Bases de Datos

Apuntes de la Asignatura

Ángel Manuel Soria Gil
Universidad de Málaga
E.T.S. de Ingeniería Informática
Curso 2023/2024

Índice general

Capítulo 1

Introducción a las Bases de Datos.

A lo largo de este capítulo se darán a conocer ciertos aspectos teóricos y prácticos de los sistemas de bases de datos. Veremos su evolución histórica solucionando problemas que surgían según se demandaban nuevas funcionalidades, aprenderemos a clasificar según distintos criterios las bases de datos y por último veremos qué es un SGBD.

1.1. Casos prácticos introductorios

Ejemplo 1 Supongamos un caso hipotético de una entidad bancaria. Dicha entidad pretende almacenar datos sobre los trabajadores que tienen en nómina. Buscan guardar nombre, DNI, dirección, teléfono, correo electrónico, salario... En dicha entidad son un total de 10000 empleados. Nos surge una primera pregunta básica: ¿cómo almacenamos sus datos?

A la mente del estudiante en informática se le pasa como primera opción por la cabeza un array, pero esto lleva consigo todos los problemas que tiene asociado el manejo de arrays a la hora de duplicarlos, modificar, acceder a posiciones no válidas... y además, ¿y si se va la luz? Los datos almacenados en el array están allí mientras el programa se ejecuta. Si nos quedamos sin batería o se va la luz y el programa cesa su actividad de forma repentina perdemos todos los datos. Es por esto que pasamos al sistema de ficheros, aún así, el resto de problemas sigue.

Ejemplo 2 Ahora supongamos que se quiere hacer una transferencia de una cuenta bancaria a otra. Necesitamos primero leer el saldo de la cuenta de origen, después leer el saldo de la cuenta destino, calcular el nuevo saldo de la cuenta origen, grabarlo, calcular el nuevo saldo de la cuenta destino y grabarlo. Imaginemos que durante la ejecución, antes de grabar el saldo en la cuenta de destino, ocurre un error y el programa se interrumpe. ¿Qué ocurre con el dinero que se ha pasado de la cuenta origen a la de destino? ¿Se pierde? Conocemos por lo que ocurre en la vida cotidiana que no. Para asegurar que esto no ocurre deben establecerse unos estados válidos a los que el programa pueda llegar, en caso de interrumpirse y no llegar a un estado válido, se queda en el estado de origen.

Ejemplo 3 Con el mismo caso anterior, imaginamos ahora que queremos sacar dinero de una misma cuenta de forma simultánea desde distintos cajeros. Si del primero saco 200 euros y desde el segundo 100 pero comienzo la segunda operación mientras la primera aún no ha guardado el saldo, no se me mostrará la información actualizada en el segundo y este utilizará los datos sin haber descontado los 200 euros iniciales de forma que el saldo final que se grabará será con solamente 100 euros menos pero hemos sacado 300. Un banco no va a permitir que esto pase. Es por ello que no hay simultaneidad de operaciones, primero se realiza una acción y luego otra.

Ejemplo 4 Ahora supongamos que tenemos una serie de datos almacenados y disponemos de una copia de seguridad suya. Se ejecutan ciertas operaciones durante la ejecución ocurre un error grave y se deben recuperar los datos de la copia de seguridad. Debemos asegurar que las operaciones posteriores a esta copia de seguridad no queden perdidas, si no que una vez restaurada se vuelvan a ejecutar.

1.2. Evolución de las Bases de Datos.

1.2.1. Antecedentes.

Las bases de datos tal y como las conocemos hoy en día han evolucionado para adaptarse a las exigencias de las tecnologías actuales. Comenzaron con los primeros discos duros y unidades de almacenamiento permanente. Surgieron de la necesidad de almacenar y recuperar datos de forma eficiente. Con el abaratamiento de estas primeras memorias permanentes se hizo posible el almacén de grandes cantidades de datos hasta llegar al día de hoy.

1.2.2. Actualidad y Recorrido.

En la actualidad las bases de datos (BBDD) tienen una gran presencia en nuestro día a día aunque no lo parezca. Por ejemplo, todas las redes sociales tienen una base de datos funcionando por debajo y allí se almacenan nuestras publicaciones, seguidores, likes... Otro ejemplo es Amazon. Su gran catálogo de productos queda recogido en una gran base de datos. Los sistemas de bases de datos evolucionan actualmente siguiendo dos principales enfoques: hacia los datos y hacia el proceso.
El **enfoque hacia el proceso** establece un diseño flexible de la base de datos que se adapta para admitir y facilitar la ejecución de procesos. Por ejemplo, un sistema de gestión de pedidos de una tienda se organiza hacia el proceso adaptando la base datos para facilitar la gestión de pedidos y pagos correspondientes.
El **enfoque hacia los datos** por su parte tiene un diseño más rígido y hace énfasis en la calidad de datos, precisión, coherencia y facilidad para compartirlos. Servicios de marketing enfocados a la gestión de clientes suelen ser ejemplos de este enfoque.

Además, en los últimos años se está abandonando el paradigma de ficheros, que no es más que un enfoque de almacenamiento de datos basado en archivos independientes. Esto sucede puesto que el paradigma de ficheros acarrea ciertas desventajas como:

- Dificultad para compartir datos.
- Falta de flexibilidad en seguridad.
- Reglas de Integridad de Software, esto es, se han de verificar ciertas precondiciones para realizar acciones concretas.
- Dificultad para responder a nuevas necesidades y necesidad de programación específica para cada formato.

1.3. Clasificación de las Bases de Datos.

1.3.1. Clasificación de Bases de Datos según su Objetivo.

• *Bases de Datos de Propósito General:* Son bases de datos diseñadas para manejar amplia variedad de tipos de datos y ser utilizadas en múltiples aplicaciones. Por ello han de ser versátiles y adaptables a diversas necesidades. Se caracterizan por esta flexibilidad, escalabilidad, uso fácil con interfaces intuitivas, soporte para consultar y concurrencia en transacciones asegurando la integridad de datos. Ejemplo de ello son MongoDB o PostGreSQL.

• *Bases de Datos para el análisis o Data Warehousing:* Son bases de datos preparadas para almacenar cantidades ingentes de datos recopilados e integrados de múltiples fuentes. Tienen gran importancia en los sistemas de infomación de organizaciones pues gracias a estas bases de datos se realizan análisis de todos esos datos transformándolos en información que proporcione una ayuda para tomar una decisión o incluso tomarla directamente. Para conseguir esto los datos deben cumplir tres cualidades básicas: estructuración, coherencia y fácil acceso.

• *Bases de Datos en Memoria:* Son muy similares a las Data Warehousing con la salvedad de que estas se almacenan en memoria principal para disminuir tiempos de acceso y así conseguir una respuesta más rápida. Esto ha sido posible gracias a la computación en 64bits, servidores multi-core y bajada de precios de memoria RAM. Además, responden ante problemas como la pérdida de alimentación pues además de en la RAM, se va almacenando la infomación en páginas de memoria no volátil y se controlan las transacciones que han sido realizadas, cuales no y en caso de fallos recuperar desde donde se dejó sin que se pierda nada.

• *Bases de Datos de Alta Disponibilidad:* Las bases de datos de alta disponibilidad podrían resumirse en bases de datos que son accesibles mucho tiempo, de ahí el nombre de alta disponibilidad. Ejemplo de estas podrían ser las bases de datos utilizadas por

aplicaciones de mensajería como WhatsApp. Para esta alta disponibilidad casi a tiempo completo necesitamos tres factores clave: la redundancia de datos, es decir, que haya varios servidores distintos en ubicaciones distintas con varias copias y que en cualquier momento estén todos listos para pasar a ser el actor principal; supervisión de los recursos de los que se dispone; y failover, es decir, que los recursos en espera estén listos para pasar a principal. Las bases de datos Oracle de OCI son un ejemplo de alta disponibilidad.

1.3.2. Clasificación de Bases de Datos según su Ubicación.

• *Bases de Datos Locales:* Las bases de datos locales son aquellas en que tanto los datos, como el hardware, como el software que emplean la base de datos y el sistema gestor de la base de datos se encuentran en un mismo dispositivo o en una misma red de área local (LAN). Esto aporta seguridad en cuanto a accesos pero añade sobrecarga a la base de datos.

• *Bases de Datos Centralizadas:* Las bases de datos centralizadas son aquellas que almacenan los datos en un único dispositivo físico pero facilitan el acceso a los usuarios desde distintos terminales. Para asegurar la seguridad se mantiene un registro de acceso a los datos.

• *Bases de Datos Distribuidas:* Las bases de datos distribuidas son aquellas que se almacenan en distintas ubicaciones físicas ya sea replicando la base completa o solamente una parte. Estas pueden ser homogéneas en caso de que se utilice el mismo hardware y software en las distintas ubicaciones o heterogéneas si no son los mismos.

1.3.3. Clasificación de Bases de Datos según su Licencia.

• *Bases de Datos de Pago:* En ellas la empresa propietaria de la base de datos tiene los derechos sobre el software y los datos, los usuarios deben adquirir una licencia mediante pago para poder hacer uso de estos. Ejemplo de ello es Oracle.

• *Bases de Datos de Software Libre:* Estas bases de datos son de uso público y gratuito. Los clientes tienen acceso al código del software e incluso pueden modificar ciertos aspectos de la base de datos. Ejemplo de ello es MySQL o PostgreSQL.

• *Bases de Datos de Libre Uso:* Son bases de datos de uso libre, es decir, su uso es gratuito para todo el mundo pero no se tiene acceso al código del software ni a la modificación de la propia base de datos. Ejemplo de ello es BigTable de Google.

1.3.4. Clasificación de Bases de Datos según su Modelo de datos.

• *Bases de Datos Jerárquicas:* Como su nombre indica son bases de datos en que la información se almacena en estructuras jerarquizadas como son los árboles. Así se establecen las relaciones entre los datos aunque esta forma es muy sencilla y fácil de entender, tiene problemas para el acceso a los datos (hay que pasar por todos los padres hasta llegar al nodo que queramos) y se queda corta pues no permite relaciones entre "hermanos".

• *Bases de Datos en Red:* Son una evolución de las bases de datos jerarquizadas en las que se emplean grafos en lugar de árboles para guardar la información. Así todos los nodos del grafo (registros con distintos atributos) pueden relacionarse entre sí.

• *Bases de Datos Deductivas:* Una base de datos deductiva es un sistema de almacenamiento que a partir de unos datos es capaz de inferir o deducir información. Para ello utiliza una serie de reglas que se basan en la lógica básica. Suelen usar un lenguaje declarativo (Prolog). El principal problema que presentan es la dificultad para desarrollar procesos lógicos eficientes para la deducción de información y realización de consultas.

• *Bases de Datos Orientadas a Objetos:* Son aquellas bases de datos en que la información se almacena en objetos. Así en un único objeto tendremos todos los atributos del mismo y los métodos necesarios para el acceso a los datos y consultas. Su principal ventaja es su integrabilidad con los lenguajes de programación orientados a objetos y que se comportan bien con transacciones ACID.

• *Bases de Datos XML:* son un sistema de software que da persistencia a los datos en formato XML. Son asociadas a bases de datos documentales de forma que se pueden exportar, importar, serializar o consultar datos en documentos en formato XML. Dentro de ellas destacan las bases de datos con XML habilitado, que simplemente permiten obtener estos documentos en el formato XML, o con XML nativo, es decir, que utilizan documentos XML como unidad elemental de almacenamiento de la base de datos.

• *Bases de Datos RDF:* son bases de datos basadas en RDF (resource description framework) que son una familia de especificaciones para compartir datos en la web de forma homogénea y permite relacionar los datos mediante modelado en tupletas y estructuras de grafos.

• *Bases de Datos Relacionales:* estas se basan en organizar los datos distribuidos en tablas, es decir, los datos se estructuran en filas y columnas y entre tablas se relacionan los datos a partir de una clave principal y otra foránea que permiten establecer uniones entre tablas. En ellas los datos siguen las reglas de representación ACID que se explicarán a continuación.

• *Bases de Datos NoSQL:* estas bases de datos, son una categoría general de sistemas de gestión de bases de datos que se diferencia de las relaciones en que no tienen esquemas ni permiten JOINs. Tampoco siguen las reglas ACID, siguen otras llamadas BASE, aún así, son un tipo de almacenamiento estructurado. Estas estructuras pueden seguir alguna de las siguientes taxonomías: clave-valor (Voldemort), almacenes de familia de Columnas (cassandra, hbase), almacenes de documentos (mongoDB), grafos (allegroGraph, VertexBD).

