho, es habitual- **atributos descriptivos**. Por ejemplo, si tenemos el conjunto de entidades cliente y cuenta, se podría asociar el atributo fecha de acceso al conjunto de relaciones (impositor) para especificar la fecha más reciente en la que un cliente accedió a una cuenta. La Figura 8 muestra un ejemplo. Las entidades – mejor dicho, los conjuntos de entidades - son rectángulos, el conjunto de relaciones son rombos, y el atributo (descriptivo) es una elipse[12].

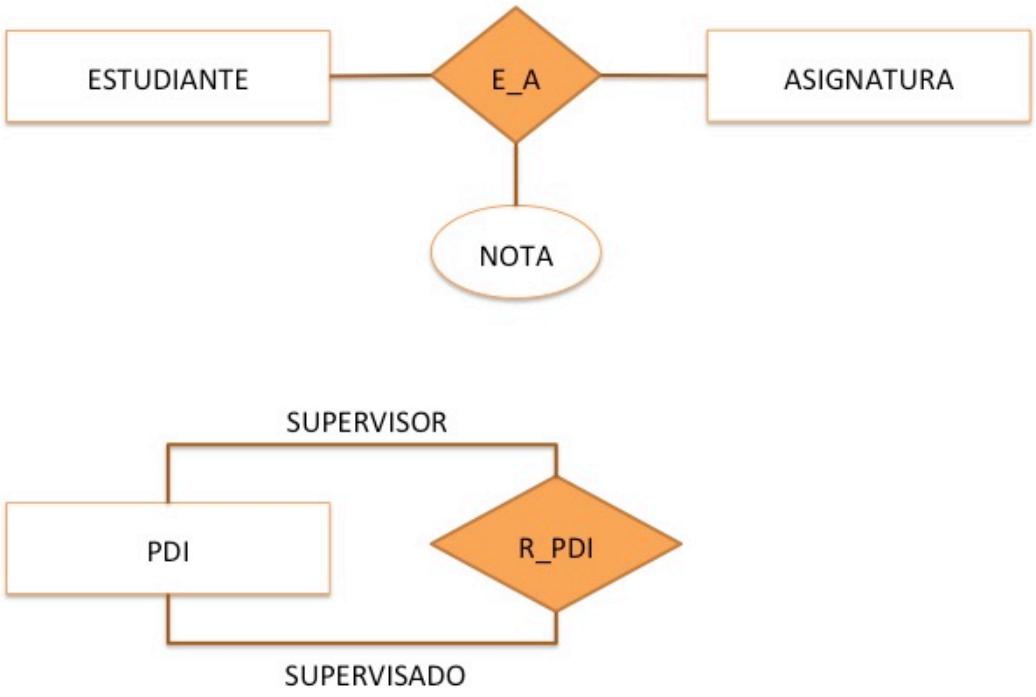


Figura 7: Relaciones, relaciones recursivas

[12] La notación que seguiremos para los diagramas de entidad-relación está esquematizada en los materiales del curso en el campus virtual.

5.2 Restricciones

Se definen dos categorías de restricciones: de cardinalidad y de participación. Ambas las extraemos habitualmente de la descripción del problema o situación a modelar.

5.2.1 Restricciones de cardinalidad

La restricción de cardinalidad expresa el número de entidades a las que otra entidad puede estar asociada mediante un conjunto de relaciones.

Para un conjunto de relaciones binarias R, entre los conjuntos de entidades A (izquierda en la figura) y B (derecha en la figura), la correspondencia de cardinalidad debe ser una de las siguientes (ver Figura 9). Fíjate que las correspondencias son bidireccionales.

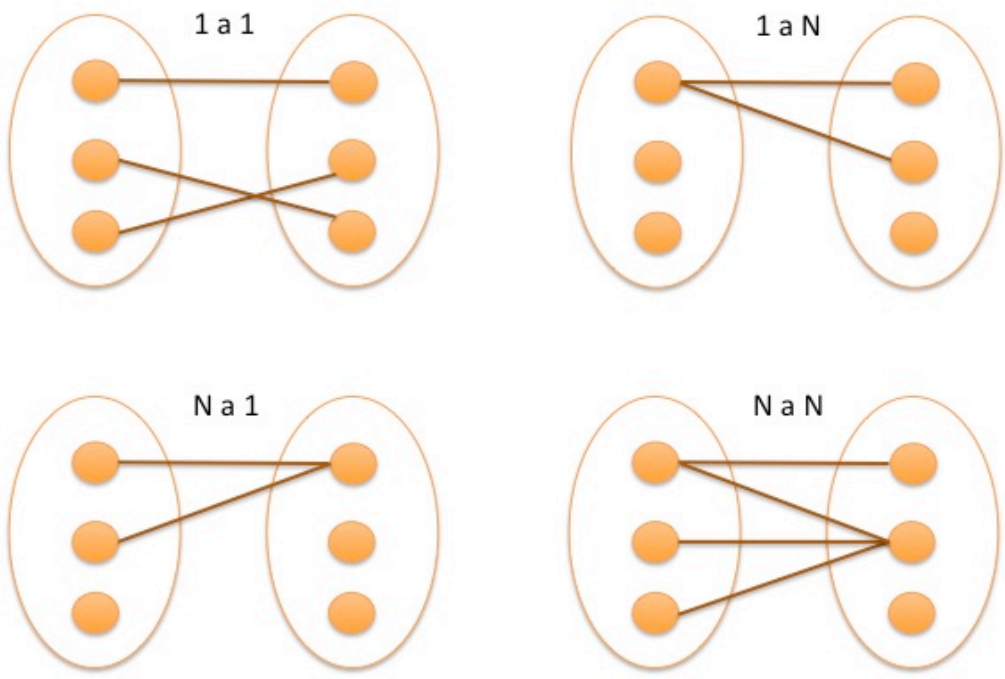


Figura 8: Restricciones de cardinalidad

1 a 1. Una entidad del conjunto de entidades A se asocia *a lo sumo* con una entidad del conjunto de entidades B, y una entidad del conjunto de entidades B se asocia *a lo sumo* con una entidad del conjunto de entidades A. Por ejemplo, podemos suponer una relación 1 a 1 entre TUTOR y GRUPO. Un TUTOR puede tutorizar a un GRUPO de estudiantes, y un grupo de estudiantes es tutorizado por un TUTOR.

1 a N. Una entidad del conjunto de entidades A se asocia con cualquier número (ninguna o varias) de entidades B, y una entidad del conjunto de entidades B, sin embargo, se puede asociar *a lo sumo* con una entidad del conjunto de entidades A. Por ejemplo, en una universidad tenemos DEPARTAMENTOS y PROFESORES. Un DEPARTAMENTO tiene varios PROFESORES. Sin embargo, un PROFESOR pertenece sólo (normalmente) a un DEPARTAMENTO.

N a 1. Una entidad del conjunto de entidades A se asocia *a lo sumo* con una entidad del conjunto de entidades B. Una entidad en B, sin embargo, se puede asociar con cualquier número de entidades (ninguna o varias) en A. Por ejemplo, podemos decir que un ALUMNO tiene un grupo de PRÁCTICAS (en una asignatura), pero un grupo de PRÁCTICAS está formado por varios ALUMNOS.

Quizás puedes entender N a 1 como el opuesto (o como algún estudiante me ha comentado, el inverso) de 1 a N. Si te ayuda, bien, pero fíjate en el orden: vamos de A a B, y en el significado de las restricciones de cardinalidad.

N a N. Una entidad del conjunto de entidades A se asocia con cualquier número (ninguna o varias) de entidades B, y una entidad en B, se asocia con cualquier número (ninguna o varias) de entidades A. Por ejemplo, un ESTUDIANTE puede realizar (ninguna), una o más de una ACTIVIDAD EXTRAESCOLAR, y en una ACTIVIDAD EXTRAESCOLAR, participan varios ESTUDIANTES.

Nota: las restricciones de cardinalidad las extraemos habitualmente de la descripción del problema o situación a modelar. Las relaciones entre los conjuntos de entidades suelen ser verbos; mientras que las entidades suelen ser sustantivos.

5.2.2 Restricción de participación

La participación de un conjunto de entidades E en un conjunto de relaciones R se dice que es **total** si cada entidad en E participa en al menos una relación en R. Si sólo algunas entidades en E participan en relaciones en R, la participación del conjunto de entidades E es **parcial**.

Por ejemplo, supongamos que tenemos el conjunto de relaciones prestatarios que relaciona el conjunto de entidades CLIENTE con el conjunto de entidades PRESTAMO. Podemos asumir que la participación de préstamos en prestatario es total: si existe un préstamo, éste tendrá un cliente asociado. En cambio, un cliente puede no tener un préstamo. Por tanto, la participación del conjunto de clientes cliente es parcial.

5.3 Claves

La clave primaria de un conjunto de entidades nos permite distinguir entre las diferentes entidades del conjunto E. Necesitamos un mecanismo similar para distinguir entre las diferentes relaciones de un **conjunto de relaciones**.

Dado unos conjuntos de entidades E_1, $E_2 \ldots E_n$, si el conjunto de relaciones R en el que participan no tiene atributos asociados, entonces el conjunto de atributos siguiente describe una relación individual en el conjunto R:

$$\text{clave-primaria } (E_1) \cup \text{clave-primaria } (E_2) \cup \ldots \text{ clave-primaria } (E_n)$$

Si el conjunto de relaciones R tiene atributos, a_1, $a_2 \ldots a_n$, entonces el conjunto de atributos siguiente describe una relación individual en el conjunto R:

$$\text{clave-primaria } (E_1) \cup \text{clave-primaria } (E_2) \cup \ldots \text{ clave-primaria } (E_n) \cup \{ a_1, a_2 \ldots a_n \}$$

En ambos casos, el conjunto de atributos {clave-primaria (E1) U clave-primaria (En)} es la **superclave para el conjunto de relaciones**. ¿Porqué la superclave y no la clave primaria? La estructura de la clave primaria depende de la correspondencia de cardinalidad asociada al conjunto de relaciones.

- Si la restricción de cardinalidad es **1 a 1**, **se puede usar cualquier clave primaria** (del lado de la izquierda o de la derecha) en el conjunto de relaciones. Si vuelves a considerar el ejemplo del conjunto de relaciones entre TUTOR y GRUPO - un TUTOR puede tutorizar a un GRUPO de estudiantes, y un grupo de estudiantes es tutorizado por un TUTOR – el conjunto de relaciones TU-GR relaciona tutores y grupos. Para identificar cualquier relación, basta con utilizar la clave primaria de TUTOR o de GRUPO, ya que cada tutor (grupo) participa únicamente en una relación.
- Si la restricción de cardinalidad es de **1 a N**, l**a clave primaria del conjunto de relaciones es la clave primaria del lado de la N**. ¿Se te ocurre porqué? Considera el conjunto de entidades DEPARTAMENTOS y PROFESORES de universidad. Un DEPARTAMENTO tiene varios PROFESORES. Sin embargo, un PROFESOR (normalmente) pertenece sólo a un DEPARTAMENTO. El conjunto de relaciones DE-PR relaciona parejas (departamento, profesor). ¿Qué distingue cada par en el conjunto de relaciones? Si un departamento tiene varios profesores, significa que voy a tener varias filas con la misma entrada para departamento. Sin embargo, como cada profesor pertenece a un solo departamento, no voy a tener a profesores duplicados. La clave primaria del PROFESORES (del lado de la N) identifica unívocamente, y como mucho, una fila del conjunto de relaciones DE-PR.
- Si la restricción de cardinalidad es de **N a 1, la clave primaria del conjunto de relaciones es la clave primaria del lado de la N**. ¿Se te ocurre porqué? El razonamiento es exactamente el mismo que el utilizado para el caso de a 1 a N.

- Si la restricción de cardinalidad es de **N a N, la clave primaria del conjunto de relaciones es la unión de las claves primarias de los dos conjuntos de entidades**. ¿Porqué? Porque cada entidad participa con cardinalidad N. Por tanto, para identificar unívocamente, y como mucho, una fila en el conjunto de relaciones, necesito la clave primaria de ambos conjuntos de entidades.

5.4 Entidades débiles

Una entidad débil **depende de otra** para su existencia. Su característica principal es que no puede ser identificada únicamente utilizando los atributos de la entidad débil. ¿Existen entidades débiles?

Por ejemplo, considera el caso de los despachos de los profesores en los departamentos de universidades. Se puede dar el caso – de hecho, yo me he encontrado – en que diferentes departamentos comparten la nomenclatura de los despachos. El código – si no me falla mucho la memoria – era Planta:Zona:Lado (derecha: par / izquierda: impar). Por tanto, si tenemos un código de despacho (1:A:12), es muy difícil saber a qué despacho nos estamos refiriendo. Necesitamos saber el departamento.

La clave primaria del conjunto de entidades débiles se forma **con la clave primaria del conjunto de entidades fuertes de las que depende, más el discriminante** (que lo podemos entender como el identificador – atributo o conjunto de atributos – de la entidad débil). En nuestro caso, sería el código del despacho, y su clave primaria seria:

$$\text{clave-primaria (departamento)} \cup \text{código despacho.}$$

5.5 Cuestiones de diseño

En la práctica, te encontrarás con algunas cuestiones de diseño.

5.5.1 ¿Entidades o atributos?

¿Qué constituye un atributo? Algunas veces podemos tener dudas. Podemos pensar en el caso del teléfono y una persona. Si modelamos el teléfono como una entidad, entonces un teléfono "tiene vida" – puede tener atributos, y nos interesa modelar esta información (ubicación, tipo…). También participa en relaciones – una persona tiene varios teléfonos – se insertan, eliminan.. Al tratarlo como entidad, es más general. En cambio, si lo tratamos como un atributo, no tiene "mucha vida", y no podemos asignarle mucha más información. Es un atributo que representa una característica de un conjunto de entidades. ¿Solución? Depende de la situación / escenario a modelar.