1.3.5. ACID vs BASE.

En el mundo de las bases de datos relacionales se necesitan garantizar ciertas condiciones y los datos han de seguir unas reglas para conseguir un funcionamiento correcto. Estas se conocen como las reglas ACID que han de verificar las transacciones relacionales. Son las siguientes:

- Atomicidad: esta propiedad asegura una operación se realiza completamente o no se realiza, es decir, hay una serie de estados válidos en los que podemos viajar pero nunca nos quedaremos en mitad entre dos estados.
- Consistencia: hace referencia a que los datos han de asegurar unas condiciones que define el programador para que la base de datos funcione correctamente.
- Aislamiento: unas operaciones no afectan a otras, así hay un solo acceso a la vez.
- Durabilidad: tras realizarse una acción esta perdura a no ser que haya otra acción que la deshaga.

Por su parte en las bases de datos no sql se siguen unas reglas un poco más optimistas que siguen el modelo BASE que verifica:

- Disponibilidad básica (basic availability): el almacén funciona en la mayoría de tiempo incluso ante fallos gracias a la existencia de almacenamiento distribuido y replicado.
- Soft-state: los almacenes no han de ser consistentes de por sí, si no que el programador puede verificar si es consistente o no.
- Eventual Consistency: puede ser consistente pero solo eventualmente.

1.4. Arquitectura.

1.4.1. Arquitectura en tres niveles.

Definición 1 El informe ANSI/X3/SPARC también conocido como Standard Planning and Requirements Committee of the American National Standards Institute on Computers and Information Processing establece que una base datos sigue una arquitectura en tres niveles:

- Externo: un nivel de la base cercano a los usuarios, es la visión que tienen los usuarios de los datos que utilizan y las operaciones que pueden hacer sobre ellos. Así, existen varios perfiles a nivel de usuario que facilitan distintas vistas.
- Conceptual: en cuanto a contenido global, proporcional nombres de las entidades, características, relaciones entre ellas...
- Interno: es una descripción a nivel físico de la base de datos, Se especifican las estructuras de almacenamiento, métodos de recuperación eficiente, dispositivos...

Definición 2 Se define un esquema como la descripción de la estructura de un nivel de la base de datos.

Así se tienen los siguientes lenguajes para trabajar en los diferentes niveles:

- Lenguajes de definición de datos (DDL): lenguaje empleado para definir esquema externo, conceptual e interno.
- Lenguajes de manipulación de datos (DML): lenguaje para realizar consultas y actualizaciones sobre la base de datos.
- Lenguajes de programación: es un lenguaje propio de la base de datos, van más allá de las consultas y actualizaciones, por ejemplo, SQL o PL.

Con esto se consiguen los objetivos de independencia de datos (podemos modificar un nivel de la base de datos sin que se vean afectados otros niveles superiores), independencia lógica (podemos modificar la estructura de la base de datos sin que las aplicaciones que la usan se vean afectadas) e independencia física (se puede alterar la estructura física y su ubicación sin alterar a la estructura lógica). Así podemos definir el funcionamiento de la arquitectura de la base de datos en el siguiente esquema.

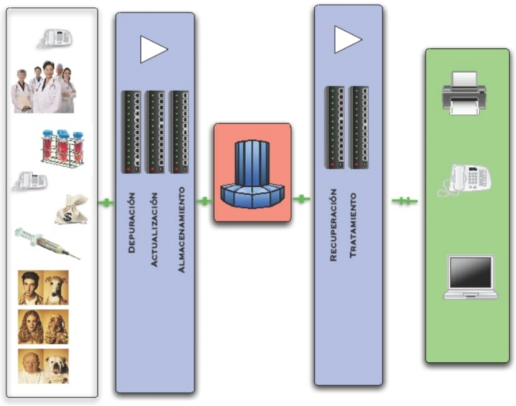


Figura 1.1: Funcionamiento de la arquitectura de una base de datos.

1.4.2. Elementos de una Base de Datos.

Hasta ahora hemos hablado de muchos componentes, tipos de bases de datos, ideas sobre ellas... pero no hemos definido aún qué es exactamente.

Definición 3 Se define una base de datos como una colección de datos lógicamente relacionados entre sí con una definición y descripción comunes, es decir, en un mismo lenguaje, y estructurados de forma particular. Además ha de ser eficiente, con datos independientes, íntegros y seguros de forma que se garantice el acceso concurrente.

Definición 4 Se define un sistema gestor de base de datos como un conjunto coordinado de programas, procedimientos, lenguajes y demás elementos que suministra a los usuarios y administrador los medios necesarios para describir, recuperar y manipular los datos integrados en la base de datos asegurando su confidencialidad y seguridad. Por consiguiente, se definen sus tareas como la creación y manutención de objetos de la base de datos, modificar y operar con los datos, rutinas de recuperación, mantenimiento de seguridad y reglas de integridad.

Definición 5 Se define un sistema de bases de datos como la unión del sistemas gestor de bases de datos y la propia base de datos.

1.4.3. El administrador.

Dentro de los usuarios hay uno que destaca en especial, por lo que se le da el nombre de administrador.

Definición 6 Se define el administrador de una base de datos como un usuario experto que gestiona la disponibilidad de la base de datos, describe esquemas y crea la base de datos, gestiona sus estructuras físicas, gestiona los dispositivos de almacenamiento y seguridad además de la administración de la red, recuperación, backups y afinamiento de la base de datos.

Capítulo 2

Diseño lógico de Bases de Datos.

Tras el primer capítulo teórico de introducción para conocer los conceptos de las bases de datos vamos a tratar de conocer cuál es el ciclo de vida del diseño de una base de datos y a conocer como desarrollar un modelo entidad relación para la utilización de la base.

2.1. Ciclo de vida del diseño de una base de datos.

El diseño de una base de datos consta de cuatro partes distinguidas. La primera de ellas es el análisis de requisitos, el diseño lógico, diseño físico y por último monitorización y afinamiento.

Análisis de requisitos:
Para construir una base de datos se necesita saber para qué se va usar. Para ello se realiza una entrevista con los usuarios de donde se recogen que datos van a emplearse, relaciones que hay entre los datos, restricciones que habrá para ello y la plataforma software que se vaya a emplear en caso de haber una ya existente.

Diseño Lógico:
Consta de cuatro partes.

- Diseño conceptual: se trata de realizar un esquema general en el que se recogen todos los datos y relaciones. Se conoce como modelo entidad relación.
- Integración de vistas: no va a poder hacer lo mismo un gerente de la empresa que el jefe. Es por ello que se necesitan varias vistas que necesitan integrar en una única forma consciente eliminando la redundancia e inconsistencia.
- Modelado relacional: este modelado relacional se basa en tablas que son creadas generalmente con el lenguaje SQL y obtenidas a partir de unas reglas de transformación.

- Normalización: esta consiste en eliminar anomalías y redundancia.

Diseño físico:
En esta etapa se crean métodos de acceso, particiones, clusters, rendimiento y en ocasiones ha de aplicarse desnormalización para conseguir eficiencia.

Monitorización y afinamiento:
Se implementa la base de datos en DDL y se realizan modificaciones para conseguir los requisitos y rendimiento esperados.

2.2. Modelo Entidad Relación

Nuestra tarea será recibir un requerimiento con las tareas y requisitos que nos piden y abstraernos para identificar los sujetos del modelo, propiedades e interconexiones entre ellos. Para ello vamos a necesitar de una serie de definiciones.

2.2.1. Introducción al modelo E/R. Entidades y Relaciones.

Definición 7 Se define una entidad como un ser distinguible del mundo real.

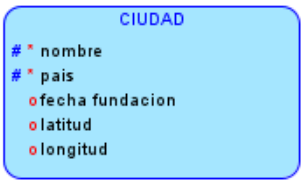
Una entidad debe ser un objeto de interés para nuestro cliente, es decir, si formamos parte del departamento de cardiología de un hospital, me interesarán asuntos sobre el presupuesto que se destina a nuestro departamento, pero la contabilidad de la cafetería no nos interesa. Además, han de ser objetos reales o abstractos que se identificarán a partir de un nombre o sustantivo y servirá como almacén u objeto que recoge datos y organiza la información.

Definición 8 Se define un atributo como una propiedad de las entidades.

Los atributos son utilizados para describir las entidades y es donde se recogen los datos que vamos a tratar. En una entidad se recogerán varios atributos. De este modo para la representación de una entidad se utilizan diagramas de entidades que consisten de rectángulos suaves con un único nombre en mayúsculas y en singular que sea el nombre identificador de la entidad y con una serie de atributos en minúsculas. Por otro lado tendremos restricciones de identificación, es decir, habrá un conjunto de atributos de la entidad que permitan distinguir a unas instancias de otras de la misma entidad.

Definición 9 Se define una clave como un conjunto de atributos que definen inequívocamente a una instancia de la entidad.

Ejemplo 5



Entidad ciudad con una serie de atributos. Los atributos marcados con #∗ son clave de la entidad. No obstante, puede no ser única. Latitud y longitud también podrían ser clave, a estas se les llama claves candidatas.

Definición 10 Se define una relación como una conexión entre entidades.

Estas conexiones pueden distinguirse según las siguientes características: el esquema binario, si son bidireccionales, son nombradas, el orden y la obligatoriedad. Representaremos un relación como una línea que conecta dos entidades y con un nombre que será escrito en minúsculas. Dependiendo de si la línea es continua o no estaremos hablando de relaciones obligatorias o no, respectivamente. Además, si nos fijamos en la terminación de la línea, podemos tener relaciones de uno a uno o uno a muchos. Esto es lo que llamamos grado. También suele darse el caso de que la relación es obligatoria para una parte, pero no para la otra. Es más, a la hora de crearlos, primero deberemos crear una de las dos partes, sin relacionarla obligatoriamente con nadie y luego la otra parte con la relación, pues en caso de ser la relación obligatoria por ambos lados no podría crearse puesto que hay que tener a una de las partes antes que la otra pero la una no puede ser creada sin la otra. Puede esto expresarse de forma más sencilla diciendo que se crea antes el continente que el contenido.

Ejemplo 6

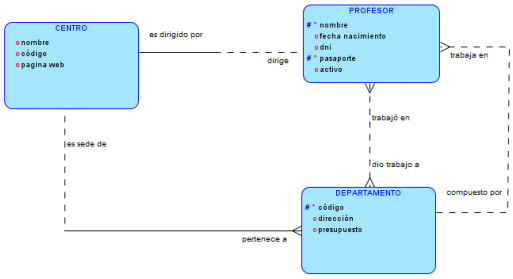


Figura 2.1: EL ejemplo acoge todas las relaciones vistas. Hay una relación obligatoria entre las entidades centro y profesor. Es obligatoria solo por el lado del centro, pues un centro ha de ser dirigido sí o sí por un profesor, pero no todo profesor tiene obligatoriedad de dirigir. Además, es una relación uno a uno, pues solo un profesor dirige el centro y un profesor dirige un solo centro. Sin embargo, el centro está formado por muchos departamentos de los que es sede. El resto de relaciones son similares.

2.2.2. Atributos. Entidades débiles y subentidades.

Recordamos la *Definición 8* de atributo. Además de ser una propiedad o descripción de las entidades, los atributos son datos del mundo real que han de cumplir las siguientes propiedades:

- Atomicidad: No tienen estructura, lo que implica que por ejemplo, no hay direcciones dadas por calle, número, código postal... si queremos esos datos, se han de tener por separado.
- No son computables a partir de otros atributos. Esto quiere decir que vamos a reducir el número de datos que guardamos al mínimo. Por ejemplo, si tenemos la fecha de nacimiento, no guardaremos la edad, pues podemos calcular esta a partir de la fecha de nacimiento y además, la edad necesitaría ser actualizada.
- Son mono-evaluados, es decir, no puedo tener por ejemplo dos números de teléfono, se guardan como teléfono primario y teléfono secundario. Es por esto que decimos que los atributos son singulares. Pueden ser requeridos.
- Un atributo nunca representará a otra entidad, pues esa es la labor de las relaciones.

Además recordamos la *Definición 9* sobre claves. Esto nos lleva a una **Regla de Completitud**. Esta establece que siempre puede determinarse un subconjunto de atributos que identifica a una instancia de una determinada entidad. Estas claves pueden ser simples y estar definidas por un solo atributo pero ya vimos que se definen como un conjunto de claves, es decir, las claves puede ser compuestas y estar formadas por varios atributos. Además dijimos en el Ejemplo 5 que pueden haber varias claves posibles. En este caso tenemos una clave primaria y claves candidatas. Las primarias ya se vieron, las secundarias pueden ser atributos no obligatorios, por ello no hay una forma concreta de representación pero sí se da la unicidad del conjunto de atributos. Esta unicidad de indica con el símbolo $\cup$.