5.5.2 Pasar de relaciones ternarias a binarias

En el modelo y diagrama E-R buscamos tener relacionas binarias. Si tenemos un conjunto de relaciones R que asocia E_1, E_2 y E_3, podemos seguir los siguientes pasos para pasarla a binaria. Este proceso se puede generalizar.

Paso 1) Eliminar R

Paso 2) Añadir una nueva entidad E4 con los atributos R (si los hay)

Paso 3) Añadir tres relaciones binarias nuevas

R1 (E4, E1); R2 (E4, E2); R3 (R4, E3)

Paso 4) Si R tiene atributos asociados, éstos pasan al conjunto de entidades E4.

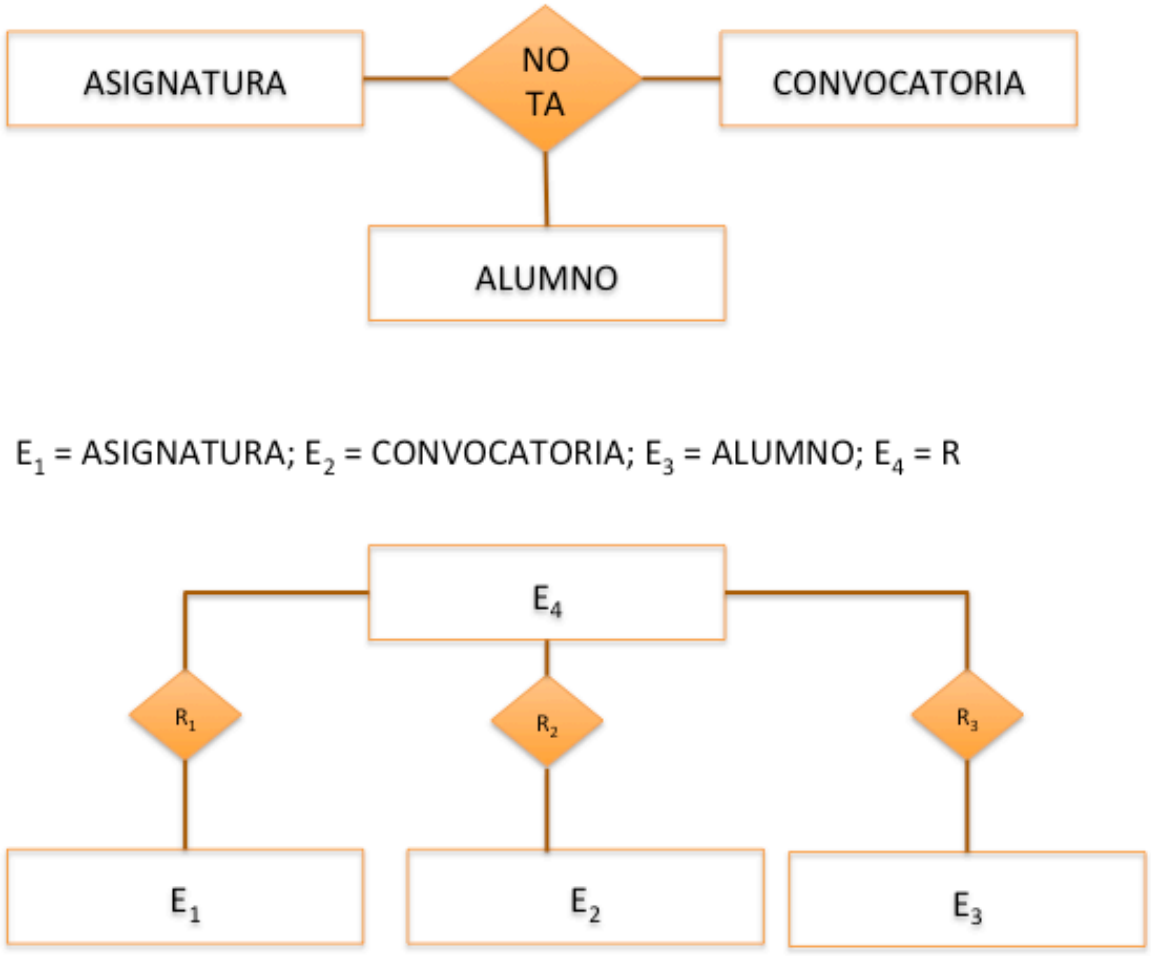


Figura 9: Paso de relación ternaria a binaria

5.5.3 Ubicación de los atributos de las relaciones

La restricción de cardinalidad puede afectar a la ubicación de los atributos de un conjunto de relaciones.

Si **la restricción de cardinalidad es de 1 a 1**, **los atributos se ubican en las entidades** y no en la relación, porque los atributos no dependen de la relación.

Si **la restricción de cardinalidad es de 1 a N, o de N a 1, los atributos se colocan en la relación o en el conjunto de entidades del lado de la N**. Es una decisión del diseñador.

Si **la restricción de cardinalidad es de N a N, los atributos se colocan en la relación**, porque el valor del atributo se determina mediante la combinación de los conjuntos de las entidades participantes, en lugar de por cada entidad por separado.

Por ejemplo, imagínate que tienes un conjunto de entidades CLIENTE y CUENTA, que participan en una relación IMPOSITOR con cardinalidad de 1 a N, de CLIENTE a CUENTA. Un cliente puede tener varias cuentas, pero una cuenta sólo pertenece a un cliente. Si queremos añadir la fecha de último acceso a la cuenta, lo podemos hacer en la entidad CUENTA, ya que cada entidad cuenta participa en la relación con a lo sumo un ejemplar de cliente (y es el mismo). El atributo fecha de acceso también lo podemos poner en IMPOSITOR.

En cambio, si la cardinalidad fuera N a N, la fecha de acceso no podría estar en CUENTA, ya que las cuentas son compartidas – la fecha de acceso correspondería…al último cliente, perdiendo las fechas de acceso de otros clientes. En este caso, el atributo se determina en la relación.

5.6 Paso a tablas

Las bases de datos relacionales almacenan (entre otros elementos) relaciones, es decir, tablas bidimensionales. Por tanto, nos interesa saber cómo pasar un diagrama E-R (que lo verás en prácticas) a tablas.

5.6.1 Representación tabular entidades fuertes

Sea E un conjunto de entidades fuertes con atributos descriptivos a_1, $a_2 \ldots a_n$. Este conjunto de entidades se representa mediante una tabla E con n columnas, una para cada atributo.

Entidad fuerte: ASIGNATURA (código, nombre, semestre)

Código	Nombre	Semestre
BBDD	Bases de Datos	2
PII	Programación II	2

5.6.2 Representación tabular entidades débiles

Sea A un conjunto de entidades débiles con los atributos a_1, $a_2 \ldots a_n$. Sea B el conjunto de entidades fuertes del que A depende, con la clave primaria formada por los atributos b_1, $b_2 \ldots b_n$. Se representa el conjunto de entidades A con una tabla con una columna para cada atributo del conjunto

$$\{ a_1, a_2 \ldots a_n \} \cup \{ b_1, b_2 \ldots b_n \}$$

5.6.3 Representación tabular conjunto de relaciones

Sea R un conjunto de relaciones. Sea a_1, $a_2 \ldots a_n$ el conjunto de atributos formados por la unión de las claves primarias de cada uno de los conjuntos de entidades que participan en R. Sea b_1, $b_2 \ldots b_n$ el conjunto de atributos descriptivos de R (si los hay). El conjunto de relaciones R se representa con una tabla con una columna para cada uno de los atributos del conjunto

$$\{ a_1, a_2 \ldots a_n \} \cup \{ b_1, b_2 \ldots b_n \}$$

El conjunto de relaciones que une una entidad fuerte con una débil no tiene tabla. ¿Porqué? Porque no aporta información.

5.6.4 Combinación de tablas

Según lo visto, para un conjunto de relaciones R entre un par de conjuntos de entidades (A, B) creamos tres tablas: T_A, T_B, y T_{AB}. Algunas veces las podemos combinar, y así reducir el número de tablas.

La combinación de tablas requiere tener en cuenta la restricción de cardinalidad:

En el caso de **1 a 1**, **la tabla de la relación (**T_{AB}**) se puede combinar con cualquiera de las tablas del conjunto de entidades**. ¿Qué tiene la tabla de la relación? Las claves primarias de A y B (y atributos descriptivos, si los hay). Por tanto, combinar es pasar la clave primaria de A a B (o viceversa).

En el caso **de N a N, la tabla de la relación (**T_{AB}**) no se puede combinar**, porque necesitamos tener la tabla de la relación para no perder datos.

En el caso de **1 a N y de N a 1**, **la combinación se reduce a mover la clave primaria del lado de 1 (contenida en** T_{AB}**) a la tabla del lado de N**. Por ejemplo, imagínate que tenemos CUENTAS y SUCURSALES bancarias. La relación es de N a 1 de CUENTAS a SUCURSALES – una sucursal tiene varias cuentas, pero una cuenta sólo puede pertenecer a una sucursal. Tenemos las tablas:

$$T_{CUENTA} = (num\_cuenta, saldo)$$

$$T_{SUCURSAL} = (nombre\_sucursal, ciudad, activo)$$

$$T_{CU\text{-}SU} = (num\_cuenta, nombre\_sucursal)$$

Con la combinación, quedaría:

$$T_{CUENTA} = (num\_cuenta, saldo, nombre\_sucursal)$$

$$T_{SUCURSAL} = (nombre\_sucursal, ciudad, activo)$$

No necesitamos la tabla $T_{CU\text{-}SU}$

5.6.5 Representación tabular atributos compuestos

No se crea una tabla para ellos. Tampoco se crea una columna específica para el propio atributo compuesto. Lo que sí hacemos es crear columnas para cada uno de los atributos componentes. Por ejemplo, si tenemos el atributo compuesto dirección con componentes ciudad y calle, la tabla generada contendría las columnas calle.dirección, ciudad.dirección.

5.6.6. Representación tabular atributos multivalorados

Para un atributo multivalorado M se crea una tabla T con una columna C que corresponde a la clave primaria del conjunto de entidades (o relaciones) del que M es atributo. Cada fila de la tabla se corresponde con un valor (de los posibles) que puede adoptar el atributo multivalorado.

5.7 Referencias

- Capítulo 2 de Silberschatz, Korht & Sudarshan.

6. TEMA 5 – Normalización

Los objetivos del tema son:

- Conocer la importancia de la normalización en el diseño E-R.
- Saber identificar la redundancia de datos y anomalías de actualización en diseños E-R.
- Conocer y saber identificar las dependencias funcionales.
- Conocer y saber distinguir entre dependencias funcionales completas y parciales.
- Conocer las tres formas normales principales: 1FN, 2FN y 3FN.
- Pasar una relación de 1FN a 3FN.

6.1 Introducción

En este tema continuamos con el diseño E-R. La normalización es una técnica de diseño de bases de datos que comienza examinando las relaciones (concretamente, las dependencias funcionales) entre los atributos. La normalización utiliza una serie de formas normales que nos ayudan a identificar el **agrupamiento óptimo de los atributos**. Por agrupamiento óptimo nos solemos referir a los siguientes aspectos:

1) mínimo número de atributos necesarios para satisfacer los requerimientos,
2) los atributos que tienen una relación lógica muy fuerte (dependencia funcional) se encuentran en la misma relación,
3) mínima redundancia de atributos, con excepción de las claves foráneas.

Y todo esto, ¿porqué? Gracias a la normalización, obtenemos un conjunto de relaciones con el que es fácil trabajar y que ocupa un mínimo espacio – gracias, por ejemplo, a eliminar redundancias.

La normalización se puede aplicar, a grandes rasgos, de dos maneras: *bottom-up* para construir relaciones bien diseñadas, y *top-down*, para validar el modelo E-R. Los Apuntes están escritos para que puedas aplicar la normalización siguiendo las dos aproximaciones.

6.2 Redundancia de datos y anomalías de actualización

En las bases de datos tenemos redundancia de datos. Un ejemplo son las copias de las claves primarias que se convierten en claves foráneas. Esta redundancia es necesaria. Lo que no queremos es tener redundancia innecesaria. Por ejemplo, considera el siguiente esquema de relaciones:

```
STAFF (staffNo, name, position, salary, branchNo)
BRANCH (branchNo, address)
STAFFBRANCH (staffNo, name, position, salary, branchNo, address)
```

La relación STAFFBRANCH tiene datos redundantes. Por ejemplo, si dos empleados trabajan en la misma oficina, la relación tiene los datos de la oficina duplicados. Las relaciones que tienen datos redundantes innecesarios presentan anomalías de actualización (inserción y eliminación).

6.2.1 Anomalías de inserción

Si consideramos la relación STAFFBRANCH, una anomalía de inserción sucede si queremos añadir un nuevo empleado. Para añadir el trabajador correctamente, tenemos que introducir correctamente los datos de la oficina. Si tenemos estos datos por separado – en otra relación, únicamente tendríamos que escribir el identificador de la oficina.

Otra anomalía de inserción es la que sucede cuando queremos introducir una oficina nueva en la que todavía no trabaja nadie. Esta situación nos forzaría a escribir valores NULL. Sin embargo, la clave primaria *staffNo* no puede ser NULL – violaría la restricción de integridad.

Como puedes ver, el diseño de las relaciones es importante.