Ejemplo 7

Figura 2.2: Latitud y longitud representan la clave primaria, por otro lado, el conjunto de nombre país y nombre ciudad son un clave secundaria que en este caso no son datos a recoger obligatoriamente, en caso de ser considerados como obligatorios serían representado con el símbolo $\cup*$.

Definición 11 Se define una entidad débil como una entidad que toma su clave o parte de su clave prestada de otra entidad.

Vamos a ver cómo funcionan en el siguiente ejemplo.

Ejemplo 8

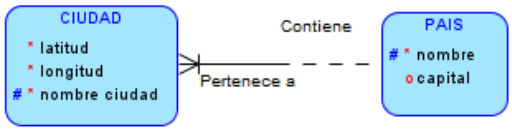


Figura 2.3: Hemos hecho una representación distinta en la que además de la entidad ciudad tenemos la entidad país. Así que el nombre del país ya no se guarda en la ciudad. Además hemos cambiado la clave de ciudad, ya no son latitud y longitud, es la clave que en el ejemplo anterior considerábamos candidata. No obstante, el nombre del país debemos tomarlo prestado de la entidad país, para ello se estable una relación de identificación entre ciudad y país. Esta relación se representa como se ve en el diagrama.

Ejemplo 9 También se nos pueden dar casos en que necesitemos añadir un atributo y ponerlo en dos sitios a la vez. Pongamos un ejemplo para verlo más claro. Si queremos guardar la nota que tiene un alumno en una asignatura. ¿Dónde ponemos la nota? ¿En la asignatura o en el alumno? Si la ponemos en el alumno tenemos un problema porque el alumno tendrá más asignaturas, así que no es una única nota, pero tampoco vamos a poner una nota por asignatura dentro del alumno porque cada año cambia de asignaturas... Si lo ponemos en la asignatura, la asignatura tiene varios alumnos y no todos tienen esa nota... Necesitamos una entidad intermedia que relacione la asignatura y el alumno para poner la nota.

Cabe destacar que el papel de las relaciones reflexivas, es decir, una entidad puede relacionarse consigo mismo. Por ejemplo, a la hora definir una relación jefe-empleado dentro de una empresa, el jefe es también un empleado, así que se relaciona con los otros empleados mediante una relación reflexiva, no creamos una clase especial jefe. No obstante, puede que el jefe tenga algún atributo en especial que no tengan el resto empleados y que queramos destacar, para ello creamos subentidades. Llegamos a las siguientes definiciones y ejemplos.

Definición 12 Se define una relación reflexiva como aquella de una entidad consigo misma.

Definición 13 Se define una subentidad como una entidad que hereda atributos de otra ya existente y añade algún otro atributo o característica notable.

Ejemplo 10

Capítulo 3

Bases de Datos Relacionales.

El objetivo de este capítulo es introducirnos a las bases de datos relacionales y aprender un poco sobre álgebra relacional y las reglas de Codd.

3.1. Conceptos y definiciones.

Buscamos transformar el diseño lógico del modelo relacional del tema anterior en tablas que nos sean fácil de utilizar para el manejo de datos.

Definición 14 Se define una tabla o relación como una estructura formada por un esquema o conjunto de pares (dominio) y un cuerpo (imagen del dominio) formado por tuplas de pares de la forma (k, v).

Departamento	Nombre	Lugar
10	Mantenimiento	Málaga
20	Contabilidad	Madrid
30	I+D	Málaga
40	Marketing	Barcelona

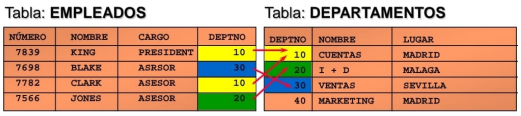
Esquema
Cuerpo

Definición 15 Una base de datos relacional es un conjunto de relaciones o tablas.

A continuación se definen conceptos triviales y evidentes sobre tablas como fila o tupla, columna, dato o valor, valor nulo, clave primaria (al igual que se definía en el capítulo anterior) y por último clave foránea. Esta última es de especial interés.

Definición 16 Se define una clave foránea como un atributo de la entidad que hace referencia a la clave primaria de otra entidad y que sirve como nexo entre tablas para representar las relaciones.

Funcionan del siguiente modo:



3.1.1. Reglas de integridad, redundancia y dependencia funcional.

Proposición 1 *Primera Regla de Integridad.*
Las componentes de una clave primaria no pueden ser nulas.

Proposición 2 *Segunda Regla de Integridad.*
Las componentes de una clave foránea o bien son nulas (si esa fila en concreto no se relaciona) o bien son iguales que el valor de alguna clave primaria en una tabla del modelo.

Definición 17 Se dice que hay redundancia en una base de datos si existe un dato que puede ser deducible a partir del resto de datos. Sigue el ejemplo de la fecha de nacimiento y la edad. Solamente almacenamos la primera y con esta calculamos la edad.

La redundancia provoca un mayor consumo de almacenamiento que en los últimos años no es muy relevante pero también ocasiona posibles incoherencias, es decir, que puedan aparecer dos valores distintos para una misma cosa. Por ejemplo, en caso de que se nos olvide actualizar la edad, si la calculamos con la fecha de nacimiento y comparamos con la almacenada no sería la misma. Por eso se almacena la fecha de nacimiento, es más fácil de mantener.

Definición 18 Se dice que entre dos atributos X e Y de una relación o tabla R existe una dependencia funcional si siempre que hay dos tuplas de R que coinciden en sus valores de X, coinciden también en sus valores de Y.

3.2. Álgebra Relacional.

En esta sección vamos a estudiar algunas operaciones del álgebra relacional que vamos a utilizar en nuestras bases de datos. Distinguimos entre operaciones de conjuntos como la unión, intersección y diferencia de relaciones y operaciones relacionales como la selección, proyección, reunión o join y renombramiento. Vamos a verlas

Unión.
Se trata de unión de tablas como si se tratase de conjuntos. Se denota al igual que en conjuntos. Dadas dos tablas R y S representamos a su unión como $R \cup S$.

Intersección.
Se trata de intersección de tablas como si se tratase de conjuntos. Se denota al igual que en conjuntos. Dadas dos tablas R y S representamos a su intersección como $R \cap S$.

Diferencia.
Se trata de diferencia de tablas como si se tratase de conjuntos. Se denota al igual que en conjuntos. Dadas dos tablas R y S representamos a su diferencia como $R - S$.

Ejemplo 11

Nombre	Departamento
Juan	Mantenimiento
Fran	Contabilidad
Esteban	I+D

$\cup$

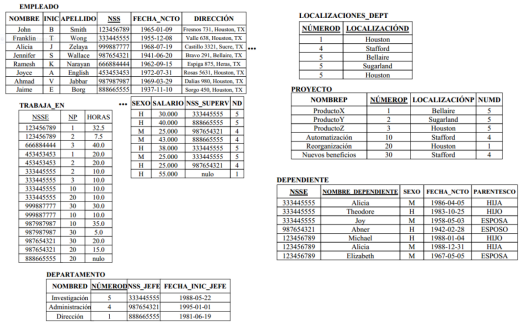
Nombre	Departamento
Juan	Mantenimiento
Fran	Contabilidad
Maite	I+D

$=$

Nombre	Departamento
Juan	Mantenimiento
Fran	Contabilidad
Esteban	I+D
Maite	I+D

Proyeción.
Se trata de la extracción de columnas de una tabla, es decir, se seleccionan y extraen ciertos atributos de una entidad. Se denota por $\pi_{atributo1,atributo1...}(NombreEntidad)$.
Selección.
Se trata de un filtrado de instancias de la entidad según un criterio booleano relacionado con algún atributo. Se denota por $\sigma_{propsAtributos}(NombreEntidad)$.
Reunión o Join.
Se trata de concatenación de tablas como si se tratase de producto cartesiano pero indicando a partir de donde para hacerlo eficiente. Dadas dos tablas R y S representamos a su reunión como $R \bowtie_{atributoR=atributoS} S$.

Ejemplo 12 De la siguiente base de datos deseamos extraer el nombre y apellidos de empleados que trabajan en un proyecto localizado en Houston.



EMPLEADO

NOMBRE	INIC	APELLIDO	NSS	FECHA_NCTO	DIRECCIÓN
John	B	Smith	123456789	1965-01-09	Fresnos 731, Houston, TX
Franklin	T	Wong	333445555	1955-12-08	Valle 638, Houston, TX
Alicia	J	Zelaya	999887777	1968-07-19	Castillo 3321, Sueco, TX
Jennifer	S	Wallace	987654321	1941-06-20	Bravo 291, Bellaire, TX
Ramesh	K	Narayan	666884444	1962-09-15	Espiga 975, Heras, TX
Joyce	A	English	453453453	1972-07-31	Rosas 5631, Houston, TX
Ahmad	V	Jabbar	987987987	1969-03-29	Dallas 980, Houston, TX
Jaime	E	Borg	888665555	1937-11-10	Sturgo 450, Houston, TX

LOCALIZACIONES_DEPT

NÚMEROD	LOCALIZACIÓND
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

TRABAJA_EN

NSSE	NP	HORAS
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	nulo

SEXO	SALARIO	NSS_SUPERV	ND
H	30.000	333445555	5
H	40.000	888665555	5
M	25.000	987654321	4
M	43.000	888665555	4
H	38.000	333445555	5
M	25.000	333445555	5
H	25.000	987654321	4
H	55.000	nulo	1

PROYECTO

NOMBREP	NÚMEROP	LOCALIZACIÓNP	NUMD
ProductoX	1	Bellaire	5
ProductoY	2	Sugarland	5
ProductoZ	3	Houston	5
Automatización	10	Stafford	4
Reorganización	20	Houston	1
Nuevos beneficios	30	Stafford	4

DEPENDIENTE

NSSE	NOMBRE_DEPENDIENTE	SEXO	FECHA_NCTO	PARENTESCO
333445555	Alicia	M	1986-04-05	HIJA
333445555	Theodore	H	1983-10-25	HIJO
333445555	Joy	M	1958-05-03	ESPOSA
987654321	Abner	H	1942-02-28	ESPOSO
123456789	Michael	H	1988-01-04	HIJO
123456789	Alicia	M	1988-12-31	HIJA
123456789	Elizabeth	M	1967-05-05	ESPOSA

DEPARTAMENTO

NOMBRED	NÚMEROD	NSS_JEFE	FECHA_INIC_JEFE
Investigación	5	333445555	1988-05-22
Administración	4	987654321	1995-01-01
Dirección	1	888665555	1981-06-19

Lo hacemos con la siguiente sentencia: $\pi_{nombre,apellido}(\sigma_{localizacion='Houston'}((Empleado \bowtie_{nss=nsse} TrabajaEn) \bowtie_{np=numerop} Proyecto)$.

3.3. Reglas de Codd.

Las reglas de Cood surgen en los años ochenta puesto que muchas bases de datos se decían relacionales pero realmente no lo eran. Así que Codd estableció doce reglas que se debían cumplir para poder se llamada base de datos relacional. Veámoslas.

Regla 1 ***Regla de Información.***
Todos los datos existentes en una base de datos relacional han de ser almacenados en tablas que deben cumplir las premisas del modelo relacional y no se puede acceder a información por otras vías.

Regla 2 ***Regla de Acceso Garantizado.***
Se debe garantizar el acceso a cualquier dato existente si se conoce la clave de una fila (instancia de la entidad) y el nombre del atributo al que se quiere acceder.

Regla 3 ***Regla de los Valores nulos.***
La base de datos debe ser capaz de representar valores desconocidos más allá de escribir en ellos un 0 o una cadena vacía. Los datos nulos deben tener un tipo especial null independientemente del tipo de dato que se maneje en esa columna.

Regla 4 ***Regla del catálogo activo en línea basado en modelo relacional.***
La descripción de la base de datos se almacena de la misma forma que se almacenan los datos ordinarios y deben ser accesibles a los usuarios autorizados bajo los mismo operadores que el resto de tipos de datos.

Regla 5 ***Regla de Sublenguaje Integral o Completo.***
Debe existir al menos un lenguaje con una sintaxis bien definida que pueda ser usado para administrar completamente la base de datos y que tenga sintaxis lineal y pueda ser utilizado dentro de programas de uso independientemente de que el SGBD tenga interfaces más fáciles de usar.

Regla 6 ***Regla de Actualización de vistas.***
Todas las vistas que teóricamente son actualizables deben serlo por el sistema mismo.

Regla 7 ***Regla de Inserción, actualización y borrado.***
El sistema debe permitir la manipulación de alto nivel de los datos, dando posibilidad a todas las operaciones anteriores además de acutalizaciones y borrados de datos, no solamente sobre registros individuales, si no sobre varias tablas al mismo tiempo para mantener la consistencia de datos.

Regla 8 ***Regla de Independencia física de los datos.***
El cambio del diseño físico de la base de datos no afecta a las aplicaciones ni al acceso a tablas.

Regla 9 ***Regla de Independencia Lógica de los datos.***
El cambio del diseño lógico no debe alterar o modificar las aplicaciones.

Regla 10 ***Regla de Independencia de Integridad.***
Todas las restricciones de integridad deben ser definibles en los datos y almacenables en el diccionario de datos, es decir, la integridad forma parte del esquema conceptual de la base de datos, no de las aplicaciones.

Regla 11 ***Regla de Independencia de distribución.***
Dónde se almacene o gestione la base de datos no debe afectar o modificar las aplicaciones. Para esto se debe garantizar la independencia de la localización física, de la framentación o distribución en distintos servidores de la base de datos y de la replicación de la base.

Regla 12 ***Regla de la No Subversión.***
Si se admite una interfaz de acceso a bajo nivel esta no debe permitir saltarse ningún tipo de restricción de integridad o seguridad.

Capítulo 4

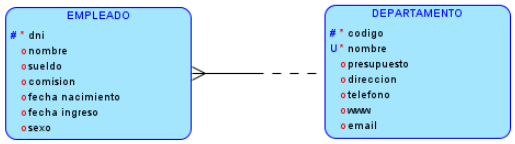
Equivalencia entre el Modelo E/R y el Modelo Relacional.

Durante el segundo capítulo estudiamos el modelo lógico de entidad relación y durante el tercer tema hemos visto el modelo relacional basado en tablas. Ahora, en este tercer capítulo veremos la forma en que se pasa de uno a otro. Veremos para ello una aproximación de lo que hace el programa Data Modeler. También debemos destacar que no hay una única equivalencia entre estos modelos, pueden surgir varias formas de hacerlo pues convertir relaciones y restricciones conlleva una parte creativa.

4.1. Entidades Regulares.

Las entidades regulares son aquellas entidades que no son débiles ni son subentidades. Para cada entidad regular deberemos crear una tabla. Además, deberemos crear las claves y atributos con sus tipos. También podemos especificar restricciones sobre los atributos y deberemos crear índices para los atributos por los que se vayan a realizar búsquedas. Para hacer todo esto tendremos unas sentencias bastante claras. Para la creación de la tabla utilizaremos la sentencia CREATE TABLE. PRIMARY KEY indicará que es clave primaria, NOT NULL nos permitirá indicar que son datos obligatorios y por ello no pueden faltar, no pueden ser nulos. CONSTRAINT permitirá añadir restricciones sobre atributos. Para la creación de índices se empleará CREATE INDEX. Por otro lado, si queremos modificar la tabla para añadir restricciones de unicidad por ejemplo utilizaremos los comandos ALTER TABLE para la modificación y UNIQUE para indicar unicidad. Esto explicado en texto puede resultar un tanto difícil de comprender. No obstante, veamos un ejemplo bastante clarificador.

Ejemplo 13 Vamos a pasar a tablas las siguientes entidades del modelo relacional.



Lo hacemos con el siguiente código:

```
CREATE TABLE EMPLEADO (
    dni NUMBER(9) PRIMARY KEY,
    nombre VARCHAR2(15) NOT NULL,
    sueldo NUMBER(5) NOT NULL,
    comision NUMBER(5),
    fecha_nacimiento DATE,
    fecha_ingreso DATE,
    sexo CHAR(1),
    CONSTRAINT sexo_valido CHECK (sexo IN ('M','H'))
);\\
```

Listing 4.1: Ejemplo SQL 1.

```
CREATE INDEX Emp_Nombre_Idx ON Empleado(nombre);
CREATE TABLE DEPARTAMENTO (
    codigo NUMBER PRIMARY KEY,
    nombre VARCHAR2(25) NOT NULL,
    presupuesto NUMBER(7),
    direccion VARCHAR2(35),
    telefono VARCHAR2(9),
    www VARCHAR2(90)
    correo VARCHAR2(30),
);
ALTER TABLE Departamento ADD CONSTRAINT UK_NOMBRE UNIQUE (nombre);
```

Listing 4.2: Ejemplo SQL 2.

4.2. Relaciones.

Vamos a comenzar a transformar las relaciones. Vamos a ir poco a poco. Comenzaremos por relaciones 1:M (uno a muchos), después con relaciones M:M (muchos a muchos), entidades débiles, subentidades y por último relaciones 1:1.

Relación 1:M.

Planteamos cómo realizar una relación 1 a muchos entre dos entidades. Queremos representar una relación en las tablas, para ello necesitamos algo que nos permita pasar de una tabla a otra por así decirlo. Esto nos lo permite la operación de reunión vista en el capítulo anterior. Así que simplemente deberemos añadir un atributo en una de las tablas que haga

de nexo, es decir, crearemos un atributo nuevo que haga de referencia a la clave de la otra entidad. A esto lo llamaremos clave foránea.

Ahora nos planteamos la pregunta, ¿a cuál de las dos tablas le añadimos la clave foránea? La respuesta es sencilla. Estamos ante una relación uno a muchos, es decir, la entidad A tiene muchas instancias de B con las que se relaciona mientras que una instancia de B se relaciona con una sola instancia de A, así que es más sencillo añadirla en la entidad B. Además, si la relación es obligatoria, el atributo creado deberá ser NOT NULL.

Así si continuamos el *Ejemplo 13* deberíamos añadir las siguientes líneas:

```
1 ALTER TABLE Empleado ADD (departamento\_codigo NUMBER NOT NULL);
  ALTER TABLE Empleado ADD CONSTRAINT trabaja\_en FOREING KEY (departamento\
      _codigo) REFERENCES Departamento (codigo);
```

Listing 4.3: Ejemplo SQL 3.

Relación M:M.
Las relaciones muchos a muchos entre entidades son similares a las relaciones uno a muchos, pero aquí tenemos un problema añadido. Cuando vamos a preguntarnos dónde añadir la clave foránea, si en la entidad A o en la entidad B, ambas tienen el problema de que pueden relacionarse con muchos y necesitar muchas claves, no solo una. Así que no podemos simplemente añadir un atributo a una de las entidades existentes, en este caso, creamos una nueva tabla que representa a la relación e incluye la clave foránea asociada a A y la clave foránea asociada a B. Así si en el *Ejemplo 13* cambiamos la relación de trabajar en un departamento por la relación, haber sido entrevistado por un departamento, pasamos a tener una relación muchos a muchos que se modelaría con las siguiente líneas:

```
  CREATE TABLE Entrevista (
2 ----departamento_codigo NUMBER NOT NULL REFERENCES Departamento (codigo),
  ----empleado_dni VARCHAR(10) NOT NULL REFERENCES Empleado (dni),
4 ----PRIMARY KEY (departamento_codigo, empleado\_dni)
  );
6 CREATE INDEX Entrevista_emp_idx ON Entrevista (empleado\_dni);
```

Listing 4.4: Ejemplo SQL 4.

Entidad Débil.
Las entidades débiles de otras entidades regulares se implementan simplemente incluyendo referencias no nulas a las entidades regulares en la entidad débil. Así se incluyen las claves foráneas en la entidad débil. Tras esto, se asocia como clave primaria de la entidad débil la composición formada por las claves foráneas y la clave propia de la entidad débil en caso de existir esta. Así, simplemente habría que crear la tabla para la entidad débil, tras crearla, se asocia su clave primaria, después se indican cuáles son claves foráneas y por último se crea un índice para cada clave foránea. Vemos un ejemplo.

Empleado

DNI	Nombre	Departamento Codigo	Superior DNI
33387199W	Eduardo Gris	3000	77623977J
24887120L	Pedro Rojas	2800	
77623977J	Ana Negrín	3500	

Área

Departamento Codigo	Codigo Area	Funcion
ING	PRG	Programación
ING	PRL	Personal C.
FNZ	HBL	Habilitación
ING	HDW	Hardware
FNZ	PRL	Personal

Tarea

empleado dni	area departamento codigo	area codigo area	fecha	estado
33387199W	ING	PRG	3/10/17	S
33387199W	ING	PRL	11/12/17	S
24887120L	FNZ	HBL	3/1/18	N
77623977J	FNZ	PRL	8/1/18	S

Subentidad.
Las subentidades no son más que entidades que heredan de otra. Por ello lo único que debemos hacer es crear una tabla para la subentidad, añadirle los atributos propios de esta y después añadir una referencia a la clave primaria de la super-entidad de la que hereda. Destacamos que estas comparten clave primaria, a diferencia de la entidad débil, que la utilizaba, pero no era exactamente la misma.
A la hora de modelar una subentidad jefe tendríamos las siguientes sentencias:

```
CREATE TABLE jefe (
    empleado_dni NUMBER NOT NULL
    coche VARCHAR2(16) NOT NULL
    despacho VARCHAR2(10) NOT NULL
);
ALTER TABLE jefe ADD CONSTRAINT JEFE_PK PRIMARY KEY (empleado_id);
ALTER TABLE jefe ADD CONSTRAINT es_jefe FOREING KEY (empleado_id) REFERENCES
    EMPLEADO (ID);
```

Listing 4.5: Ejemplo SQL 5.

Relación 1:1
Estas relaciones se modelan según la obligatoriedad. Dependiendo de en qué lado es obligatoria, se añade en esta entidad la clave foránea. Además, este atributo deberá añadir la

restricción de ser único para impedir que varias filas de A referencien a la misma de B. En caso de que B tenga clave compuesta se añadirán tantos atributos como tenga la clave de B. En caso de ser opcional por ambos lados la relación se implementa en la entidad que menos instancias vaya a generar para reducir memoria.

4.3. Optimización.

4.3.1. Buenas prácticas.

En general para que todo funcione correctamente debemos evitar usar claves compuestas, siempre que podamos usar una simple. Los índices son básicos, pues sin ellos no podemos acceder a los datos almacenados en la tabla, para ello necesitamos que las claves primarias sean identificadores internos y así simplemente deberíamos usar secuencias para rellenar los datos de las claves. Vemos cómo se construyen estas secuencias.

4.3.2. Secuencias.

Definición 19 Se define una secuencia como una instancia de una entidad pero vista en el modelo relacional en tablas, es decir, una fila de una tabla.

Para crear una secuencia se utiliza la sentencia CREATE SEQUENCE. Ahora para insertar datos en esta secuencia se utiliza la sentencia INSERT. Vemos un ejemplo:

```
1 CREATE SEQUENCE Empleado_id_SEQ;
  INSERT INTO EMPLEADO VALUES (
3 Empleado_id_SEQ.next_val,
  '33789456W',
5 'Juan Pedro', ...)
```

Listing 4.6: Ejemplo SQL 6.

Capítulo 5

Introducción a la Normalización.

Ya hemos visto cómo se construye el modelo relacional, pero nos surge una pregunta importante de resolver, ¿lo hemos hecho bien? para ello existe un modelo relacional normalizado que nos ayudará a saber si nuestro modelo es correcto.

5.1. Introducción. Redundancias y anomalías.

Crear tablas donde aparezcan atributos referentes a otras entidades no es adecuado pues provoca redundancia en el almacenamiento y anomalías de inserción modificación o borrado. Pongamos un ejemplo práctico donde ver a qué nos referimos con anomalías.

Ejemplo 14

Numero Empleado	Numero Proyecto	Empleado Nombre	Proyecto Nombre	Numero Horas
23	15	Juan Pérez	Aeroespacial	1200
23	30	Juan Pérez	Mantenimiento	500
27	15	María López	Aeroespacial	750
27	32	María López	Diseño	834

Cuadro 5.1: Tabla cuya clave primaria está formada por el número de empleado y número de proyecto.

- Al cambiar el nombre del proyecto 115 de Aeroespacial a Mantenimiento habría que cambiar el nombre en todas las filas con número de proyecto 115 a Mantenimiento y hacerlo uno a uno, esto lo consideramos una anomalía de modificación.

- Si ahora quisiéramos insertar un nuevo proyecto, deberíamos insertar un nuevo empleado asociado a él. Esto es otra anomalía de de inserción. También ocurre al revés, para insertar un empleado habría que asignarle un proyecto.
- Al eliminar un proyecto se eliminan también los empleados que estén asociados solo a ese proyecto, esto es una anomalía de eliminación.

Así llegamos una semántica para la relación de atributos que establece lo siguiente: cada tupla en una relación o tabla debe representar una instancia de la relación (una fila es una instancia de la tabla). Esto es que los atributos de diferentes entidades no deberían mezclarse en una misma tabla a no ser que se empleen claves foráneas para referenciarlas haciendo que una entidad y sus atributos sean tan independientes como sea posible.
El objetivo de la normalización es proporcionar un procedimiento sistemático para eliminar las posibles redundancias y anomalías.