6.2.2 Anomalías de eliminación

Si eliminamos el último trabajador de una oficina concreta, los datos de la oficina los perdemos en el diseño de STAFFBRANCH. Si tenemos el diseño en dos relaciones, STAFF y BRANCH, esta anomalía no sucede.

6.3 Dependencias funcionales

Es un concepto importante. Yo diría que fundamental en la normalización de bases de datos (relacionales). En los Apuntes, asumimos que cada atributo en la base de datos tiene un nombre único[13].

Una **dependencia funcional** describe la relación entre los atributos de una relación. Si A y B son atributos de una relación R, B es funcionalmente dependiente de A (A ➔ B) si cada valor de A está asociado con exactamente un valor de B. A y B pueden ser atributos o conjunto de atributos.

La dependencia funcional es una **propiedad del significado de los atributos** de una relación. Si tenemos dos atributos, A y B, y B depende funcionalmente de A, significa que si examinamos la relación, cuando dos filas tienen el mismo valor de A, entonces tienen el mismo valor de B. Lo contrario, sin embargo, no siempre se cumple.

Si B depende funcionalmente de A, también podemos decir que A **determina** funcionalmente B. El determinante se refiere a un atributo, o conjunto de atributos, ubicados en el lado izquierdo de la dependencia funcional.

Un ejemplo. Considera las siguientes relaciones:

```
STAFF (staffNo, name, position, salary, branchNo)
BRANCH (branchNo, address)
```

[13] Si no tiene un nombre único, lo podemos conseguir anteponiendo el nombre de la relación al nombre del atributo

Para cada *staffNo*, podemos determinar su cargo. Por supuesto, asumimos que un empleado tiene un cargo. En otras palabras, *position* depende funcionalmente de *staffNo*. Sin embargo, *position* no determina funcionalmente *staffNo*. Un trabajador tiene un cargo, pero varios trabajadores pueden tener el mismo cargo.

staffNo ➔ position

La relación entre *staffNo* y *position* es de 1 a 1. Pero la relación entre *position* y *staffNo* es de 1 a N. En normalización, nos interesa identificar relaciones de 1 a 1 entre los atributos del determinante y los atributos que depende del determinante (los del lado de la derecha).

En normalización, nos interesa identificar dependencias funcionales que son independientes de la instancia o ejemplar de la base de datos. **Una dependencia funcional es una propiedad del diseño de la relación y no del valor de los atributos en un momento particular**. Esto es muy importante. Te recomiendo que lo tengas presente.

Otro ejemplo. Considera la siguiente relación:

```
STAFF (staffNo, name, position, salary, branchNo)
```

Dado un *staffNo*, parece que podemos identificar su nombre. Y dado un nombre, parece que podemos identificar su *staffNo*. ¿Podemos afirmar pues que *staffNo* (*name*) determina funcionalmente el atributo *name* (*staffNo*)? En un momento o instancia podemos, pero no siempre.

Para tener en cuenta este aspecto temporal, necesitamos entrar en el **propósito de cada atributo** de la relación.

El objetivo del atributo *staffNo* es identificar unívocamente a un trabajador. El objetivo del atributo *name* es guardar el nombre de un trabajador.

Por tanto, si sabemos el *staffNo* de un trabajador, siempre podemos saber su nombre. Pero podemos tener diferentes trabajadores con el mismo nombre…

La relación entre *staffNo* y *name* es de 1 a 1. La relación entre *name* y *staffNo* es de 1 a N. ¿Cuántos empleados con el nombre Jordi pueden haber en una oficina – especialmente en Catalunya?

6.3.1 Dependencia funcional completa y parcial

Una dependencia funcional **completa** indica que si A y B son atributos de una relación, B depende funcionalmente de A si y sólo si B depende funcionalmente de A pero no de cualquier subconjunto de A. En otras palabras, el determinante es mínimo.

Esta definición la puedes entender así: **una dependencia funcional es completa si al eliminar un atributo del determinante hace que la dependencia deje de existir**. Una dependencia funcional es **parcial** cuando algún atributo del determinante se puede eliminar y la dependencia se mantiene.

Un ejemplo. La dependencia funcional siguiente es parcial:

(staffNo, name) ➔ branchNo

¿Porqué es parcial? Porque si eliminamos *name*, la dependencia funcional se mantiene. El determinante no es mínimo; no tiene el mínimo número de atributos. De hecho, la dependencia funcional completa es:

staffNo ➔ branchNo

(si la relación entre oficina y trabajador fuera de 1 a 1, cosa que parece que no tiene mucho sentido, branchNo ➔ staffNo, también seria una dependencia funcional completa).

Las dependencias funcionales que utilizamos en la normalización tienen tres características:

1) Relación 1 a 1

2) Se mantienen a lo largo del tiempo

3) El determinante es mínimo

6.4 Identificar dependencias funcionales

Si se conoce el significado de los atributos, se pueden identificar las dependencias funcionales semánticamente. Si se desconoce el significado de los atributos, tenemos que examinar los datos de una instancia siempre que podamos asumir que son representativos de todos los datos de la base de datos.

Por ejemplo, si consideramos la siguiente relación

```
STAFFBRANCH (staffNo, name, position, salary, branchNo, address)
```

Podemos identificar las siguientes dependencias funcionales

staffNo ➔ (name, position, salary, branchNo, address)

El identificador de un trabajador determina el valor del resto de los atributos.

branchNo ➔ address

Si conocemos el código de la oficina, tenemos su dirección.

address ➔ branchNo

Habitualmente no tenemos dos oficinas en la misma dirección, así que el valor del campo dirección determina el código de la oficina.

(branchNo, position) ➔ salary

Podemos pensar que el salario de un trabajador depende de su cargo y también de la oficina. Este dependencia asume que trabajadores con el mismo cargo tienen diferentes sueldos dependiendo de la oficina en la que trabajen (si es la oficina central…).

Otro ejemplo. Asumimos que la instancia es representativa de todos los posibles valores que los atributos A, B, C, D y E pueden tomar en la base de datos. Identificamos pues las dependencias funcionales examinando la tabla:

A	B	C	D	E
A	B	Z	W	Q
E	B	R	W	P
A	D	Z	W	T
E	D	R	W	Q
A	F	Z	S	T
E	F	R	S	T

Para identificar las dependencias funcionales, por ejemplo, recorremos la tabla de izquierda a derecha. Primero, nos fijamos en la columna A, e identificamos pares con los valores de las otras columnas, de tal manera que para un valor de A, siempre encontremos el mismo en las otras columnas. Y repetimos el proceso para cada columna.

DF1: A ➔ C

DF2: C ➔ A

DF3: B ➔ D

DF1: Cuando el valor ‘A’ aparece en la columna A, tenemos el valor ‘Z’ en la columna C. Cuando el valor ‘E’ aparece en la columna A, tenemos el valor R en la columna C. Por tanto, tenemos una relación 1 a 1, y los valores de C son consistentes con los valores de A. Siempre que tenemos una ‘A’ en A, tenemos una ‘Z’ en C. Siempre que tenemos una ‘E’ en A, tenemos una ‘R’ en C. Hemos encontrado una dependencia funcional.

DF2: Mismo razonamiento que DF1.

DF3: Podemos observar que cuando el valor de la columna B es ‘B’ o ‘D’, el valor en D es ‘W’, y cuando en B tenemos una ‘F’, en D tenemos una ‘S’. Hay una relación de 1 a 1 de B a D. B determina funcionalmente D. Lo contrario no es cierto, porque cuando D tiene el valor ‘W’, el valor

en B es B o D. La relación es de 1 a N. Hemos encontrado una dependencia funcional entre B y D. B determina D.

El último atributo que nos queda por considerar es E. En primer lugar, no encontramos ningún par entre los valores de A, B, C y D con los valores de E. El valor 'A' de la columna A se corresponde con los valores 'Q' y 'T' en E. El valor 'B' de la columna B se corresponde con los valores 'Q' y 'P' en E. El valor 'Z' de la columna C se corresponde con los valores 'Q' y 'T' en C. El valor 'W' de la columna D se corresponde con los valores 'P', 'Q' y 'T' en C. La columna E tampoco determina el valor de ninguna de las otras columnas.

Hasta ahora hemos ido columna a columna. El siguiente paso es considerar combinaciones de pares de columnas. (C, D), (B, D), (A, C), (A, D) no determinan E. Sin embargo, (A,B), y (B, C) determinan E. Hemos encontrado dos dependencias funcionales más.

DF4: (A, B) ➔ E

DF5: (B, C) ➔ E

Seguramente quieras continuar y explorar (A, B, C), y (A,B,C, D). Aunque no es incorrecto, recuerda que en normalización, las dependencias funcionales que utilizamos son aquellas en las que los determinantes son mínimos – en otras palabras, dependencia funcional completa[14].

Con las dependencias funcionales podemos identificar las claves primarias. Todos los atributos que no son parte de la clave primaria deben depender funcionalmente de la clave. ¿Porqué? Porque la clave primaria identifica unívocamente, y como mucho uno, una fila de la tabla. Por tanto, para cada valor de la clave primaria, encontramos una única fila en la tabla.

6.5 El proceso de normalización

El proceso de normalización se basa en claves primarias (también las candidatas) y dependencias funcionales. Hay diferentes niveles de normalización. Los veremos más adelante. Si una relación incumple los requerimientos de normalización de un nivel, se descompone en relaciones que, individualmente, satisfacen los requerimientos.

Habitualmente se habla de 1FN (Primera Forma Normal), 2NF (Segunda Forma Normal), y 3FN (Tercera Forma Normal). Una variante de la 3FN es la BCNF (Boyce-Codd Normal Form). También existen la 4FN y la 5FN. En los Apuntes, y en la asignatura, nos centramos en la 1, 2, y 3FN, porque son las clásicas[15].

La 1FN es obligatoria para que un modelo relacional sea válido. El resto de formas normales son opcionales, pero recomendables.

[14] Por ejemplo, (A,B, C) ➔ E. Es una dependencia funcional parcial. Si quito C, la dependencia funcional se mantiene. (A,B) ➔ E, es una dependencia funcional completa. Si quito A, o B, la dependencia funcional no se mantiene.
[15] La 4Fn y 5FN tratan situaciones excepcionales que, aunque importantes, suceden en pocas ocasiones.

6.5.1 1FN

Una relación está en 1FN si la intersección de cada fila y columna contiene únicamente un valor.

Una relación que no está en 1FN se puede transformar a una en 1FN de varias maneras. Por ejemplo, identificamos el grupo de atributos que se repite – tomando como punto de partida un atributo clave (o más de uno). Después, completamos las columnas vacías con datos que no se repiten. Otra aproximación consiste en colocar el grupo que se repite, con una copia del atributo clave, en una relación nueva. Si hay varios grupos, se crean tantas relaciones como se necesiten hasta que todas estén en 1FN.

Un grupo que se repite es un atributo, o conjunto de atributos, que tienen más d eun valor para la única ocurrencia de otro valor (que llamaremos clave, y que identifica cada fila) de la tabla no normalizada.

Considera la siguiente tabla:

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	Rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
		PG16	5 Norvar Dr, Glasglow	1-Sept-04	1-Sep-04	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-June-03	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
		PG16	5 Norvar Dr, Glasglow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

Relación 13: CLIENTRENTAL

CLIENTRENTAL no está en 1FN porque la intersección de filas y columnas no es atómica.

El grupo que se repite está formado por los atributos (propertyNo, pAddress, rentStart, rentFinish, Rent, ownerNo, oName).

Para pasar la relación a 1FN, duplicamos los valores de los atributos que no están duplicados. La tabla resultante está en 1FN.

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	Rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
CR76	John Kay	PG16	5 Norvar Dr, Glasglow	1-Sept-04	1-Sep-04	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-June-03	350	CO40	Tina Murphy
CR56	Aline Stewart	PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
CR56	Aline Stewart	PG16	5 Norvar Dr, Glasglow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

Relación 14: CLIENTRENTAL en 1FN

6.5.2 2FN

Diremos que una relación está en 2FN si está en 1FN y cada atributo que no forma parte de la clave primaria tiene dependencia funcional completa de la clave primaria.

Observa que una tabla en 1FN cuya clave primaria sea atómica (un único atributo) automáticamente está en 2FN, ya que la 2FN aplica únicamente a las relaciones en las que la clave primaria es compuesta (por dos o más atributos).

Pasar de 1FN a 2FN consiste en eliminar las dependencias funcionales parciales con respecto a la clave primaria[16]. Para cada dependencia funcional parcial, eliminamos el atributo (o atributos) 'parcial' de la tabla original y lo ponemos en una nueva relación con una copia del determinante – uno de los atributos que forman la clave primaria. Como resultado, la tabla con la clave primaria se

[16] Únicamente eliminamos las dependencias parciales.

modifica. Concretamente, se hace pequeña, porque los atributos se eliminan y pasan a estar en otras (nuevas) relaciones. Veamos un ejemplo.

Considera la tabla CLIENTRENTAL (Relación 14). Identificamos las dependencias funcionales:

DF1: (clientNo, propertyNo) ➔ (rentStart, rentFinish). La clave primaria de la relación CLIENTRENTAL es el determinante de la dependencia funcional.

DF2: clientNo ➔ cName. Es una dependencia funcional, porque dado el identificador de un cliente, puedo obtener el nombre del cliente, pero es una dependencia funcional parcial con respecto a la clave primaria, porque contiene uno de los dos atributos que la forman.

DF3: propertyNo ➔ (address, rent, ownerNo, oName). Es una dependencia funcional, porque dado el identificador de propiedad, puedo identificar unívocamente, y a lo sumo, la dirección, alquiler, código y nombre del propietario. Sin embargo, es una dependencia funcional parcial con respecto a la clave primaria, porque el determinante tiene uno de los atributos que la forman.