5.2. Normalización.

Definición 20 Se define la normalización como el proceso de descomponer tablas o relaciones mal construidas generalmente dividiéndolas en otras relaciones más pequeñas.

5.2.1. Definiciones previas.

Definición 21 Sea una relación R y sean $A_1, ..., A_n, B$ atributos suyos. Se dice que existe una dependencia funcional entre $A_1, ..., A_n$ y B y se denota por $A_1, ..., A_n \rightarrow B$ si para todo par de tuplas de R tales que coinciden los valores de los atributos $A_1, ..., A_n$ se tiene que también coinciden los valores del atributo B.
De forma más general si la dependencia no es con solamente un atributo B sino que es generada con k atributos $B_1, ..., B_k$ se denotará por $A_1, ..., A_n \rightarrow B_1, ..., B_k$

Definición 22 Sea una relación R y sea $A_1, ..., A_n \rightarrow B_1, ..., B_k$ una dependencia funcional. A $A_1, ..., A_n$ se les llama elementos determinantes y a $B_1, ..., B_k$ se les llama elementos determinados.

Definición 23 Sea R una relación y sean $A_1, ..., A_n$ todos sus atributos. Se dice que un subconjunto $K \subseteq A_1, ..., A_n$ es una clave si se verifican:

1. Se puede identificar una única fila por el valor de la clave.
2. No hay dos tuplas que coincidan en la clave.
3. La clave determina funcionalmente a todos los atributos, es decir, $K \rightarrow A_1, ..., A_n$.

Definición 24 Sea R una relación y sea K una clave. Se dice que K es una clave minimal si $\nexists K' \subset K$ tal que K' sea clave. Dicho en palabras, no existe ninguna clave más pequeña que K.

Definición 25 Sea R relación y sea K una clave. Se dice que K es una superclave si existe K' una clave minimal contenida en K.

Nota: La clave primaria no es más que una clave seleccionada de entre las anteriores y por lo general se suele emplear la minimal más sencilla.

Ejemplo 15

Dada la tabla siguiente se tiene que:

Indicativo	Número	País	Abonado
34	952131000	España	UMA
34	951299316	España	Turismo Andaluz
34	951299300	España	Turismo Andaluz
1	8002752273	EEUU	Apple
1	8004269400	EEUU	Microsoft
1	951299300	Canadá	Peter Smith
1	8005550077	EEUU	Turismo Andaluz

Cuadro 5.2: Tabla Ejemplo para dependencias funcionales

1. $Pais, Numero \rightarrow Abonado$ puesto que para cada par de tuplas de país y número de R con valores coincidentes, también coinciden los valores de la tupla formada formada por abonado. En este caso en concreto como no hay ninguna tupla de país y número de R con valores coincidentes, se verifica la definición.

2. $Pais \rightarrow Indicativo$ puesto que para todo par de filas, siempre que el valor de la columna $Pais$ coincide, también coincide el valor de la columna $Indicativo$.

3. NO se tiene la dependencia funcional $Indicativo \rightarrow Pais$ puesto que para el indicativo con valor 1, pueden tenerse los países Canadá o EEUU.

Definición 26 Se define un atributo primario como aquel atributo miembro de alguna clave, ya sea candidata o primaria.

Definición 27 Sea R una relación y sea $A \rightarrow B$ una dependencia funcional sobre R. Se dice que es una dependencia funcional plena si $\nexists C \rightarrow B$ donde $C \subset A$

Proposición 3 Si existe un único elemento determinante, entonces la dependencia funcional es plena.

Definición 28 Sea R una relación y sea $A \to B$ una dependencia funcional sobre R. Se dice que es una dependencia funcional transitiva si $\exists C$ con $C \nsubseteq A$ y $B \nsubseteq C$ verificando que $A \to C$ y $C \to B$.

5.2.2. Formas normales.

Definición 29 Se define una forma normal como una condición que una relación cumple de acuerdo a sus claves y dependencias.

Existen varias formas normales. Desde la primera a la tercera junto con la forma normal de Boyce-Codd (FNBC) se basan en sus claves y dependencias funcionales. La cuarta se basa en las claves y dependencias multivaluadas (MVD). La quinta forma normal se basa en las claves y dependencias de unión (JD). Para obtener un diseño de la base de datos correcto nos basta con la FNBC (no estudiamos la cuarta y quinta forma normal).

Definición 30 Primera forma normal. 1ª FN. Se dice que una relación está en primera forma normal si no permite atributos multivalor, es decir, está en primera forma normal si todos sus atributos son atómicos.
Nota: toda tabla que cumpla las restricciones de DDL estará en 1ª forma normal.

Definición 31 Segunda forma normal. 2ª FN. Se dice que una relación está en segunda forma normal si todo atributo no primario se encuentra en una dependencia funcional plena con la clave primaria. En general una relación se encuentra en 2ªFN si cualquier atributo no primario se encuentra en dependencia funcional plena con todas las claves.
Nota: si una relación no está en 2ªFN puede descomponerse en varias relaciones en 2ªFN

Ejemplo 16

INDICATIVO	NUMERO	PAIS	ABONADO
34	952131000	España	UMA
34	951299316	España	Turismo Andaluz
	951299300		
1	800-275-2273	EEUU	Apple
1	800-426-9400	EEUU	Microsoft
1	951299300	Canadá	Peter Smith
1	800-555-0077	EEUU	Turismo Andaluz

1ª forma normal =>'

INDICATIVO	NUMERO	PAIS	ABONADO
34	952131000	España	UMA
34	951299316	España	Turismo Andaluz
34	951299300	España	Turismo Andaluz
1	800-275-2273	EEUU	Apple
1	800-426-9400	EEUU	Microsoft
1	951299300	Canadá	Peter Smith
1	800-555-0077	EEUU	Turismo Andaluz

2ª forma normal =>'

NUMERO	PAIS	ABONADO
952131000	España	UMA
951299316	España	Turismo Andaluz
951299300	España	Turismo Andaluz
800-275-2273	EEUU	Apple
800-426-9400	EEUU	Microsoft
951299300	Canadá	Peter Smith
800-555-0077	EEUU	Turismo Andaluz

INDICATIVO	PAIS
34	España
1	EEUU
1	Canadá

Cuadro 5.3: Obtención de la 1ª y 2ª forma normal

Definición 32 Tercera Forma Normal. 3ªFN. Se dice que una relación se encuentra en 3ªFN si está en 2ªFN y ningún atributo no primario de la relación es transitivamente dependiente de la clave primaria a través de atributos no primarios, es decir, en caso de haber una dependencia transitiva es a través de una clave.
En general, una relación se encuentra en 3ªFN si para toda dependencia funcional transitiva $X \to A$ se verifica que X es una superclave de R y A es un atributo primario
Nota: si una relación no está en 3ªFN debe poder descomponerse en varias relaciones que incluyan los atributos no primarios que determinen funcionalmente otros atributos no primarios.

Definición 33 Se dice que un esquema se encuentra en FNBC si cuando existe una dependencia funcional $X \to A$ entonces se cumple que X es una superclave de R.

Proposición 4 Caracterización de FNBC. Una relación está en FNBC si y solo sí está en 3ªFN y cada determinante es una clave candidata.

Proposición 5 $FNBC \Rightarrow 3FN \Rightarrow 2FN \Rightarrow 1FN$

Ejemplo 17

Codigo Postal	Dirección	Ciudad
3000	C/Flores, 17	Merida
4858	Av Bolivar, 72	Maracay
3000	C/Flores, 18	Merida

Cuadro 5.4: Tabla ejemplo para FNBC. La clave está formada por Ciudad y Dirección

Es claro que está en 1ªFN. Comprobamos que está en 2ªFN. El único atributo no primario es código postal. Además se encuentra en dependencia funcional plena con la clave puesto que $Ciudad, Direccion \to CodigoPostal$ pero $Ciudad \nrightarrow CodigoPostal$ y $Direccion \nrightarrow CodigoPostal$, por tanto está en segunda forma normal. Pasamos a comprobar que está en 3ªFN. Las únicas dependencias funcionales son $Ciudad, Direccion \to CodigoPostal$ y $CodigoPostal \to Ciudad$. La única dependencia transitiva que puede haber es $Ciudad, Direccion \to Ciudad$ donde $Ciudad, Direccion$ es una superclave y $Ciudad$ es un atributo primario, por tanto está en tercera forma normal. Por último vemos que no está en FNBC puesto que no todo determinante es clave candidata ya que $CodigoPostal$ no lo es. Para solucionar esto se divide la tabla. Como código postal es quien hace que no se cumpla, se separa la tabla en dos formadas por código postal y ciudad y código postal y dirección.

Capítulo 6

El lenguaje de Consultas estructurado SQL.

6.1. Introducción al SQL.

El lenguaje de consulta estructurado SQL, por sus siglas *Structure Query Language* es un lenguaje de control e interacción con un sistema gestor de bases de datos. Nos ofrece los lenguajes de definición de datos DDL y de manipulación de datos DML para un uso del álgebra relacional sencillo que nos permite efectuar consultas con el fin de recuperar información de la bases datos y realizar cambios en ella.

6.2. Consultas Básicas.

6.2.1. Sentencia select, proyección, ordenación y operadores de conjuntos.

La sentencia SELECT sirve para mostrar información sobre el cuerpo de una tabla, es decir, muestra las filas que posee. Además DESC nos proporciona un esquema de la tabla que le digamos, nos mostrará sus atributos, los valores que pueden tomar y si son nulos o no.

Por otro lado las proyecciones se harán con la sentencia SELECT y FROM que nos indicará de qué tabla queremos seleccionar qué atributos. Si hay alguna tupla repetida y queremos no mostrarla más de una vez, se hará con la cláusula DISTINCT. Simplemente seguirá la estructura siguiente:

```
1 SELECT DISTINCT atributo1, atributo2, ..., atributoN
  FROM tabla;
```

Listing 6.1: Ejemplo SQL 1.

Por otro lado, si queremos mostrar la tabla ordenada por algún atributo en específico podemos utilizar la cláusula ORDER BY. También podemos ordenarlo en orden ascendente o descendente. Se trata como en el siguiente ejemplo.

```
SELECT atributo1 , atributo2 , ... , atributoN
FROM tabla
ORDER BY atributoI ASC, atributoJ DESC;
```

Listing 6.2: Ejemplo SQL 2.

Las operaciones de conjuntos son las ya conocidas de unión, intersección, diferencia y producto cartesiano, aunque de este último no hablaremos salvo para condiciones de reunión más tarde. Las tres primeras se denotan por: UNION, INTERSECT Y MINUS. Además la unión elimina resultados repetidos, si queremos que no los elimine añadimos la opción UNION ALL. Estas operaciones se utilizan entre tablas de forma similar al siguiente ejemplo simple:

```
SELECT * FROM tabla1
UNION
SELECT * FROM tabla2 ;
```

Listing 6.3: Ejemplo SQL 3.

6.2.2. Selección y operadores.

Para seleccionar además únicamente atributos que cumplan algunas propiedades o condiciones se utiliza la cláusula WHERE. En ella se utilizan además varios operadores para indicar las condiciones. Vamos a ver algunos de estos operadores.

- Operadores relacionales: =, !=, <, >, <=, >= ...
- Operadores aritméticos: +, -, *, /, MOD, POWER...
- Operadores trigonométricos: SIN, COS, TAN, ASIN, ACOS, ATAN...
- Operadores booleanos: AND, OR, NOT.
- Between: se emplea para indicar que un atributo está entre ciertos valores. Ej: codigo BETWEEN 2 AND 4.
- Like: se indica para los char principalmente, se indican patrones, expresiones regulares para identificar cómo es un atributo. Por ejemplo LIKE 'M %ímplica que empieza por M y después puede haber cualquier otra cosa.
- CONCAT sirve para concatenar dos cadenas a y b
- SUBSTRN(cad, pos, len) devuelve el substring de la cadena cad desde la posición pos con una longitud de len.

- LOWER(cad) devuelve una cadena puesta todo en minúsculas.
- UPPER(cad) devuelve una cadena puesta todo en mayúsculas.
- ROUND(n, dec) redondea el número n a una cantidad de dec decimales.
- TRUNC(n, dec) trunca el número n a una cantidad de dec decimales.
- ADD_MONTHS(fecha, m) devuelve la fecha fecha tras sumarle m meses.
- MONTHS_BETWEEN(fecha1, fecha2) te devuelve la contidad de meses entre dos fechas.
- fecha1 - fecha2 devuelve la diferencia de días entre dos fecha.
- NEXT_DAY(fechaInicio, diaSemana) devuelve la fecha posterioa a fecha inicio correspondiente al día de la semana diaSemana.
- SYSDATE devuelve la fecha actual del sistema.
- TO_NUMBER(cad) transforma una cadena de caracteres en un formato numérico
- TO_CHAR(param) convierte el parámetro param en caracteres.
- TO_DATE(cad) convierte una cadena de caracteres en fecha.
- DECODE(atr, vali, nvali, vald) para el atributo atr, sustituyo los valores vali por nuevos valores dados nvali, y si no se da ninguno en concreto se dará el valor por defecto vald.
- ROWNUM es un atributo extra para cada tabla en que se indica el número de filas de la tabla.
- ROWID es identificador único para cada fila.