Una vez identificadas las dependencias funcionales parciales, el siguiente paso es crear nuevas relaciones para cada una de ellas:

- CLIENT (clientNo, cName)
- PROPERTYOWNER (propertyNo, address, rent, ownerNo, oName)

Al crear estas nuevas relaciones, los atributos los eliminamos de la tabla base, y mantenemos una copia de los determinantes. Por tanto, la relación CLIENTRENTAL queda de la siguiente manera:

- CLIENTRENTAL (clientNo, propertyNo, rentStart, rentFinish)

Observa que las tres relaciones están en 2FN. CLIENT está en 2FN porque su clave primaria es atómica. Lo mismo para PROPERTYOWNER. CLIENTRENTAL está en 2FN porque los atributos que no forman parte de la clave primaria tienen una dependencia funcional completa de ésta.

6.5.3 3FN

La 3FN se centra en eliminar las dependencias transitivas.

Si A ➔ B, y B ➔ C, entonces diremos que C **depende transitivamente** de A vía B.

Una relación está en 3FN si está en 2FN y los atributos que no forman parte de la clave primaria no tienen dependencia transitiva de la clave primaria.

Pasar de 2FN a 3FN significa eliminar las dependencias transitivas. Para cada dependencia transitiva, eliminamos el atributo o atributos que dependen transitivamente de la relación y lo colocamos en una nueva relación con una copia del determinante. Veamos un ejemplo.

Partimos de las relaciones en 2FN siguientes:

- CLIENT (clientNo, cName)
- PROPERTYOWNER (propertyNo, address, rent, ownerNo, oName)
- CLIENTRENTAL (clientNo, propertyNo, rentStart, rentFinish)

Las relaciones CLIENT y CLIENTRENTAL no tienen dependencias funcionales transitivas. Como sabemos que están en 2FN, entonces, según la definición, están en 3FN.

Tenemos una dependencia funcional transitiva en PROPERTYOWNER

propertyNo ➔ (address, rent, ownerNo, oName)

ownerNo ➔ oName

Es una dependencia transitiva porque propertyNo ➔ ownerNo ➔ oName.

El atributo que depende transitivamente es *oName*. Para transformar la relación PROPERTYOWNER, necesitamos eliminar el atributo *oName* de la relación base, y ponerlo en una nueva relación con su determinante.

- OWNER (ownerNo, oName)
- PROPERTYOWNER (propertyNo, address, rent, ownerNo)

Como resultado, las relaciones quedan de la siguiente manera:

- CLIENT (clientNo, cName)
- OWNER (ownerNo, oName)
- PROPERTYOWNER (propertyNo, address, rent, ownerNo)
- CLIENTRENTAL (clientNo, propertyNo, rentStart, rentFinish)

6.5.4 BCNF

BCNF añade restricciones a la 3FN. Concretamente, una relación está en BCNF si y solo si los determinantes de las dependencias funcionales son claves candidatas.

Las relaciones CLIENT, OWNER y PROPERTYOWNER están en BCNF porque los determinantes de las dependencias funcionales son atómicos, y a su vez, son clave candidata.

La relación CLIENTRENTAL tiene varias dependencias funcionales:

(clientNo, propertyNo) ➔ rentStart, rentFinish

(clientNo, rentStart) ➔ propertyNo, rentFinish.

(propertyNo, rentStart) ➔ clientNo, rentFinish

Los tres determinantes (son las tres parejas de la izquierda) son claves candidatas. Asumimos que un cliente puede alquilar únicamente una propiedad en un momento concreto. Por tanto, la relación RENTAL está en BCNF[17].

6.6 Referencias

- Capítulo 13 de Connolly & Begg
- Capítulo 7 de Silberschatz, Korht & Sudarshan

[17] Si no lo estuviera, tendríamos que aplicar un proceso similar al indicado en el capítulo 13 de Begg y capítulo 7 de Silberschatz, Korht & Sudarshan.

7. TEMA 6 – Transacciones y concurrencia

Los objetivos del tema son:

- Conocer el concepto de transacción.
- Conocer las propiedad ACID de las transacciones.
- Conocer algunos problemas importantes relacionados con la necesidad del control de la concurrencia.
- Conocer el concepto de la serialización y las planificación en serie y no en serie.
- Conocer la técnica de control de concurrencia basada en los bloqueos.
- Distinguir entre bloqueos exclusivos y compartidos.
- Conocer el protocolo 2PL (de dos fases) de bloqueo / desbloqueo.

7.1 Introducción

Una transacción es una unidad lógica de trabajo. Es una acción, o conjunto de acciones, que la realiza el usuario (o el programa de aplicación) cuando lee o modifica los contenidos de una base de datos.

Para ilustrar el concepto de transacción, toma como ejemplo las relaciones STAFF y PROPERTY4RENT (Relación 1 y 4). Una transacción simple sería actualizar el salario de un miembro del STAFF. La transacción consiste en dos **operaciones de la base de datos (lectura y escritura)**, y una **operación que no es propia de la base de datos**, el incremento del salario.

Otro ejemplo de transacción es la eliminación de un miembro del STAFF dado su identificador. En este caso, además de eliminar el trabajador de la relación STAFF, tenemos que eliminar todas las filas de PROPERTY4RENT en las que dicho trabajador aparezca, y asignarle otro trabajador – porque la propiedad no se quedará sin nadie a su cargo. Si todas estas operaciones no se realizan, la base de datos quedará en un estado inconsistente.

Una transacción siempre debe transformar una base de datos en un **estado consistente**. Mientras se realiza la transacción, la base de datos puede ser inconsistente, pero nunca al finalizar la transacción.

Si una transacción finaliza satisfactoriamente, diremos que ha hecho ***commit***. Si, por el contrario, no se ha realizado satisfactoriamente, la transacción es abortada, y para que la base de datos recupere su estado inicial, se realiza un ***rollback***.

7.2 Propiedades de las transacciones (ACID)

- **Atomicidad**. Una transacción es una unidad indivisible que se ejecuta totalmente o no se ejecuta.
- **Consistencia**: Una transacción debe transformar una base de datos de un estado consistente a otro estado consistente. Es responsabilidad del SGBD y del programador.

- **Aislamiento**. Las transacciones se ejecutan de manera independiente de otras. En otras palabras, aunque las acciones de varias transacciones estén entrelazadas, el resultado final debería ser igual a si cada transacción se hubiera realizado en serie, una después de la otra, en algún orden.
- **Durabilidad**. Los efectos de las transacciones que se han realizado correctamente (*commit*) se guardan en la base de datos.

7.3 Control de concurrencia

El control de concurrencia es el proceso de manejar operaciones en la base de datos de manera simultánea de tal manera que no se interfieran unas a las otras. Es un proceso transparente para el usuario - para ti, como estudiante de la asignatura, vale la pena conocerlo en cierto nivel de detalle.

Uno de los objetivos de las bases de datos es permitir que varios (muchos) usuarios accedan a los datos, que son compartidos, concurrentemente. Si todos leen los datos, no tenemos muchos problemas. Pero si algunos los leen, y otros los modifican, a la vez, entonces se pueden originar inconsistencias.

Las transacciones interactúan entre ellas a través de la base de datos vía operaciones de escritura y lectura; no se envían mensajes entre ellas. Creo que esto es importante.

Tres problemas típicos derivados del control de concurrencia son:

- el problema de la actualización perdida,
- el problema de la lectura corrupta,
- y el problema del análisis inconsistente.

En estos ejemplos, que los verás a continuación, hay que pensar que las transacciones tienen el valor de bal_x cuando lo leen, y las operaciones de suma y resta no modifican el valor de la variable en la base de datos hasta que no se realiza la operación de escritura.

7.3.1 El problema de la actualización perdida

Una operación de actualización puede ser sobrescrita por otro usuario. Este es el problema de la actualización perdida (ver Figura 11):

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2	BEGIN	read (bal_x)	100
t_3	read (bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write (bal_x)	200
t_5	write (bal_x)	COMMIT	90
t_6	COMMIT		90

Figura 10: Problema de la actualización perdida

Si las dos transacciones (T1y T2) se ejecutan en serie, una después de la otra, el resultado final es 190. Pero no ocurre así en la Figura 11. Las dos transacciones leen el valor de bal_x, que es 100. El problema es que la escritura que realiza T_1 sobrescribe la actualización que ha hecho T_2. T1 resta 10 a su copia de bal_x **El valor de bal_x para T1 es de 100 (y no 200) – que es el valor cuando lo leyó**. Este problema se puede solucionar impidiendo que T1 lea el valor de bal_x hasta que T2 no lo haya actualizado.

7.3.2 El problema de la lectura corrupta

Este problema ocurre cuando una transacción puede ver los resultados intermedios de otra transacción antes que de ésta haya hecho *commit* (ver Figura 12). Por ver no me refiero a que se envíen mensajes entre ellos – no se puede hacer. Por ver me refiero a la situación de la Figura 12.

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2		read (bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	BEGIN	write (bal_x)	200
t_5	read (bal_x)	...	200
t_6	$bal_x = bal_x - 10$	ROLLBACK	100
t_7	write (bal_x)		190
t_8	COMMIT		190

Figura 11: Problema de la lectura corrupta

En este ejemplo, T2 incrementa en 100 el valor de bal_x. Sin embargo, T2 decide abortar la operación una vez que ha realizado la operación de escritura. Para cuando aborta, T1 ya he empezado y ha leído de la base de datos el valor de bal_x que es 200, resultado de la actualización de T2. Entonces realiza la operación de resta, siendo el resultado final de bal_x 190 y no 90, si tenemos en cuenta que T2 ha deshecho los cambios.

Realizar el *rollback* en T2 no es un error. **El problema es que T1 no debería acceder al valor de bal_x hasta que T2 no haya abortado y hecho *commit***.

7.3.3 El problema del análisis inconsistente

Este problema ocurre cuando una transacción lee varios valores de la base de datos pero una segunda transacción actualiza algunos de ellos mientras la primera se ejecuta. Por ejemplo, si una transacción está haciendo un balance de cuentas bancarias (suma la cantidad de todas ellas) obtendrá un valor incorrecto si, mientras se está ejecutando, otras transacciones están actualizando las cuentas bancarias (ver Figura 13).

Tiempo	T_1	T_2	bal_x	bal_y	bal_z	sum
t_1		BEGIN				
t_2	BEGIN	sum=0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	bal_x = balx-10	sum = sum + bal_x	100	50	25	100
t_5	write (bal_x)	read(bal_y)	100	50	25	100
t_6	read (bal_z)	sum = sum + bal_y	90	50	25	150
t_7	bal_z = bal_z + 10		90	50	25	150
t_8	write (bal_z)		90	50	35	150
t_9	COMMIT	read (bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		COMMIT	90	50	35	185

Figura 12: Problema del análisis inconsistente

T2 está calculando el balance de las cuentas bal_x, bal_y, y bal_z, Sin embargo, al mismo tiempo, la transacción T1 a transferido 10 euros de la cuenta x a la cuenta z. Cuando T2 lee el valor de la cuenta z, se lo encuentra actualizado (10 euros más), pero el valor de la cuenta x, para ella, sigue siendo el original – sin los 10 euros de menos. Como resultado, T2 tiene un resultado incorrecto (10 euros de más; en vez de 175 = 100 + 50 + 25, el resultado es 185 = 100 + **50** + **35**).

La solución pasa por no permitir que T2 lea las cuentas x y z hasta que T1 no haya acabado sus actualizaciones.

7.4 Serialización

El objetivo del control de concurrencia es organizar transacciones de tal manera que no haya ninguna interferencia entre ellas. La solución más obvia es ejecutar una transacción después de otra. Pero esto no nos ayuda – dónde está la concurrencia, entonces?

La serialización nos ayuda a identificar aquellas ejecuciones de transacciones que garanticen la consistencia de los datos.

Una **planificación** es una secuencia de operaciones de un conjunto de transacciones que preserva el orden de las operaciones de cada transacción individual.

Una **planificación en serie** es una planificación en la que las operaciones de cada transacción se ejecutan consecutivamente sin operaciones de otras transacciones.

Una **planificación no en serie** es una planificación en las que las operaciones de un conjunto de transacciones están intercaladas.

El objetivo de la serialización es **encontrar una planificación no en serie** que permita a las transacciones ejecutarse concurrentemente sin interferirse unas a las otras. Si encontramos una

planificación no en serie – en algún orden – que lo permite, diremos que la planificación es **serializable**.

En la serialización, el orden de las operaciones de escritura y lectura son importantes:

- Si dos transacciones únicamente leen datos, el orden no es importante, porque no hay conflictos.
- Si dos transacciones leen y escriben datos independientes, el orden tampoco es importante, porque los datos no se comparten.
- Si una transacción lee un dato y la otra también lo lee o escribe sobre él, entonces el orden si es importante.

7.5 Técnicas de control de concurrencia

Dos técnicas principales: bloqueo y marca de tiempo. Las dos aproximaciones son pesimistas, porque retrasan las transacciones en caso de conflicto. Las técnicas optimistas no se discuten en los Apuntes, porque considero que se discutirían mejor en un curso más avanzado. En los Apuntes, nos centramos en el bloqueo.

7.5.1 Bloqueo

Es un procedimiento para controlar el acceso concurrente a los datos. Cuando una transacción accede a la base de datos, un bloque puede denegar el acceso a otras transacciones para impedir resultados incorrectos.

La característica fundamental es que una transacción debe reclamar un bloqueo compartido (leer) o exclusivo (escribir) sobre un elemento ANTES de realizar la operación de lectura o escritura de la base de datos.

Las operaciones de bloqueo se pueden aplicar a diferentes niveles de granularidad, desde una tabla, a una fila, o a toda la base de datos.

7.5.2 Bloque compartido y exclusivo

Si una transacción tiene un **bloqueo compartido**, puede leer pero no modificar. Varias transacciones pueden compartir bloqueos compartidos.

Si una transacción tiene un **bloqueo exclusivo**, puede leer y modificar. Ninguna otra transacción puede acceder al elemento bloqueado.

Más de una transacción puede tener bloqueos compartidos sobre el mismo elemento a la vez. Un bloqueo exclusivo significa que ninguna otra transacción puede acceder al elemento bloqueado.

Los bloqueos funcionan de la siguiente manera:

1)Si una transacción necesita acceder a un elemento, primero debe bloquearlo.

2)Si un elemento no está bloqueado, entonces la transacción lo puede bloquear.

3)Si el elemento está bloqueado, el SGBD determina si la solicitud de bloqueo es compatible con el bloqueo actual.

a. Si el bloqueo compartido se solicita sobre un elemento que tiene un bloqueo compartido, la solicitud se acepta.
b. De lo contrario – sería un bloqueo exclusivo, y, por definición, otras transacciones diferentes a las que tiene el bloqueo no pueden acceder al elemento - la transacción debe esperar.

4)Una transacción bloquea un elemento hasta que lo desbloquea durante su ejecución o termina. Es entonces cuando el bloqueo exclusivo acaba y el resultado de la operación se hace visible al resto de transacciones.

Como te puedes imaginar, tenemos dos grandes tipos de operaciones de bloqueo: *lock* y *unlock*.

7.5.3 2PL

Un protocolo de bloqueo es un conjunto de reglas que cada transacción debe seguir para asegurar que, incluso cuando las acciones de las transacciones estén entrelazadas, el resultado final es idéntico al resultado de ejecutar las transacciones en algún orden en serie.

2PL. Una transacción sigue el protocolo de bloqueo 2PL (en dos fases) si todas las operaciones de bloqueo preceden a la primera operación de desbloqueo.

Por lo tanto, **una transacción que sigue el protocolo 2PL tiene dos fases**: una en la que se hace grande, en la que solicita y adquiere todos los bloqueos que necesita. Esta fase va seguida de otra en la que desbloquea y no puede solicitar nuevos bloqueos hasta que hayan acabado todos los desbloqueos. Normalmente, las fases de alternan: bloqueo – desbloqueo – bloqueo…

7.5.4. 2PL para prevenir el problema de la actualización perdida

La Figura 14 muestra la aplicación del protocolo 2PL para prevenir el problema de la actualización perdida. La T2 establece un bloqueo exclusivo sobre bal_x, ya que lo va a modificar. Realiza las operaciones y actualiza el valor. Cuando T1 comienza, solicita un bloqueo exclusivo sobre bal_x, porque lo quiere actualizar. Sin embargo, tiene que esperar, porque el elemento tiene un bloqueo exclusivo. Es cuando T2 acaba que lo desbloquea, y entonces T1 puede bloquear exclusivamente el elemento y actualizarlo. Como resultado de este protocolo, el valor de bal_x es el esperado y equivalente a realizar T1 y T2 en serie – una después de otra.

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2	BEGIN	**write_lock**(bal_x)	100
t_3	write_lock(bal_x)	read(bal_x)	100
t_4	**WAIT**	$bal_x = bal_x + 100$	100
t_5	**WAIT**	write (bal_x)	200
t_6	**WAIT**	COMMIT / UNLOCK (bal_x)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x$-10		200
t_9	write (bal_x)		190
t_{10}	COMMIT / UNLOCK (bal_x)		190

Figura 13: 2PL para prevenir el problema de actualización perdida

7.5.5 2PL para prevenir el problema de la lectura corrupta

La Figura 15 muestra la aplicación del protocolo 2PL para prevenir el problema de la lectura corrupta.

Tiempo	T_1	T_2	bal_x
t_1		BEGIN	100
t_2		**write_lock**(bal_x)	100
t_3		read(bal_x)	100
t_4	BEGIN	$bal_x = bal_x + 100$	100
t_5	write_lock(bal_x)	write (bal_x)	200
t_6	**WAIT**	**ROLLBACK / UNLOCK** (bal_x)	100
t_7	read(bal_x)		100
t_8	$bal_x = bal_x$ - 10		100
t_9	write (bal_x)		90
t_{10}	COMMIT / UNLOCK (bal_x)		90

Figura 14: 2PL para prevenir el problema de la lectura corrupta

La T2 realiza un bloqueo exclusivo sobre bal_x. Entonces procede a leer su valor y actualizarlo. Cuando se ejecuta el *rollback*, se produce un desbloqueo y los resultados son visibles. Cuando la T1 ha empezado, también ha pedido un bloqueo exclusivo, pero se ha tenido que esperar hasta que la T2 finalize o aborte. Cuando esto pasa, la T1 puede leer el valor de bal_x y actualizarlo. Como resultado de seguir el protocolo 2PL, el valor de bal_x es el esperado.

7.5.6 2PL para prevenir el problema del análisis inconsistente

La Figura 16 muestra la aplicación del protocolo 2PL para prevenir el problema del análisis inconsistente.

Tiempo	T1	T2	bal_x	bal_y	bal_z	sum
t1		BEGIN	100	50	25	
t2	BEGIN	Sum = 0	100	50	25	0
t3	**write_lock**(bal_x)		100	50	25	0
t4	read(bal_x)	**read_lock**(bal_x)	100	50	25	0
t5	$bal_x = bal_x - 10$	**WAIT**	100	50	25	0
t6	write (bal_x)	**WAIT**	90	50	25	0
t7	write_lock(bal_z)	**WAIT**	90	50	25	0
t8	write(bal_z)	**WAIT**	90	50	25	0
t9	$bal_z = bal_z + 10$	**WAIT**	90	50	25	0
t10	write (bal_z)	**WAIT**	90	50	35	0
t11	**COMMIT / UNLOCK**	**WAIT**	90	50	35	0
t12		read(bal_x)	90	50	35	0
t13		sum += bal_x	90	50	35	90
t14		read_lock(bal_y)	90	50	35	90
t15		read(bal_y)	90	50	35	90
t16		sum += bal_y	90	50	35	140
t17		read_lock(bal_z)	90	50	35	140
t18		read(bal_z)	90	50	35	140
t19		sum += bal_z	90	50	35	175
t20		COMMIT / UNLOCK	90	50	35	175

Figura 15: 2PL para prevenir el problema del análisis inconsistente

T1 comienza solicitando un bloqueo exclusivo sobre bal_x. T2, cuando solicita un bloqueo compartido, tiene que esperar – recuerda que un bloqueo exclusivo es justo eso, exclusivo. Mientras, T1 realiza cada operación de lectura con un bloqueo exclusivo, ya que quiere modificar. T2 tiene que esperar hasta que T5 acaba, cuando se produce todos los desbloqueos. Este punto quizá es importante remarcarlo. Según el protocolo 2PL, todos los bloqueos van antes que el primer desbloqueo, recuerda.

7.5.7 Punto muerto (*deadlock*)

El protocolo 2PL tiene problema. Por ejemplo, el punto muerto. El punto muerto sucede cuando dos (o más) transacciones esperan desbloqueos de elementos bloqueados por otras transacciones. La Figura 17 muestra un ejemplo.

Tiempo	T1	T2
t_1	BEGIN	
t_2	**write_lock(bal_x)**	BEGIN
t_3	read(bal_x)	**write_lock(bal_y)**
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write (bal_x)	$bal_y = bal_y + 100$
t_6	**write_lock(bal_y)**	write (bal_y)
t_7	WAIT	**write_lock(bal_x)**
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	...	WAIT
T_{11}	...	...

Figura 16: Ejemplo de punto muerto

La manera de solucionar el punto muerto es abortar una o más transacciones. Esto involucra deshacer todos los cambios realizados por la transacción abortada, y, entonces, las otras pueden continuar. La transacción abortada puede volver a comenzar.

Para evitar el punto muerto podemos tener un contador. Una transacción solicita un bloqueo y espera un tiempo determinado. Si pasado este tiempo, no ha podido realizar el bloqueo, el SGBD asume que está en punto muerto, y aborta la transacción y la reinicia.

7.6 Referencias

- Capítulo 20 de Connolly & Begg
- Capítulo 16 de Ramakrishnan & Gehrke

8. Tema 7 – Almacenamiento y árboles B+

Los objetivos de este tema son:

- Conocer algunos aspectos de almacenamiento externo de las bases de datos.
- Conocer las principales estructuras de datos de indexación.
- Conocer una de las principales estructuras de datos en forma arborescente en bases de datos: árboles B+.
- Saber aplicar los algoritmos de búsqueda, inserción y eliminación en árboles B+.

8.1 Algunos aspectos de almacenamiento

En este tema me centraré en almacenamiento **externo**. Con esto quiero decir almacenamiento fuera de la memoria principal del ordenador. ¿Porqué? Los SGBD suelen trabajar con un gran volumen de datos, y éstos se suelen guardar en discos externos, por su capacidad, persistencia (y también, precio). Los datos se guardan en discos externos, se pasan a la memoria para ser procesados, y se vuelven a guardar en los discos.

El acceso a disco externo es más lento que el acceso a memoria. Es habitual hablar de nanosegundos para acceder a la memoria principal del ordenador, y en milisegundos para hacerlo a un disco externo – sabes cuál es la equivalencia entre nanosegundos y milisegundos? Si la buscas en algún libro o en Google, verás que estamos hablando de una cantidad nada despreciable de tiempo. El tiempo de acceso a disco depende, entre otros factores, del tiempo de buscar el bloque, el tiempo de rotación del disco (recuerda que es un disco físico), y el tiempo de transferencia.

Las bases de datos en almacenamiento externo o secundario se organizan en ficheros. De hecho, la abstracción básica de los datos en un SGBD es un conjunto de registros, que son un fichero. Cada registro consiste de uno o más campos (por ejemplo, columnas de una tabla), y un fichero consiste, a nivel físico, de una o más páginas. Una página es la unidad mínima de información que se lee o se escribe en disco. Típicos valores de páginas son 4 / 8 Kb.

8.2 Estructuras de datos indexadas

Un índice es una estructura de datos que organiza los registros de datos en disco para optimizar operaciones de búsqueda, inserción y eliminación. Una base de datos puede tener más de un índice.

Dos estructuras de datos indexadas muy conocidas y usadas son las tablas de hash y las estructuras de datos en forma de árbol. En las tablas de hash, dada una clave, calculamos, con la función de hash, el índice en la tabla, y obtenemos el valor. En las estructuras de datos arborescentes, hablamos de nodos y hojas, que están en el nivel más profundo del árbol, y que tienen los datos. Acceder a un nodo hoja (lectura / escritura) es una operación de entrada / salida. El primer nodo es la raíz del árbol. Los nodos intermedios tienen punteros a otros nodos, y claves. Las hojas forman una lista doblemente enlazada para navegar entre las páginas eficientemente.

Se puede considerar que cada nodo de un árbol es una página del fichero. Por tanto, obtener un nodo involucra una operación de entrada y salida. Entonces, el número de accesos a disco se corresponde con la longitud del camino del nodo raíz a la hoja. Si cada nodo de un árbol balanceado tiene 100 hijos – que es algo habitual en bases de datos – un árbol de altura 4 tiene 100 millones de hojas, y acceder a ellas nos tomaría 4 operaciones de entrada / salida. Si el árbol fuera binario, aplicando log2(100.000.000) nos requeriría más de 25 operaciones de entrada / salida.

En la asignatura veremos, con cierto nivel de talle, los árboles B+, que son estructuras de datos arborescentes. Las referencias del tema tienen información sobre las tablas de hash si estás interesado/a.

8.3 Árboles B+ (en Inglés, *B+ trees*)

Se utilizan cuando los datos no caben en la memoria. Fíjate que si un árbol no es balanceado, se puede convertir en una lista. Por tanto, se aplican una serie de restricciones / limitaciones. Definiremos un árbol B+ de orden d de la siguiente manera[18]:

- Cada nodo tiene a lo sumo 2d entradas (2d + 1 hijos si son nodos intermedios). Las entradas son pares (Puntero, Clave).
- Cada nodo tiene al menos d entradas (y d+1 hijos si son nodos intermedios).
- La raíz es la excepción en el número de entradas, 1 <= entradas <= 2d (con entrada+1 hijos)
- Los datos se guardan en los nodos hojas (no tienen hijos)
- Los nodos intermedios y raíz guardan punteros a otros nodos y claves (no datos!)
- Las claves pueden aparecer repetidas en una hoja y en un nodo interno
- Los nodos hojas forman una lista doblemente enlazada[19].

Los árboles B+ son balanceados y cumplen la propiedad de búsqueda de los árboles binarios de búsqueda: las claves dentro de los nodos están ordenadas, de menor a mayor, de izquierda a derecha. De manera informal, para una clave K, el valor de todas las claves en el subárbol de la izquierda son menores que K, y el valor de todas las claves en el subárbol de la derecha, mayores

8.3.1 Formato de un nodo

La Figura 18 muestra el formado de un nodo (intermedio y raíz). Las entradas del nodo son pares (clave, puntero), excepto el primer y último puntero.