Además de en la cláusula WHERE, estos operadores pueden ser utilizados en la cláusula SELECT, por ejemplo para seleccionar una siguiente fecha a algún dato... En esos casos podemos asignar un nombre con el uso de comillas dobles. Vemos un ejemplo de estos:

```
1 SELECT nombre, months_between(sysdate, fecha_nacimiento)/12 "edad"
  FROM profesores
3 WHERE UPPER(apellido1) LIKE 'Z%' AND ROWNUM < 4
```

Listing 6.4: Ejemplo SQL 4.

6.2.3. Manejo de valores nulos.

En SQL se maneja lógica trivaluada, esto quiere decir que además de TRUE y FALSE se utila el valor NULL. Así cualquier función evaluado con un valor NULL devolverá también NULL. Se emplean funciones como:

- IS NULL: devuelve true si un valor es null o false si no es null.
- IS NOT NULL: devuelve true si un valor no es null o false si es null.
- NVL(e1, e2): devuelve la expresión 1 si la expresión 1 no es nula ó la expresión 2 en caso de ser nula.

6.3. Reunión.

La operación de reunión se emplea con un el término JOIN que sirve para enlazar tablas. Si queremos así enlazar n tablas necesitaremos al menos n-1 condiciones de reunión. El caso genérico será el siguiente:

```
SELECT t1.nombre, t2.nombre
FROM tabla1 t1, tabla2 t2
WHERE atribt1 = atribt2 --condicion de reunion
```

Listing 6.5: Ejemplo SQL 5.

Vamos a ver tipos específicos de reunión.

6.3.1. Self Join.

Se trata de la reunión de una tabla consigo misma. En este caso es necesario saber distinguir qué atributo es de una tabla y cuál es de otra. Sin embargo, se llaman igual ambas tablas, por ello utilizamos alias para tablas como se ha visto en el ejemplo anterior.

No obstante, cuando hacemos un Self Join nos encontramos con que a se relaciona con a y que además si a se relaciona con b, entonces b se relaciona con a. Tenemos entonces datos duplicados en la tabla y que no nos suele interesar, por ello, debemos añadir una condición de que las claves identificadoras de las tablas sean una menor estricto que la otra.

```
SELECT n1.nombre, n2.nombre
FROM profesores n1, profesores n2
WHERE n1.id < n2.id
```

Listing 6.6: Ejemplo SQL 6.

En ocasiones se darán JOINS y SELF JOIN a la vez en una misma consulta, por ello también se deberá usar la cláusula DISTINCT.

6.3.2. Natural Join.

Un NATURAL JOIN se realiza entre dos tablas. Este consiste en unirlas mediante todos los atributos de las tablas que tienen el mismo nombre. Por ejemplo, una consulta en que se seleccionen atributos de una tabla tab1 t y tab2 p con condiciones de reunión t.atributo1=p.atributo1 ... t.atributon=p.atributon será una consulta de natural join, no obstante, para ahorrar escribir todo esto puede escribirse como sigue:

```
SELECT atrib1, ..., atribn
FROM tab1 t NATURAL JOIN tab2 p
WHERE ..... otras condiciones que no sean de reunion.
```

Listing 6.7: Ejemplo SQL 7.

6.3.3. Join Using.

Si nos fijamos en la definición dada de forma no formal sobre natural join, podemos ver que utiliza todos los atributos comunes que tienen las tablas... pero en ocasiones no querremos usarlos todos, solamente algunos. Así que no nos sirve el natural join, pero por pereza a la hora de escribir, en vez de utilizar las condiciones de reunión en el WHERE, se inventó el JOIN USING que se utiliza para hacer la unión de tablas usando atributos con el mismo nombre, pero sin necesidad de que sean todos. Vemos su estructura.

```
SELECT atrib1, ..., atribn
FROM tab1 t JOIN tab2 p USING (lista de atributos seleccionados)
WHERE ..... otras condiciones que no sean de reunion.
```

Listing 6.8: Ejemplo SQL 8.

6.3.4. Conditional Join.

Ahora si quisieramos hacer un join en que dos atributos no se llamen iguales pero si se identifiquen, no tenemos ninguna sentencia especial y deberíamos hacerlo como se vio en un comienzo, no obstante, a los informáticos esto no les terminaba de gustar y crearon la sentencia JOIN ... ON (...) que básicamente te facilita escribir las condiciones de reunión en el from junto a las tablas implicadas. Vemos cómo se hace con un ejemplo.

```
SELECT atrib1, ..., atribn
FROM tab1 t JOIN tab2 p ON (condicion_reunion)
WHERE ..... otras condiciones que no sean de reunion.
```

Listing 6.9: Ejemplo SQL 9.

6.3.5. Outer Join.

Cuando ya parece que tenemos cumplidas todas las necesidades de las reuniones nos damos cuenta de que nos falta el caso en que queramos mostrar todas las filas de una una tabla aunque algunas de ellas no cumplan la condición de reunión. Para ello está el OUTER JOIN, que puede ser además por izquierda (LEFT OUTER JOIN) si queremos que sean los de los la primera tabla solamente que aparezcan, por derecha (RIGHT OUTER JOIN) si queremos que sean los de la derecha, o por ambos lados (OUTER JOIN). Vemos un ejemplo concreto.

Ejemplo 18 Buscamos hacer un listado de todas las asignaturas que muestre el nombre de la asignatura y además el nombre de la materia asociada. Nos damos cuenta de que dice todas las asignaturas y el nombre de la materia asociada, pero puede ser que no tenga materia asociada... Esto es un ejemplo claro de LEFT OUTER JOIN.

```
SELECT a.nombre "asignatura" m.nombre "materia"
FROM asignaturas a LEFT OUTER JOIN materias m
    ON (a.cod_materia = m.codigo);
```

Listing 6.10: Ejemplo SQL 10.

6.4. Subconsultas.

Definición 34 Se dice que una consulta es una subconsulta si aparece dentro de otra consulta, es decir, sentencia SELECT dentro de otra sentencia SELECT.

En estos casos suele aparecer encerrada entre paréntesis dentro de las cláusulas WHERE o HAVING de la consulta principal, por ello es imprescindible que las tablas lleven alias. También observamos para cada fila de la consulta principal se ejecutará la subconsulta, por tanto son lentas de ejecutar.

Además las subconsultas nos llevan a introducir nuevos operadores. Pongamos el caso de querer listar todas las asignaturas con más créditos que las asignaturas de segundo. Para ello primero tomamos las asignaturas de segundo y buscamos que el número de créditos de las asignaturas sean mayores que todas ellas, esto nos lleva a definir un operador ALL. Si solamente quisiéramos que fuese mayor que alguna de las utilizaríamos el operador ANY. Vemos un ejemplo del segundo caso.

```
SELECT nombre
FROM asignaturas
WHERE creditos >
    ANY (SELECT creditos FROM asignaturas WHERE curso = 2);
```

Listing 6.11: Ejemplo SQL 11.

Además de estos tendremos los operadores IN Y EXISTS. El primero de ello ya lo utilizamos cuando hacíamos restricciones en el modelo relacional. El segundo de ellos es parecido, será cierto en caso de que exista alguna tupla en una subconsulta. Además tiene gran utilidad para hacer condiciones de reunión de forma diferente. Vemos un ejemplo de cómo simular una condición de reunión con ambas expresiones.

Ejemplo 19 En este ejemplo vamos a ver 3 formas de hacer la misma consulta, con reunión, con IN y con EXISTS.

```
SELECT i.asignatura "codigo asignatura"
FROM profesores p, impartir i
WHERE i.profesor = p.id
    AND UPPER(p.nombre) = 'MANUEL'
    AND UPPER(p.apellido1) = 'ENCISO';
```

Listing 6.12: Ejemplo SQL 12 con reunión.

```
SELECT asignatura "codigo asignatura"
FROM impartir
WHERE profesor IN (
    SELECT id
    FROM profesores
    WHERE  UPPER(p.nombre) = 'MANUEL'
            AND UPPER(p.apellido1) = 'ENCISO';
    )
```

Listing 6.13: Ejemplo SQL 12 con IN.

```
SELECT i.asignatura "codigo asignatura"
FROM impartir i
WHERE EXISTS (
    SELECT *
    FROM profesores
    WHERE   i.profesor = p.id
            AND UPPER(p.nombre) = 'MANUEL'
            AND UPPER(p.apellido1) = 'ENCISO';
    )
```

Listing 6.14: Ejemplo SQL 12 con EXISTS.

6.4.1. Consultas negativas.

Estas se realizan con subsconsultas. Siguen siempre la misma estructura y las identificaremos por llevar la palabra “No” en el enunciado de la consulta. Se realiza del siguiente modo: seleccionamos las que sí cumplan la condición primero como subconsulta y luego establecemos otra consulta que en la cláusula WHERE incluya un not in o not exists de la subconsulta. Vemos un ejemplo.

Ejemplo 20 Vamos a mostrar los datos de los profesores que imparten las asignaturas que no tengan código 130.

```
SELECT * FROM profesores
WHERE id NOT IN (SELECT i.profesor FROM impartir i
                 WHERE i.profesor I p.id
                 and i.asignatura = 112)
    )
```

Listing 6.15: Ejemplo SQL 13.

6.5. Funciones de agregación.

Definición 35 Se define una función de agregación como aquella que opera sobre un conjunto de tuplas, es decir, sobre una tabla y devuelve un único valor que será la imagen por dicha función.

Estas son útiles para calcular información que está almacenada de forma implícita en la base de datos. Algunas de ellas son:

- SUM (expr): suma los valores que haya en la tabla expr.
- MIN (expr): devuelve el valor mínimo que haya en la tabla expr.
- MAX (expr): devuelve el valor máximo que haya en la tabla expr.
- COUNT(expr): cuenta el número de tuplas que hay en una tabla.
- COUNT(*): cuenta el número de tuplas que hay en una tabla sin que ignore valores nulos.
- AVG(expr): calcula la media aritmética de los valores.

Es importante destacar la opción de añadir DISTINCT a cada una de las funciones anteriores para que así considere únicamente valores distintos.

6.5.1. Group By y Having.

La función GROUP BY nos permite dividir las tuplas seleccionadas del resultado por la cláusula WHERE en conjuntos de tuplas disjuntos sobre cada uno de los cuales se va a aplicar una función de agregación, es decir, se crea una partición de las tuplas según algún criterio y después se le aplican una serie de funciones de agregación a dichas particiones. Notamos que en la cláusula SELECT solo podremos incluir información sobre las entidades del group by y funciones de agregación sobre ellos. Vemos un ejemplo:

```
SELECT trunc(months_between(sysdate,fecha_nacimiento)/12/10) ''Decenio"
--------MAX(fecha_macimiento) ''Fecha"
FROM profesores
GROUP BY trunc(months_between(systade,fecha_nacimiento)/12/10)
ORDER BY ''Decenio"
```

Listing 6.16: Ejemplo SQL 14.

Observamos que no podemos indicar "Decenio"directamente en la cláusula group by pero si en order by. Esto se debe al orden de ejecución que sigue sql. Por otro lado la función HAVING lo que hace es permitir filtrado dentro de cada grupo, igual que hacíamos en el where e imponíamos restricciones, para grupos, se imponen las restricciones en la cláusula HAVING. De este modo, el código en general de una consulta que utilice group by y having nos quedaría tal que:

```
SELECT fagregacion1, fagregacion2, ...
FROM tablas
WHERE filtros_filas
GROUP BY exp1, exp2, exp3, ...
HAVING filtros_grupos
ORDER BY crit_orden
```

Listing 6.17: Ejemplo SQL 15.

Otro resultado destacable es que las funciones de agregación solamente pueden ser utilizadas en las cláusulas SELECT, HAVING y ORDER BY. Esto es de nuevo debido al orden de ejecución dentro de una sentencia SELECT. Es el siguiente:

1. FROM tablas
2. WHERE filtro_filas
3. GROUP BY expr1...
4. HAVING filtro_grupos
5. SELECT seleccion
6. ORDER BY criterios

Notamos que para identificar este tipo de consultas vamos a tener habitualmente en el enunciado de la consulta una expresión del tipo para cada .... (group by) pero... (having). Vamos a ver un ejemplo.