- P_0 apunta a un árbol cuyas claves son $< K_1$
- P_1 apunta a un árbol cuyas claves son $K_1 \leq K < K_2$
- P_2 apunta a un árbol cuyas claves son $K_2 \leq K < K_3$
- P_m apunta a un árbol cuyas claves son $\geq K_m$

[18] En la bibliografía y en Internet puedes encontrar otras definiciones, que son ligeramente diferentes.
[19] De esta manera, podemos recorrerla sin necesidad de hacer más accesos a disco.

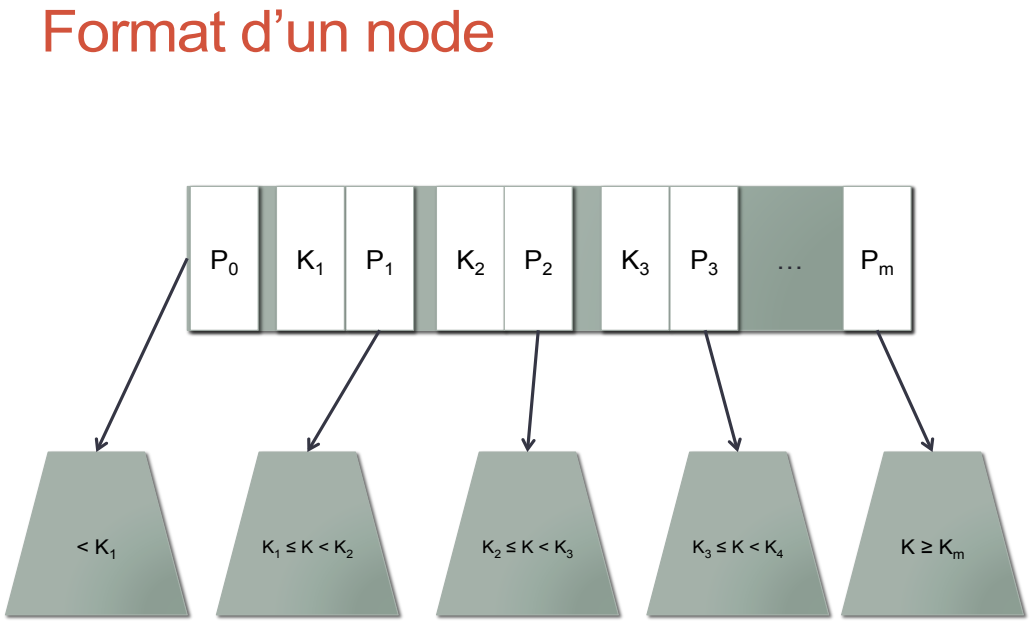


Figura 17: Formato de un nodo

8.3.2 Algoritmo de búsqueda

El algoritmo de búsqueda encuentra el nodo (hoja) en el que se encuentra la clave. El esquema del algoritmo es recursivo. La idea es bajar el árbol hasta encontrar el nodo hoja en el que se encuentra la clave. Para llegar a dicho nodo hoja, debemos tomar decisiones: seguir el camino de la izquierda o de la derecha. Para esto, comparamos el valor de la clave que buscamos con el valor de las claves de los nodos intermedios del árbol. La Figura 19 muestra un esquema del algoritmo. En la asignatura no nos centramos en los algoritmos ni en la recursividad, aspectos que ya has estudiado en anteriores asignaturas, por lo que para centrarnos en los aspectos más relacionados de los árboles B+, algunos detalles los he obviado en los esquemas de algoritmos.

Cerca: algoritme a alt nivell

- ...
- ***cercar (arrel, k)***

  ...

  Si $k < k_1$, ***llavors*** *cercar* (P_0, k)

  Si no

  Si $k \geq k_m$, ***llavors*** *cercar* (P_m, k)

  Si no

  Trobar i tal que $k_i \leq k < k_{i+1}$ *(cerca lineal o binària)*

  cercar (P_i, k)

Figura 18: Esquema algoritmo de búsqueda

Veamos un par de ejemplos. Imagínate que te pido aplicar el algoritmo de búsqueda para encontrar la clave 5 y 14 en el siguiente árbol B+. ¿Cómo lo harías?

Cerca: un parell d'exemples

- Considerem el següent arbre B+ d'ordre 2
- Cada node té entre 2 i 4 entrades

- **Cerca (arrel, 5) -> cerca (P_0,5) ja que 5 < 13**
- **Cerca (arrel, 14) -> cerca (P_1, 5) ja que 13 ≤ 14 < 17**

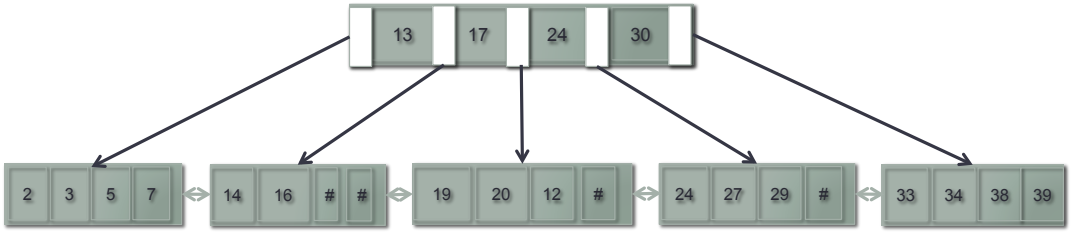


Figura 19: Árbol B+

Primero, en el nodo raíz, los punteros están representados por cajas de color en blanco. El valor de las claves están representadas en las cajas de color verde oscuro.

En el primer caso, como 5 < 13, el algoritmo buscaría por la izquierda, bajando por P_0, y encontraría el valor de la clave en la hoja de más a la izquierda.

En el caso del 14, no se cumple la condición 14 < 13. Tampoco se cumple la condición 14 ≥ 39. Por tanto, tenemos que buscar por el medio. En concreto, bajaríamos utilizando P1, ya que 13 ≤ 14 ≤ 17, y encontraría el valor de la clave en la segunda hoja – comenzando por la izquierda.

8.3.3 Algoritmo de inserción

El algoritmo de inserción es un poco más complicado que el algoritmo de búsqueda, porque tenemos que, en primer lugar, situarnos en el nodo donde nos toca insertar la nueva clave o entrada, y entonces mirar si dicho nodo tiene un exceso de entradas. De manera intuitiva podemos anticipar el algoritmo: si el nodo no está lleno (no tiene exceso de entradas), entonces insertamos la nueva entrada y ya está. Pero si el nodo está lleno, algo tenemos que hacer. En concreto, como verás en el esquema del algoritmo, lo que se hace es dividir el nodo en dos y redistribuir las claves / entradas.

Veamos el esquema del algoritmo paso a paso.

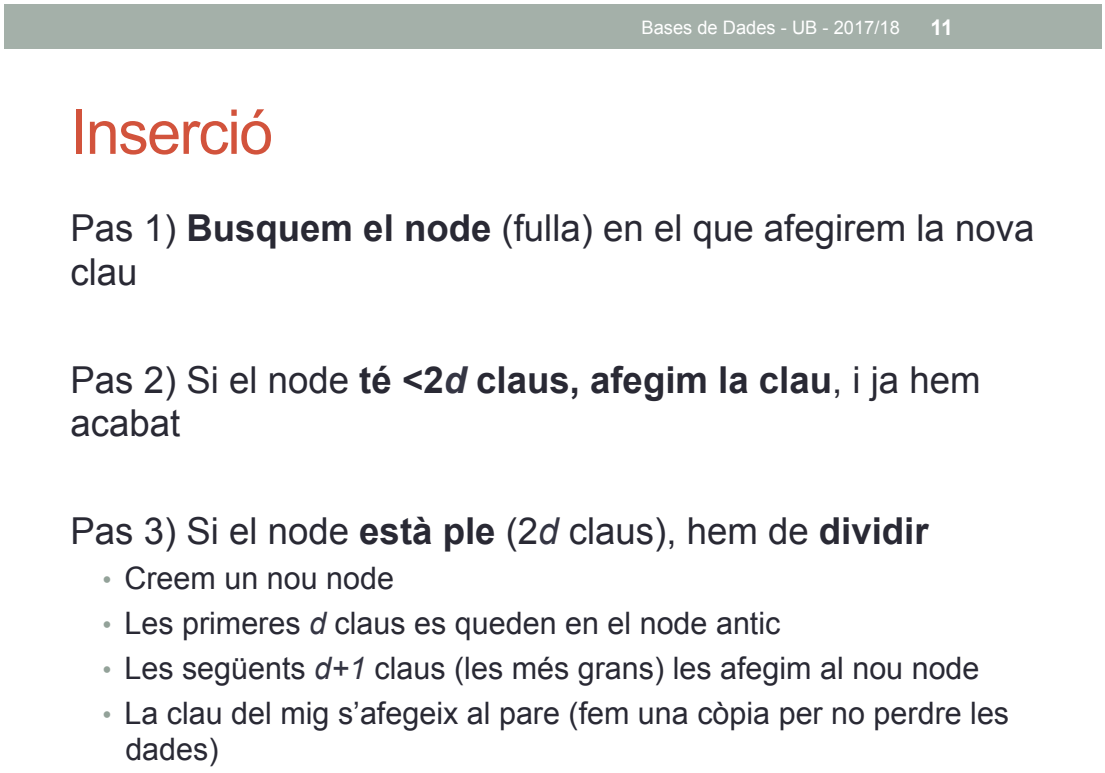


Figura 20: Algoritmo de inserción (pasos 1, 2 y 3)

El primer paso del algoritmo es buscar el nodo (que será una hoja) en el que queremos añadir la nueva clave o entrada. Si el nodo no tiene exceso de claves, entonces añadimos la nueva, y hemos acabado. Pero si, por el contrario, el nodo está lleno, entonces lo dividimos de la siguiente manera. Creamos un nuevo nodo, las primeras d claves – recuerda que d es el orden del árbol; por primeras me refiero de izquierda a derecha – se quedan en el nodo, y el resto se añaden en el nuevo nodo. La clave del medio es importante, porque hemos creado un nuevo nodo, y estamos en una hoja – tenemos datos. Dicha clave se sube al padre y se mantiene en el nuevo nodo.

Ahora puedes pensar que el algoritmo ya ha acabado, pero ¿qué pasa con el padre? Ups! El padre puede estar lleno. Y si lo está, ¿qué hacemos? Volvemos a dividir y a crear un nuevo nodo. Las primeras d claves se quedan en el nodo padre, el resto de claves se mueven al nuevo nodo, y la clave del medio se mueve al padre, pero no se mantiene en el hijo. ¿Porqué? Porque no tenemos datos – solo punteros. Y aplicamos el proceso recursivamente.

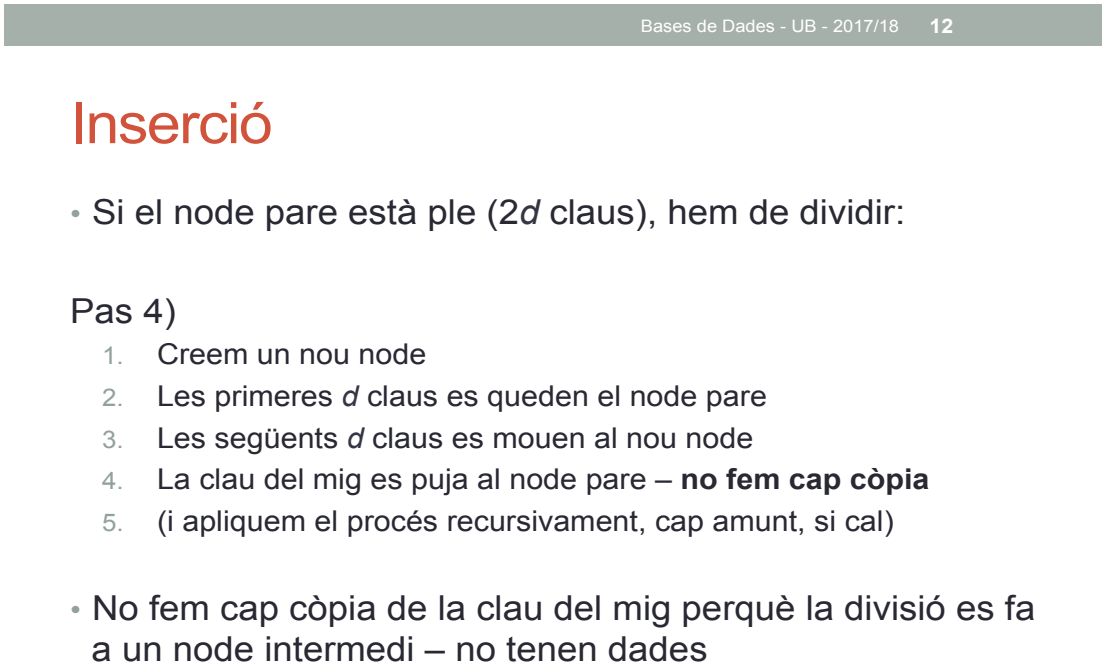


Figura 21: Algoritmo de inserción, paso 4

Veamos un ejemplo. Considera el árbol B+ de la Figura 20 e inserta la clave 8. Si aplicamos el algoritmo, los pasos son:

Paso 1) Buscamos el nodo (hoja) en el que insertar la clave 8. El nodo es el formado por las entradas (2,3,5,7). El nodo está lleno. El árbol B+ es de orden 2, y el nodo tiene 4 entradas, que es el máximo. Por tanto, tenemos que dividir, y pasamos al Paso 3).