Ejemplo 21 Listar el código, nota media y número de alumnos matriculados en cada una de las asignaturas, pero sólo de las asignaturas que tiene un grupo B y más de 10 alumnos. El resultado ha de estar ordenado por número de alumnos matriculados.

```
SELECT asignatura, COUNT(*) ''MATRICULADOS",
    ROUND(AVG(DECODE(calificacion, 'MH', 10, 'SB' 9, 'NT', 7, 'AP' 5, 'SP' 0,
        0)),2)
    ''MEDIA"
FROM matricular
WHERE grupo = 'B'
GROUP BY asignatura
HAVING COUNT(*)>10
ORDER BY COUNT(*);
```

Listing 6.18: Ejemplo SQL 16.

6.5.2. Anidamiento avanzado.

El anidamiento avanzado consiste en realizar subconsultas solo que en vez de emplearlas en la cláusula WHERE lo haremos directamente en el FROM. Esto tiene ventajas como que estructuramos mejor la consulta y podemos hacerlo adaptando así el orden de ejecución, por ejemplo, si nos piden las dos asignaturas con más créditos en total. En un primer momento se nos ocurriría emplear la siguiente consulta.

```
SELECT *
FROM asignaturas
WHERE rownum <= 2 AND creditos IS NOT NULL
ORDER BY creditos DESC
```

Listing 6.19: Ejemplo erróneo SQL 17.

No obstante, hemos visto que en el orden de ejecución lo último que se hace es el order by... Así que este código lo que hace es tomar la tabla asignaturas, se queda con dos cualesquiera cuyos créditos son no nulas (no necesariamente las de mayor cantidad de créditos) y después ordena... En consecuencia, no devuelve lo que queríamos... para que primero se realice la ordenación no nos queda otra que incluir un anidamiento avanzada de la siguiente forma:

```
SELECT *
FROM (SELECT * FROM ASIGNATURAS
    WHERE creditos IS NOT NULL
    ORDER BY creditos DESC)
WHERE rownum <= 2;
```

Listing 6.20: Ejemplo corregido SQL 18.

Añadimos dos observaciones. Este tipo de consultas pueden resultar fallidas más fácilmente pues en caso de equivocarnos se hará directamente en el origen de la consulta por lo que desde un principio habrá datos erróneos. Por otro lado, si queremos dar un alias a algún elemento seleccionado en la subconsulta contenida en la cláusula FROM se deberá hacer con la cláusula as para definirla y poder emplearla en otros lugares de la consulta. Véase el siguiente ejemplo:

```
SELECT a*, numero
FROM asignaturas a JOIN
    (SELECT COUNT (DISTINCT alumno) as numero, asignatura
    FROM matricular
    GROUP BY asignatura)
    ON (a.codigo=asignatura)
WHERE numero >25
```

Listing 6.21: Ejemplo SQL 19.

Capítulo 7

SQL y otros elementos de la Base de Datos.

7.1. Operaciones de manipulación.

SQL además de permitirnos consultar información de las tablas como hemos visto en el capítulo anterior nos proporciona operaciones para manipular dicha información. Para ello están las sentencias $INSERT$, $UPDATE$, y $DELETE$. Esto supone una importante herramienta, pero ¿y si queremos ejecutar una secuencia de operaciones? Para ello están las transacciones.

Definición 36 Se define una transacción como un conjunto de operaciones de manipulación que se tratan atómicamente, es decir, como si fueran una única operación de forma que o se ejecutan todas las operaciones de la transacción o ninguna.

Operación se inserción.
La operación de inserción sirve como su nombre indica para insertar datos en tablas. La sentencia asociada a esta es $INSERT$ a la que hay que indicarle en qué tabla se van a insertar los datos y qué datos se insertarán. Puede insertarse fila a fila o varias a la vez. Para ello se distinguen dos esquemas básicos.

- INSERT INTO tabla VALUES (val1, val2, ... , valn) para insertar una única fila.
- INSERT INTO tabla SUBCONSULTA para insertar un conjunto de filas.

Además se le puede indicar el orden en que se le dan los atributos junto a la tabla de la forma tabla(col1, ..., coln) indicando que la columna i-ésima de la subconsulta se insertará en la í-ésima columna de la tabla. Observamos que en caso de que algún atributo no sea especificado se le asigna el valor nulo o uno por defecto en caso de haberlo.

Ejemplo 22 Matricular a todos los alumnos en el grupo B de la asignatura con código 111 del curso 22/23.

```
INSERT INTO matricular (asignatura, grupo, curso, alumno)
SELECT 1111, 'B', '22/23', dni
FROM ALUMNOS;
```

Listing 7.1: Ejemplo SQL 19.

Operación de modificación.
La modificación de tablas sigue también un patrón establecido. Se emplea la sentencia $UPDATE$ para indicar qué tabla se va a actualizar, le sigue SET para indicar el valor de los nuevos atributos y por último un $WHERE$ para indicar que se cambien únicamente para las filas que cumplan cierta condición. En ausencia de esta última sentencia se modifica para todas las filas de las tablas. Además, podemos utilizar subconsultas tanto en tanto para indicar los nuevos valores como en las condiciones. Vamos a ver el patrón y un ejemplo

```
UPDATE tabla
SET atrib1 = expr1, ..., atribn = exprn
WHERE condiciones
```

Listing 7.2: Patrón modificaciones.

Ejemplo 23 Asignar 6 créditos a todas las asignaturas del grupo B.

```
UPDATE asignaturas
SET creditos = 6
WHERE codigo IN (SELECT asigantura
----------------FROM matricular
----------------WHERE grupo = 'B');
```

Listing 7.3: Ejemplo 20 SQL.

Operación de eliminación.
Para eliminar distinguimos dos tipos de eliminación. La eliminación de tablas enteras o solo de ciertas filas. Cuando hablamos de tablas enteras se utiliza la sentencia $DROP$ mientras que cuando se trata de filas se emplea $DELETE$. En este último se suele seguir el esquema DELETE FROM tabla WHERE condición. Hacemos notar que no se dice qué eliminar de la tabla, se eliminan las filas completas que cumplen la condición de la cláusula WHERE.

Ejemplo 24 Borrar las asignaturas que no tienen a nadie matriculado.

```
DELETE FROM asignaturas asig
WHERE NOT EXISTS (SELECT *
----------------FROM matricular
----------------WHERE asignatura = asig.codigo);
```

Listing 7.4: Ejemplo 21 SQL.

Transacciones.
Las transacciones se definen mediante un conjunto de sentencias dentro de un bloque PL/SQL, el lenguaje de programación incrustado en Oracle. Se empieza en una línea con el comando $BEGIN$, después siguen las sentencias que conformen la transacción y se indica el final de esta con END. Además las sentencias del bloque deben acabar con uno de los siguientes comandos:

- COMMIT: confirma los cambios que se hayan realizado y se guardan en las tablas.
- ROLLBACK: es un comando que deshace los cambios y devuelve la base de datos al estado previo.

Destacamos que las sentencias DDL como CREATE, ALTER o DROP llevan un COMMIT implícito.

7.2. Elementos del nivel externo. Vistas y permisos.

Si recordamos la estructura en tres niveles de las bases de datos vista en el primer capítulo de este documento, en el nivel externo nos encontrábamos con vistas de la información creada para que los usuarios las empleen en sus labores. Además, cada usuario podía tener unos permisos u otros sobre estas vistas.

7.2.1. Vistas.

Las vistas se pueden utilizar como si fuesen una tabla más y siempre muestran la información actualizada. Para crear vistas, modificarlas o eliminarlas se utilizarán también las sentencias $CREATE$, $ALTER$, y $DROP$. El esquema general que sigue la creación de una vista es el siguiente:

```
CREATE [OR REPLACE] [FORCE] VIEW nombreVista (atrib1, ... atribn)
AS subconsulta_de_definicion
[WITH READ ONLY]
[WITH CHECK OPTION]
```

Listing 7.5: Esquema creación de vistas.

Las partes que aparecen entre corchetes son opcionales. Vamos a detallar qué hace cada una de ellas.

- OR REPLACE: en caso de que ya exista dicha vista, sobrescribe sobre ella la nueva vista indicada.
- FORCE: crea la vista a pesar de que no existan las tablas base o no se tengan permisos.

- WITH READ ONLY: impide que se pueda modificar la vista
- WITH CHECK OPTION: impide que se puedan modificar filas de la vista si el resultado no coincide con el que devolvería la subconsulta de definición. Por ejemplo si la vista se crea a partir de los deportistas con nacionalidad española, no se pueda añadir un atleta portugués.

A la hora de actualizar una vista debemos tener en cuenta que en caso de ser posible, los cambios se propagarán a las tablas base correspondientes. Para que una vista sea actualizable y se propaguen los cambios se deben cumplir dos condiciones:

1. Sus columnas hacen referencias únicamente a columnas de alguna tabla base, esto implica que no son resultado de ninguna expresión formada por algo que más que una columna de las tablas base.
2. La subconsulta de definición no contiene ni operaciones de conjuntos, ni DISTINCT ni funciones de agregación, ni group by, ni order by ni reuniones.

Ejemplo 25 Vamos a crear una vista con asignaturas con código menor a 130 con CHECK OPTION y la modificaremos provocando un error.

```
CREATE OR REPLACE VIEW asignaturasmenores130
AS
SELECT * FROM asignaturas WHERE codigo < 130
WITH CHECK OPTION;

INSERT INTO asignaturasmenores130 (codigo, nombre)
VALUES (300, 'Aprendizaje Computacional')
COMMIT;
```

Listing 7.6: Ejemplo 22 SQL.

7.2.2. Permisos.

Existen diferentes tipos de permisos en la base de datos. Son los siguiente: lectura (SELECT), inserción de filas (INSERT), modificación de filas ya existentes (UPDATE) y eliminación de filas (DELETE). Para conceder permisos a un usuario se utiliza la sentencia *GRANT* indicando los permisos sobre qué tabla y a qué usuario. Además se le puede indicar si ese usuario tiene también permiso para dar permiso a otros usuarios con WITH GRANT OPTION. Por otra parte, para quitar permisos se utiliza la sentencia *REVOKE*. Vemos cómo serían las estructuras generales.

```
-- dar permisos
GRANT permisos ON tabla TO usuario [WITH GRANT OPTION];
-- quitar permisos
REVOKE permisos ON TABLA TO usuario
```

Listing 7.7: Esquema permisos.

7.3. Metadatos.

Definición 37 Se definen los metadatos como datos que describen datos.

Dentro de las bases de datos, por la primera regla de Codd, los metadatos han de guardarse también en tablas del sistema y describen la estructura de los elementos de la base de datos.

7.3.1. Diccionario de datos.

Definición 38 Se define el diccionario de datos o catálogo del sistema como el conjunto de tablas que contienen metadatos.

En Oracle, las tablas que forman parte del diccionario de datos sólo son leídas y modificadas por el SGBD. A pesar de ello, tenemos unas vistas disponibles para consultar información. El nombre de las vistas del diccionario de datos empezará con alguno de los siguientes prefijos e indican lo siguiente:

- ALL: son vistas que describen los elementos a los que el usuario tiene acceso.
- DBA: son vistas que describen los elementos de la base de datos completa y sólo los administradores tienen permiso para utilizar dichas vistas.
- USER: son vistas que describen los elementos creados por el usuario.

7.3.2. Metadatos sobre tablas.

Vamos a ver en este apartado tres tablas interesantes que contienen metadatos sobre tablas.

- ALL_TABLES: esta es una tabla que describe las tablas a las que el usuario tiene acceso. Algunos de sus atributos son TABLE_NAME TABLESPACE_NAME o OWNER que indican el nombre de la tabla, el nombre del espacio de tablas en que se creó la tabla y el creador de la tabla respectivamente.
- USER_TABLES: esta es una copia de la anterior pero aquí solo se muestran las tablas de las que el usuario es propietario, no de todas a las que tiene acceso. Además se elimina el atributo OWNER pues siempre será el mismo.
- USER_TAB_COLUMNS: esta tabla describe las columnas de las tablas del usuario. Algunos de sus atributos son: TABLE_NAME, que guarda el nombre de la tabla o vista que contiene la columna; COLUMN_NAME, guarda el nombre de la columna; DATA_TYPE, guarda el tipo de dato; DATA_LENGTH, guarda la longitud del tipo de datos en bytes; DATA_PRECISION, DATA_SACALE, la precisión y longitud de un NUMBER; NULLEABLE, si se puede hacer nulo o no; COLUMN_ID, el orden del atributo al crearse la tabla.