Paso 3) Dividimos:

- las primeras *d* (esto es, 2) claves se quedan en el nodo – ahora lo tenemos con (2,3)
- el resto de *d*+1 claves (2 más la que insertamos de nuevo) las ponemos en un nodo nuevo, que está formado por las claves (5,7,8) – las ponemos en orden, de izquierda a derecha, de menor a mayor
- la clave del medio sube al padre, y se mantiene en el nodo. ¿Cuál es la clave del medio? 5 {2,3} – 5 - {7,8}
- Subir la clave 5 al padre significa que tenemos que volver a dividir, porque el padre está lleno. Por tanto, realizamos el paso 4)

Paso 4) Dividimos el nodo padre (13, 17, 24, 30) como sigue:

- las primeras d (2) claves, se quedan en el nodo (5, 13) -> recuerda que subimos el 5, y lo insertamos en orden, de menor a mayor, de izquierda a derecha
- al resto de d claves – sin contar la del medio - (24, 30) se ponen un nuevo nodo
- la clave del medio (17) se sube al padre, que, en este caso, será la raíz.

Recursivamente, vemos que el nodo padre (la nueva raíz) no está llena, y acabamos. El árbol resultante se muestra en la Figura 23:

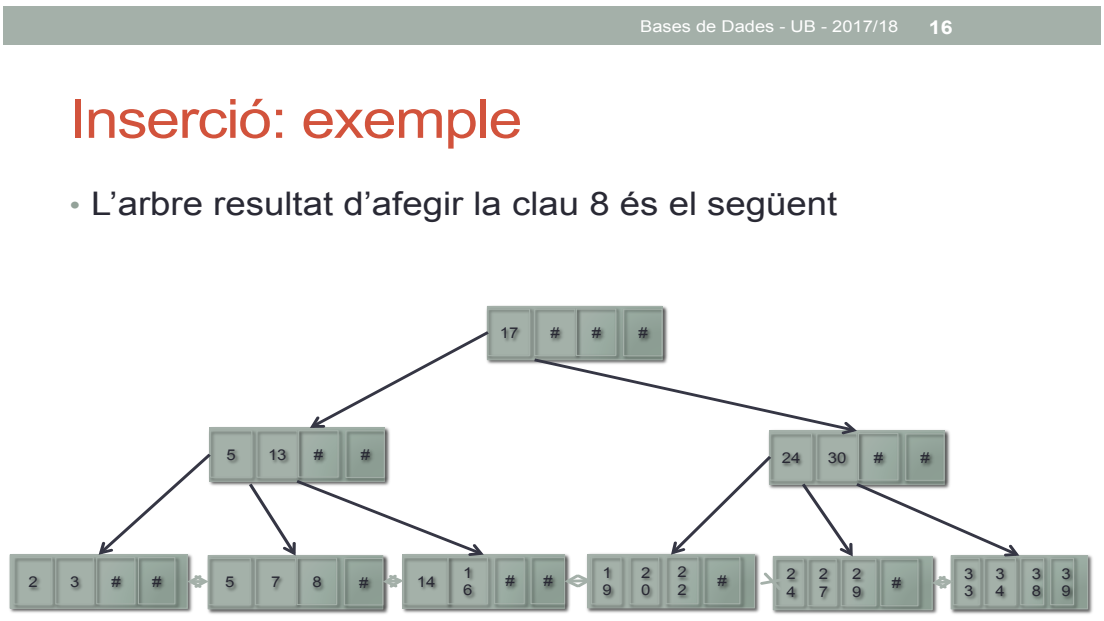


Figura 22: Árbol resultado de insertar clave 8

8.3.4 Algoritmo de eliminación

El algoritmo de eliminación es el más complicado, porque se nos presentan varios casos. Para comenzar, los nodos pueden ser intermedios o hojas – normalmente no se borra la raíz. Es más,

dado que las datos están en las hojas, habitualmente las operaciones de eliminación se realizan en las hojas, pero algún nodo intermedio se puede ver afectado. Además de este caso, los nodos pueden tener un número suficiente de claves después de la eliminación o no. Si no lo tienen, tenemos que diseñar una estrategia para que lo tengan, porque de lo contrario no cumplirían con la definición de un árbol B+. Vamos por casos.

Primero de todo aplicamos el algoritmo de búsqueda para localizar la entrada a eliminar y el nodo (N) en el que se encuentra. Si la entrada no se encuentra, no la podemos eliminar!

Caso 1: Nodo intermedio

Si la entrada se encuentra en un nodo intermedio, y éste tiene un número suficiente de claves después de eliminar la que nos aplica, ya hemos acabado. ¿Qué significa que tiene un número suficiente de claves? Según la definición de árbol B+ de los Apuntes, tiene que tener al menos *d* claves.

Si no tiene un número suficiente, cogemos un nodo adyacente hermano (S), y realizamos los siguientes pasos:

(a) Si S tiene excedente de claves, **redistribuimos** las claves que sobran entre S, N y su padre (para mantener el orden), y actualizamos los punteros

(b) Si S no tiene excedente de claves, **unimos** N y S. Bajamos la clave del padre al nodo de la izquierda. Movemos las claves del nodo de la derecha (M) al nodo de la izquierda. Descartamos M y actualizamos los punteros.

El siguiente paso es recursivo, tenemos que comprobar que los nodos tengan un número suficiente de claves.

Caso 2: Nodo hoja

Si la entrada se encuentra en un nodo hoja con un número suficiente de claves después de eliminar la que nos aplica, ya hemos acabado.

En caso contrario, cogemos un nodo hermano adyacente (S) y realizamos los siguientes pasos:

(a) Si S tiene excedente de claves (más de las necesarias), **redistribuimos** las claves entre N y S de tal manera que cada nodo tenga un número suficiente de claves). Buscamos la clave en el nodo padre relacionada con el nodo de la derecha (M), y substituimos el valor de esta clave por el valor de la clave más pequeña del nodo M.

(b) Si S no tiene excedente de claves, **unimos** N y S. Movemos las entradas de M al nodo de la izquierda. Descartamos M y ajustamos punteros. Eliminamos la clave en el nodo padre que apuntaba a M porque M ya no existe.

El siguiente paso es recursivo, tenemos que comprobar que los nodos tengan un número suficiente de claves.

Veamos un ejemplo. Considera el siguiente árbol B+ y elimina la clave 19.

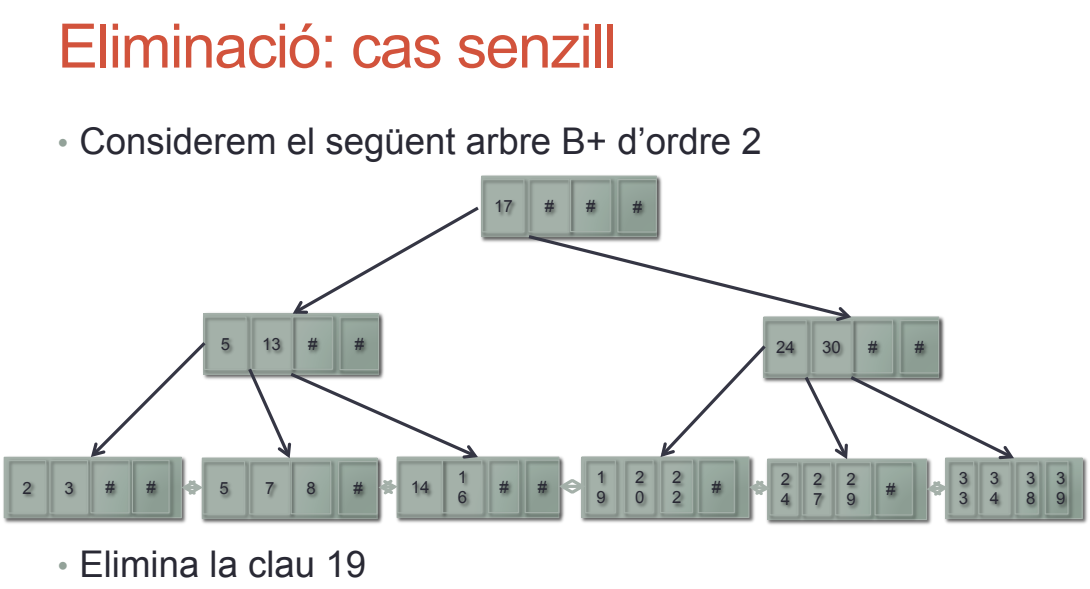


Figura 23: Eliminar clave 19

Buscamos el nodo (N) con la clave 19. Lo encontramos en una hoja (19, 20, 22, #). Por tanto, aplicamos el caso 2. El nodo N tiene un número de clave suficiente después de eliminar 19. Ya hemos acabado. La nueva hoja queda de la siguiente manera (20, 22, #, #).

Ahora eliminamos la clave 20.

Buscamos el nodo (N) con la clave 20. Lo encontramos en una hoja (20, 22, #, #). Por tanto, aplicamos el caso 2. El nodo N no tiene un número de clave suficiente después de eliminar 20. N = (22, #, #, #). Por tanto, cogemos un nodo hermano adyacente, S = (24, 27, 29, #). S tiene excedente de claves, por tanto, aplicamos la redistribución para que ambos nodos tengan un número suficiente de claves: N = (22, 24, #, #), S=(27, 29, #, #). A continuación, buscamos la clave en el nodo padre relacionada con el nodo de la derecha (en este caso, es S). La clave es 24. Y substituimos el valor de esta clave por el valor de la clave más pequeña del nodo S. El resultado se muestra en la Figura 25.

Ahora queremos eliminar la clave 24.

Buscamos el nodo (N) que tiene la clave 24. Lo encontramos en un nodo hoja N=(22, 24, #, #). Aplicamos el caso 2. N no tiene un número suficiente de claves después de eliminar la clave 24, N=(22, #, #, #). Por tanto, buscamos un nodo hermano adyacente, S=(27, 29, #, #), y miramos el número de claves. No tiene exceso de claves (tiene el mínimo), porque lo aplicamos la unión.

Movemos las entradas entre los dos nodos (22, 27, 29, #) y eliminamos la clave en el nodo padre que apuntaba a S porque ya no existe. La clave es 27.

Al eliminar la clave 27 del nodo (27, 30, #, #), tenemos un nodo intermedio con un número insuficiente de claves. Por tanto, seguimos aplicando el algoritmo. En concreto, aplicamos el Caso 1.

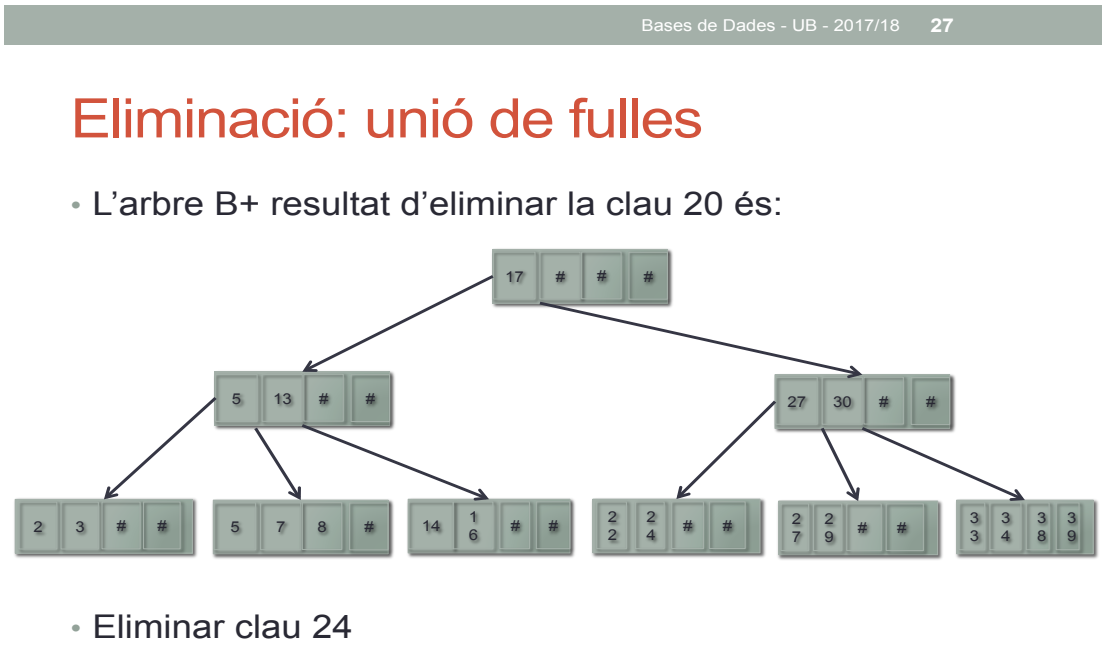


Figura 24: Árbol B+ resultado de eliminar la clave 20

El nodo N = (30, #, #, #) es intermedio y no tiene un número suficiente de claves después de eliminar la clave 27. Por tanto, cogemos un nodo hermano adyacente, S = (5, 13, #, #). Este hermano suyo no tiene un número suficiente de claves, por lo que unimos. Bajamos la clave del padre (que toque) al nodo de la izquierda, (5, 13, 17, #). Movemos las claves del nodo de la derecha al nodo de la izquierda, (5, 13, 17, 30). Descartamos el nodo de la derecha, y actualizamos punteros. Ahora tenemos un nivel menos en el árbol. El árbol resultado se muestra en la Figura 26.