7.3.3. Metadatos sobre restricciones.

Las restricciones también cuentan con tablas de metadatos. Las más interesantes son:

- USER_CONSTRAINTS: describe las restricciones creadas por el usuario y sus atributos más destacables son: CONSTRAINT_NAME (nombre único de la restricción), TABLE_NAME (nombre de la tabla que tiene dicha restricción), CONSTRAINT_TYPE (tipo de restrcción de la que se trata) SEARCH_CONDITION (predicado de la restricción si es de tipo C (check)), R_CONSTRAINT_NAME (define la clave destino, es decir, en caso de ser una restricción de tipo R (clave foránea) indica a qué tabla y a qué columna hace referencia).
- USER_CONS_COLUMNS: esta describe las columnas que aparece en las restricciones que ha creado el usuario, es decir, si la anterior describía la restricción, esta describe a las columnas que participan de la restricción. Sus principales atributos son: CONSTRAINT_NAME, TABLE_NAME, COLUMN_NAME y POSITION.

7.4. Disparadores o Triggers.

Definición 39 Se define un disparador o Trigger como un elemento que especifica acciones a realizar desencadenadas por una operación DML, operación DDL o por un evento de la Base de Datos.

Esto quiere decir que son básicamente una serie de acciones que se ejecutan sin necesidad de ser llamadas cuando sucede el evento que las dispara. Observamos también que un disparador, al no ser una función ni similares, no admite argumentos. Igual que las tablas, tienen una vista USER_TRIGGERS donde se guardan los disparadores creados por el usuario. Con unos ejemplos de uso puede entenderse mejor el concepto.

Ejemplo 26 Ejemplos de la usabilidad de un disparador.

1. Mantenimiento de restricciones de integridad complejas como por ejemplo que el sueldo de un empleado solo pueda aumentar.
2. Auditoría de una tabla, es decir, registrar los cambios efectuados en la tabla y quién los ha realizado.
3. Lanzar cualquier acción al modificar una tabla.

7.4.1. Creación de un Trigger.

Este es el esquema de creación de un trigger. Los elementos que aparecen entre corchetes, [ ], son opcionales y pueden o no aparecer $\{BEFORE|AFTER\}$ significa que BEFORE o AFTER, solo uno de ellos. Vamos a ver ahora qué quieren decir cada uno de los elementos que aparecen en las observaciones que siguen.

```
CREATE [OR REPLACE] TRIGGER nombreTrigger
{BEFORE | AFTER} sucesoDisparo ON tabla
[FOR EACH ROW [WHEN condicionDisparo ]]
    BLOQUE TRIGGER;
```

Listing 7.8: Esquema de creación de un trigger.

- sucesoDisparo es la operación DML que al efectuarse sobre tabla disparará el trigger. Así puede tomar los valores: INSERT, DELETE o UPDATE. No obstante, puede tomar más de uno de ellos, en ese caso se separan por la palabra OR. Pueden usarse predicados INSERTING, UPDATING o DELETING para discriminar el suceso del cual se trata.
- En caso de que el suceso que dispara el trigger sea un UPDATE, ¿de qué atributos? debemos indicar la actualización de qué atributos disparará el trigger. Por ello se indica UPDATE OF listaAtributos.
- BEFORE y AFTER indican si el trigger se ejecuta antes o después de la operación que lo ha causado. Obviamente, si se indica after se ejecutará el trigger después de la acción y si se indica before, antes.
- FOR EACH ROW nos indica el nivel del trigger. Se dice que un trigger es a nivel de tabla si se activa solo una vez, ya sea antes o después de la operación DML. Por otro lado se dice que es a nivel de fila si se activa una vez por cada fila afectada en la operación DML. Que a la hora de crear el trigger se incluya FOR EACH ROW quiere decir que es a nivel de fila, si no se incluye, se interpreta a nivel de tabla.
- BLOQUE TRIGGER es el bloque del trigger, es decir, indica la secuencia de acciones a realizar tras dispararse el trigger. Observamos que su cuerpo es un bloque de código PL/SQL.
- El cuerpo del bloque no tiene órdenes de control de transacciones ni rutinas llamadas por el disparador. Tampoco se pueden crear crear variables de tipo LONG o LONG RAW.
- No se puede acceder a cualquier tabla, esto es debido al problema de la tabla mutante, es decir, acceder a ciertas tablas puede dar problemas de actualizaciones y derivados.

7.4.2. Orden de ejecución.

1. Se ejecutan los disparadores de tipo BEFORE a nivel de tabla.
2. Para cada fila afectada por la orden:

a) Ejecutar disparador BEFORE a nivel de fila si cumple la condición de la cláusula WHEN en caso de existir.

b) Ejecutar la orden en sí.

c) Ejecutar disparadores AFTER a nivel de fila si cumple la condición de la cláusula WHEN en caso de existir.

3. Ejecutar disparadores de tipo AFTER a nivel de tabla.

7.4.3. Variables de acoplamiento.

Antes hemos mencionado que una de las funcionalidades de un trigger puede ser comprobar que el sueldo de un empleado no pueda bajarse, es decir, que si se actualiza la tabla, el valor a insertar sea mayor, pero, ¿cómo accedo o cómo sé si ese valor es realmente mayor si aún no lo he guardado? Para ello se utilizan las variables de acoplamiento. Solamente en caso de que que sea un trigger a nivel de fila podemos tener dos valores para los atributos, el antiguo y el nuevo. A ellos se accederá como :old.atributo y :new.atributo.

Ejemplo 27 Buscamos en un sistema de suministros actualizar el código de una pieza, pero queremos actualizarlo en toda la tabla, es decir, al cambiarlo en una fila, para el resto de filas correspondientes a la misma pieza, también le cambia el código.

```
CREATE [OR REPLACE] TRIGGER actualizaSuministrosPieza
BEFORE UPDATE OF CODIGO ON PIEZA FOR EACH ROW
BEGIN
    UPDATE SUMINISTROS SET PIEZA = :new.CODIGO
    WHERE PIEZA = :old.CODIGO
```

Listing 7.9: Ejemplo SQL 23.

Ahora buscamos crear un sistema de auditoría sencillo donde almacenar quien ha cambiado qué en la base de datos.

```
CREATE [OR REPLACE] TRIGGER controlAsignaturas
AFTER INSERT OR DELETE OR UPDATE ON ASIGNATURAS
BEGIN
    IF INSERTING THEN -- solo se ejecuta si se dispara por un insert
        INSERT INTO controlTablas (TABLA, USUARIO, FECHA, OPERACION)
        VALUES ('ASIGNATURAS', USER, SYSDATE, 'INSERT');
    ELSIF DELETING THEN -- solo se ejecuta para delete
        INSERT INTO controlTablas (TABLA, USUARIO, FECHA, OPERACION)
        VALUES ('ASIGNATURAS', USER, SYSDATE, 'DELETE');
    ELSE -- updating
        INSERT INTO controlTablas (TABLA, USUARIO, FECHA, OPERACION)
        VALUES ('ASIGNATURAS', USER, SYSDATE, 'UPDATE');
    END IF;
END controlAsignaturas;
```

Listing 7.10: Ejemplo SQL 24.

7.4.4. Disparadores de sustitución.

Definición 40 Se define un trigger de sustitución como aquel que se ejecuta en lugar de la orden DML que lo dispara.

Esto quiere decir que no se ejecutan ni antes ni después de la orden que lo dispara puesto que dicha orden no se llega a ejecutar nunca, es sustituida por el disparador. Para poder crear un trigger de este tipo debe definirse sobre una vista y a nivel de filas. Para declararse se sustituye el after o before por INSTEAD OF. Un ejemplo típico de uso es que en lugar de borrar en la vista se borre en la tabla de la que proviene la vista.

7.5. Trabajos o Jobs.

Las bases de datos se emplean, para ello están los trabajos. Cuando hacemos una aplicación se requiere de una base de datos y se accede a ella, pero también puede acceder otra aplicación u otro usuario. Para ello se requiere un planificador que diga quién dispone de la base de datos en un momento determinado, por eso hay un planificador de trabajos, el DBMS_SCHEDULER.
De este modo un trabajo puede entenderse como un programa planificado. Se pueden ejecutar unidades de programa de BBDD o programas externos de forma local en otras bases de datos. La ejecución puede darse en tres circustancias:

- Time-based scheduling: se ha definido previamente la fecha y repetición del trabajo.
- Event-based scheduling: se realiza el trabajo cuando sucede un evento como puede ser una transacción.
- Dependency scheduling: se definen cadenas complejas de trabajos.

Así un trabajo es un objeto y tiene un dueño que es el usuario que lo crea. Cuando el programa va a ejecutarse, lo hace con las credenciales del trabajo y despierta un proceso para ejecutar el programa. Consta de estos pasos:

1. Recoge metadatos necesarios, argumentos del programa e información de privilegios.
2. Crea una sesión con las credenciales del dueño del trabajo, se comienza una transacción y se ejecuta el programa del trabajo.
3. Al completarse el programa se hace un commit y termina la transacción.
4. Cierra la sesión.

Cuando ya se ha terminado el trabajo interviene el planificador para planificar la siguiente ejecución si es necesario, modificar las tablas de trabajos indicando si se ha terminado o se ejecutará de nuevo y se insertará una entrada de log de trabajos, dicha tabla es USER_SCHEDULER_JOB_LOG.

7.5.1. Gestión de trabajos.

Tenemos las siguientes acciones básicas a realizar con trabajos:

- Ver trabajos: select * from user_scheduler_jobs;
- Borrar trabajos: DBMS_SCHEDULER.DROP_JOB('job1, ... , jobn');
- Deshabilitar trabajos: DBMS_SCHEDULER.DISABLE('job1, ... , jobn');
- Habilitar trabajos: DBMS_SCHEDULER.ENABLE('job1, ... , jobn');

7.5.2. Planificación de trabajos.

Para planificar trabajos podemos hacerlo según una frecuencia, con un intervalo de de repeticiones y dependiendo del día. Por ejemplo, para hacerlo cada semana usamos FREQ = WEEKLY. Si queremos que sea cada dos viernes, es decir, semanalmente pero con un intervalo e indicando el día se emplearía FREQ = WEEKLY; INTERVAL = 2; BYDAY = FRI;. Un último ejemplo para acabar la asignatura será si queremos repetir el trabajo cada año el día 28 de febrero. Utilizamos: FREQ = YEARLY; BYMONTH = FEB; BYMONTHDAY = 28. Podríamos indicarlo también con un -1 si fuese el último día del mes pero como es febrero, podría ser bisiesto y tener 29 días... Sería entonces -2... Así que resolvemos de la primera forma y nos libramos de problemas, o quien prefiera pensarlo... se deja como ejercicio.

CONCLUSIÓN.

Con esto acabamos la asignatura. Podríamos resumirla en hacer un esquema con el modelo entidad relación de qué vamos a guardar, hacer el paso a tablas para tenerlo en modelo relacional, ese modelo relacional puede normalizarse para que sea más sencillo y esté mejor hecho y por último tenemos que usar y meterle datos... que una base de datos sin datos pues como que no... Así en pocas palabras es rápido, pero hay que saber de qué estamos hablando un formalizarlo aunque sea un poco. Agradecer por último a Alberto Ramírez Collado por haber cursado la asignatura conmigo y hacerla más soportable y menos aburrida.

Bibliografía

[1] Bases de datos de propósito general. https://chat.openai.com

[2] Bases de análisis de datos. https://www.sas.com/es_es/insights/data-management/data-warehouse.html

[3] Bases de datos en memoria. https://es.wikipedia.org/wiki/Base_de_datos_en_memoria

[4] Bases de datos de alta disponibilidad. https://docs.oracle.com/es-ww/iaas/Content/cloud-adoption-framework/high-availability.htm

[5] Bases de datos locales. https://www.informaticaparatunegocio.com/lo-debes-saber-la-base-datos-local/

[6] Bases de datos centralizadas y distribuidas. https://www.diarlu.com/tipos-bases-de-datos-ejemplos/

[7] Bases de datos según su licencia. https://chat.openai.com

[8] Bases de datos jerárquicas. https://es.wikipedia.org/wiki/Base_de_datos_jer%C3%A1rquica

[9] Bases de datos en red. https://es.wikipedia.org/wiki/Base_de_datos_de_red

[10] Bases de datos deductivas. https://ayudaleyprotecciondatos.es/bases-de-datos/deductivas/

[11] Bases de datos orientas a objetos. https://ayudaleyprotecciondatos.es/bases-de-datos/orientas-a-objetos/#google_vignette

[12] Bases de datos XML. https://es.wikipedia.org/wiki/Base_de_datos_XML

[13] Bases de datos RDF. https://es.wikipedia.org/wiki/Resource_Description_Framework

[14] Bases de datos NOSQL. https://informatica.cv.uma.es/course/view.php?id=5103

[15] Bases de datos relacionales. https://www.ibm.com/mx-es/topics/relational-databases

[16] SQL. https://es.wikipedia.org/wiki/SQL