Eliminació: unió de nodes intermedis

- L'arbre resultat d'eliminar la clau 24 és

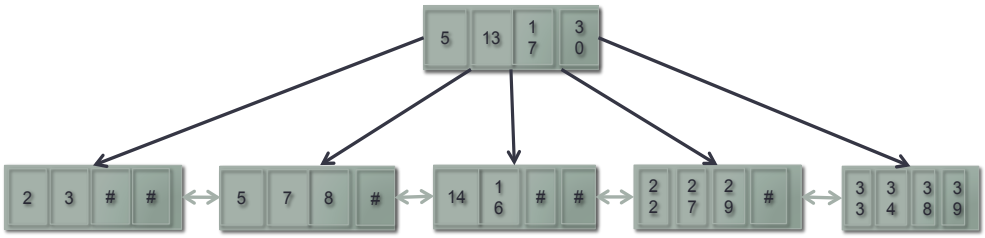


Figura 25: Árbol resultado de eliminar clave 24

El algoritmo de eliminación en árboles B+ se propaga de abajo a arriba, ya que habitualmente se borran los datos en las hojas.

No se ha incluido un pseudocódigo porque mi objetivo del curso es que entiendas cómo funciona y no cómo implementarlo.

El algoritmo de eliminación asume que no hay claves duplicadas. Considerar este caso sería el objeto de un curso más avanzado y no de introducción.

Tanto en el algoritmo como en los ejemplos he considero sólo un nodo hermano. Este nodo hermano puede ser el de su derecha o el de su izquierda – si existen.

8.4 Referencias

- Capítulo 8, 9 y 10 de Ramakrishnan
- Apéndice C de Begg & Connolly

9. TEMA 8 – Algunas pinceladas sobre NoSQL

Los objetivos del tema son:

- Conocer el mundo NoSQL, su motivación y características
- Conocer algunos aspectos importantes de las bases de datos NoSQL
- Reflexionar sobre la respuesta a la pregunta de qué bases de datos escoger
- Establecer la relación entre el NoSQL y el BigData

9.1 Entrando en el mundo NoSQL

Al principio de los Apuntes, comenté, con cierta prudencia, que las bases de datos relacionales son muy populares y utilizadas. El movimiento NoSQL es más rotundo en esta cuestión. Para los profesionales de las bases de datos, la principal pregunta a la que se han tenido de afrontar a lo largo de los años ha sido qué base de datos relacional deberían utilizar. Pero esta situación, con el movimiento NoSQL, está cambiando.

El movimiento NoSQL (No solo SQL) nace de la necesidad de trabajar con un gran volumen de datos (y además, los datos son heterogéneos). Te puedes preguntar si este incremento es una característica del siglo XXI o no. Si miramos hacia atrás, se puede afirmar que siempre ha habido un incremento en el volumen de datos con los que las bases de datos trabajan. Por ejemplo, al principio, los sitios web era visitados por pocos usuarios, con la explosión o *burbuja del punto com*, la cantidad de usuarios (y datos) incrementó considerablemente. El aspecto a destacar para la asignatura es que el incremento en el volumen de datos de hace un pocos años atrás hasta ahora ha sido tal que las bases de datos relacionales no parecen ser capaces de dar respuesta a las necesidades de los usuarios y de los profesionales / desarrolladores.

El principal problema de las bases de datos relacionales es justo su principal característica: las **relaciones**. Todo se reduce a tablas y filas, y el valor de cada elemento de una fila tiene que ser atómico – no podemos almacenar, por ejemplo, una lista de valores. Y esto, ¿porqué es un problema? Porque necesitamos transformar o traducir los datos que queremos almacenar en la base de datos a tablas y relaciones. ¿Porqué no podemos guardarlos directamente con la estructura que tienen – como si fueran objetos en POO?[20] Otro problema o limitación de las bases de datos relacionales es que necesitamos definir el **esquema** de la bases de datos para trabajar con ellas. Sin embargo, no siempre sabemos la estructura del modelo de datos, o dicha estructura es muy variable y difícil de prever… Otro problema es la **escalabilidad**. Al principio, durante la expansión de la web, era suficiente con comprar el mejor servidor de bases de datos. Sin embargo, cuando el

[20] Las bases de datos orientadas a objetos intentaron solucionar este problema. Sin embargo, no acabaron de implantarse. Una de las razones es que el diseño siguiendo la filosofía de objetos – como en POO - no resultó tan claro para los usuarios que desconocían la POO como las relaciones. Además, el modelo ER ya estaba implantado, con sistemas funcionando con bases de datos relacionales...

volumen se transforma en millones de usuarios (y datos relacionados), el mejor servidor (con RAM, disco duro…) no es suficiente, porque el hardware, aunque no nos guste, tiene un límite. En vez de escalar en vertical – mejorando el servidor central de bases de datos- podemos escalar en horizontal – añadiendo servidores de bases de datos.

9.2 Bases de datos NoSQL

Se dice que las bases de datos NoSQL se comenzaron a definir el 2009-2010. Las bases de datos NoSQL no usan (exclusivamente) SQL como lenguaje de consulta, son de código abierto, y trabajan sin un esquema.

Uno de los objetivos de las bases de datos NoSQL es mejorar la **escalabilidad** de las bases de datos relacionales. La escalabilidad la puedes entender como la habilidad de satisfacer las necesidades de los usuarios para varios tipos de carga. Para alcanzar este objetivo, habitualmente se habla de replicación y *sharding*. La primera hace copia de los mismos datos en varios servidores de bases de datos. La segunda involucra el particionado de datos en múltiples bases de datos, alojados en servidores diferentes. En bases de datos relacionales, el *sharding* es un "dolor de cabeza". Si tenemos los datos en diferentes bases de datos, ¿cómo hacemos los JOINS? También te puedes preguntar cómo se pueden garantizar las propiedades ACID de las transacciones con *sharding*. Las bases de datos NoSQL se basan en la replicación y / o en el *sharding* automático.

Otro de los objetivos de las bases de datos NoSQL es mejorar la **disponibilidad** y **flexibilidad** de las bases de datos relacionales. Al trabajar con varios servidores, si uno falla, podemos continuar ofreciendo el servicio – esto es crucial actualmente. Respecto a la flexibilidad, las bases de datos relacionales nos fuerzan a definir el esquema, pero no siempre es posible. Podrías pensar que un desarrollador inteligente podría crear una tabla muy general para cualquier tipo de esquema, pero esto, como te puedes imaginar, no es muy recomendable.

NoSQL nace con el objetivo de que sus bases de datos sean de código libre, para intentar abaratar los **costes**.

8.3 Variedad de bases de datos NoSQL

El equivalente a las propiedades ACID de las (transacciones) de las bases de datos relacionales son las propiedades **BASE**. BA significa "disponible muchas veces, pero no siempre", porque el sistema NoSQL se ejecuta en diferentes servidores. S es para "estado débil", porque los datos, al estar replicados, pueden estar actualizados en un servidor y no en otro. E significa "consistente al final", porque puede haber momentos en los que el sistema de base de datos no sea consistente. Por ejemplo, cuando estamos comprando por Internet y utilizamos el carrito de la compra, no nos interesa que el usuario tenga que esperar hasta que los datos de su carrito estén actualizados en todas las copias de la base de datos que tenemos en los servidores (por cierto, normalmente se habla de clusters, que son un conjunto de servidores). Por ello, preferimos guardar el carrito en la

memoria (RAM) de uno de los servidores, e, internamente, dicho carrito se va "copiando" en los diferentes servidores. Por tanto, en algún momento, un servidor puede no tener actualizado el carrito de la compra del usuario.

8.3.1 Bases de datos orientadas a clave – valor

Son las bases de datos NoSQL más simples. Conceptualmente las puedes entender como una tabla de hash o un vector en el que guardamos parejas de clave – valor. Desde este punto de vista, son rápidas, porque accedemos por la clave. Sin embargo, los valores son opacos. Es decir, no podemos buscar por el contenido de los valores. Las operaciones de búsqueda son a nivel de clave. No suelen imponer restricciones en la estructura de datos que guardamos.

Ejemplos de bases de datos orientadas a clave – valor son

- Redis (https://redis.io/)
- Amazon DynamoDB (https://aws.amazon.com/dynamodb/)
- Riak (http://basho.com/products/riak-kv/)

Este tipo de bases de datos es muy interesante para gestionar las sesiones de los usuarios. Por otro lado, no son muy recomendables cuando nos interesa la relación entre los datos, porque son opacos.

8.3.2 Bases de datos orientadas a documentos

Son una de las más populares del movimiento NoSL. Puedes pensar en estas bases de datos como si fueran clave – valor, en el que el valor es una estructura de datos en forma de árbol, y se puede consultar. Esta es una diferencia clave con respecto a las bases de datos orientadas a clave-valor.

Los documentos combinan estructura y contenido. Piensa en un HTML. JSON es un formato muy utilizado para definir y guardar documentos en este tipo de bases de datos. JSON es un formato de intercambio ligero, que los humanos lo podemos interpretar, basado en JavaScript. Es el sucesor del XML.

Ejemplos de bases de datos orientadas a documentos son:

- MongoDB (https://docs.mongodb.com/manual/introduction/)
- CouchDB (https://en.wikipedia.org/wiki/CouchDB9)

No tienen un esquema definido. En otras palabras, podemos guardar documentos con estructuras diferentes, y los documentos pueden tener a su vez documentos. ¿Ves la potencia de este tipo de bases de datos? Por ejemplo, en vez de realizar varias consultas, con una consulta puedes obtener todos los documentos de un documento – la guardas en memoria, y su acceso, como sabes, es más rápido que a disco.

8.3.3 Bases de datos orientadas a grafos

Las anteriores bases de datos no se preocupan de las relaciones. Sin embargo, muchas veces nos interesan las relaciones entre los elementos de una base de datos. Un ejemplo típico son las redes sociales: los nodos son los usuarios, y las relaciones entre los usuarios son de amistad, de *like*, etc.

Las bases de datos orientadas a grafos nos permiten guardar nodos y sus relaciones, y no tenemos que definir el esquema. Es decir, podemos recorrer el grafo como queramos.

Estas bases de datos no suelen utilizar *sharding*, porque los nodos de un grafo pueden estar relacionados con cualquier otro nodo.

Ejemplos de bases de datos orientadas a grafo son

- FlockDB (https://github.com/twitter-archive/flockdb)
- OrientDB (http://orientdb.com/orientdb/), Neo4J (https://neo4j.com/)

¿Cómo se consulta un grafo? Tenemos lenguajes como Gremlin o Cypher que están especialmente diseñados para consultar grafos.

8.3.4 Bases de datos orientadas a columna

Muchos de nosotros tenemos una mentalidad de fila. Vemos los datos organizados horizontalmente, y de izquierda a derecha. Sin embargo, en ocasiones, no queremos procesar todas las columnas de una fila. De hecho, podemos querer procesar el valor de una columna en todas las filas. Las bases de datos orientadas a columnas almacenan las columnas como si fueran filas.

Al estar las columnas guardadas en disco, las consultas que tienen que acceder al valor de las columnas están optimizadas, porque los valores están en el mismo bloque de disco. El lado negativo es la inserción, que no está optimizada.

BigTable de Google es el primer ejemplo de bases de datos orientadas a columna. BigTable se diseño para servicios web muy grandes, como Google Earth. Otros ejemplos de bases de datos orientadas a columna son

- MonetDB (https://www.monetdb.org/)
- Cassandra (https://en.wikipedia.org/wiki/Apache_Cassandra)

Las bases de datos orientadas a columna son recomendables para *data warehouses* – almacén y análisis de muchos datos.

8.4 Una breve (pero importante) reflexión

Un aspecto fundamental a principios del siglo XXI es que en vez de seleccionar una base de datos relacional porque todo el mundo lo hace, necesitamos entender la naturaleza de los datos y cómo vamos a manipularlos.

8.5 Conectando NoSQL con BigData

Las bases de datos NoSQL permiten el almacenamiento de BigData, que se puede definir como la habilidad de la sociedad para controlar y hacer uso de una gran cantidad de datos de manera que se produzcan nuevos servicios de gran valor. A nivel más técnico, también se suele definir en clave de las 3 Vs: Volumen, Velocidad, Variedad.

Para darte algunos datos, según las referencias que cito al final del capítulo, Google procesa más de 24 Petabytes (1 Pb = 1000 Tb; i 1Tb = 1000 Gb) de datos por días. Facebook recibe 10 millones de nuevas fotos cada hora. 400 millones de tweets se generaron por día en el 2012. Aunque estos datos pueden variar en el momento en el que estés leyendo el documento, te ofrecen una perspectiva del volumen de datos del que estamos hablando.

El Big Data se dice que cambiará el mundo porque nuestra "obsesión" por entender la causalidad de las cosas se transformará en las correlaciones: no será el porqué sino el qué. Podemos predecir que los precios (de los billetes de avión, por ejemplo) subirán o bajarán, pero no sabemos (o no nos interesa) el porqué. ¿Qué piensas? ¿Dónde nos deja a los humanos – aprender de la experiencia, intuición?

Esta pregunta te puede parecer un poco alejada del contenido de los apuntes y de la asignatura, y seguramente lo sea, pero como futuro ingeniero o ingeniera, la implicación de nuestro trabajo e investigación en la sociedad es un aspecto muy importante, y quería acabar estos apuntes con un toque "reflexivo".

8.6 Referencias

- Harrison, G. (2015). *Next generation databases*. NoSQL, NewSQL, and BigData. apress
- Kitchin, R. (2014). *The Data Revolution: Big Data, Open Data, Data Infrastructures and their consequences*. SAGE
- Mayer-Schönberger, V., Cukier, K. (2014). *Big Data: A revolution that will transform how we live, work, and think*. John Murray Publishers.
- Saddle, P., Fowler, M. (2013). *NoSQL Distilled: A brief guide to the emerging world of polyglot persistence*. Person Education.
- Sullivan, D. (2015). *NoSQL for mere mortals*. Addison-Wesley.