

CURSO 2022-2023
CICLO SUPERIOR DE DESARROLLO DE APLICACIONES WEB
IES AGUADULCE

Bases de Datos

Francisco Javier Sueza Rodríguez

24 de marzo de 2023

Índice general

1. Almacenamiento de la Información	5
1.1. Introducción	5
1.2. Los Ficheros de Información	6
1.2.1. ¿Que es un Fichero?	6
1.2.2. Tipos de Ficheros	6
1.2.3. Los Soportes de la Información	7
1.2.4. Métodos de Acceso a Ficheros	8
1.2.4.1. Ficheros Secuenciales	8
1.2.4.2. Ficheros de Acceso Directo	9
1.2.4.3. Ficheros Indexados	10
1.2.4.4. Otros Tipos de Organización	11
1.2.5. Parámetros de Utilización	12
1.3. Bases de Datos	13
1.3.1. Conceptos Básicos	13
1.3.2. Uso de las Bases de Datos	14
1.3.3. Ubicación de la Información	15
1.4. Modelos de Bases de Datos	16
1.4.1. Modelo Jerárquico	16
1.4.2. Modelo en Red	17
1.4.3. Modelo Relacional	17
1.4.4. Modelo Orientado a Objetos	18
1.4.5. Modelo NoSQL	19
1.4.6. Otros Modelos	21
1.5. Tipos de Bases de Datos	22
1.6. Sistemas Gestores de Bases de Datos	26
1.6.1. Funciones de SGBD	27
1.6.2. Componente de un SGBD	28
1.6.3. Arquitectura de un SGBD	29
1.6.4. Tipos de SGBD	30
1.7. SGBD Comerciales	32
1.8. SGBD Libres	32
1.9. Bases de Datos Centralizadas	33
1.10. Bases de Datos Distribuidas	33
1.10.1. Fragmentación en Bases de Datos Distribuidas	34
1.11. Enlaces de Ampliación	36
2. Unidad 2: Bases de Datos Relacionales - TODO	37

3. Unidad 3: Implantación de Bases de Datos Relacionales - TODO	38
4. Unidad 4: Realización de Consultas	39
4.1. Introducción	39
4.1.1. En Oracle	39
4.1.2. MySQL	40
4.1.3. Modelo de la Base de Datos de Ejemplo	40
4.2. La Sentencia SELECT	42
4.2.1. Cláusula SELECT	42
4.2.2. Cláusula FROM	43
4.2.3. Cláusula WHERE	44
4.2.4. Cláusula HAVING	44
4.2.5. Cláusula ORDER BY	45
4.2.6. Tipos de JOIN	46
4.3. Operadores	47
4.3.1. Operadores Relacionales	48
4.3.2. Operadores Aritméticos	49
4.3.3. Operadores Lógicos	50
4.3.4. Precedencias	50
4.4. Consultas Calculadas	51
A. Anexos Tema 4	52
A.1. Creación de Tablas y Preparación de Oracle y MySQL	52
A.1.1. Creación de las Tablas en Oracle	52
A.1.2. Creación de las Tablas en MySQL	54
Glosario	56
Bibliografía	58

Índice de figuras

1.2.1.	Clasificación de ficheros según su función	7
1.2.2.	Tipos de ficheros por método de acceso	8
1.2.3.	Estructura de un fichero secuencial	8
1.2.4.	Modos de acceso a un registro en archivos de acceso directo	9
1.2.5.	Ficheros indexados	10
1.4.1.	Modelo Jerárquico de Bases de Datos	17
1.4.2.	Modelo en Red de Bases de Datos	17
1.4.3.	Modelo Relacional de Bases de Datos	18
1.4.4.	Modelo Orientado a Objetos de Bases de Datos	19
1.6.1.	Arquitectura de SGBD	30
1.7.1.	SGBD comerciales	32
1.8.1.	SGBD libres	32
1.9.1.	Ventajas e Inconvenientes de las bases de datos centralizadas	33
1.10.1.	Base de datos distribuida	34
1.10.2.	Ventajas e Inconvenientes de las bases de datos distribuidas	35
4.1.1.	Esquema entidad relación para la base de datos Oracle	40
4.1.2.	Esquema de paso a tablas para Oracle	41
4.1.3.	Grafo relacional de la base de datos para MySQL	41
4.1.4.	Esquema relacional en Workbench para MySQL	41
4.2.1.	Tipos de JOIN	46
4.3.1.	Tabla de operadores relacionales	48
4.3.2.	Operadores aritméticos en SQL	49
4.3.3.	Operadores lógicos en SQL	50
A.1.1.	Creación del espacio de trabajo y el usuario	52
A.1.2.	Línea de comandos de SQL en Oracle	53
A.1.3.	Conexión al espacio de trabajo creado	53
A.1.4.	Ejecución del script proporcionado	53
A.1.5.	Desconexión de la base de datos	54
A.1.6.	Inicio de sesión en el servidor MySQL	54
A.1.7.	Carga del script en MySQL Workbench	54
A.1.8.	Ejecución del script y creación de la base de datos	55
A.1.9.	Actualización de la lista de bases de datos en Workbench	55

Tema 1

Almacenamiento de la Información

En este primer tema, vamos a estudiar los conceptos básicos sobre el almacenamiento de la información, así como de las bases de datos y los SGBD (Sistemas de Gestión de Bases de Datos), pero en primer lugar, vamos a hacer una introducción más detallada sobre que consideramos información y el contenido de este módulo.

1.1 Introducción

Si pensamos cualquier en cualquier aspecto de nuestra vida cotidiana, o si analizamos la mayoría de ámbitos de actividad, nos encontramos que la utilización de bases de datos esta ampliamente extendida. Estás, y los datos contenidos en ellas, serán imprescindibles para llevar a cabo multitud de acciones.

Algunas de las situaciones en las que es necesario el uso de bases de datos son las siguientes:

- Cuando seleccionamos un canal de la TDT.
- Al utilizar la agenda del móvil para realizar una llamada telefónica.
- Cuando utilizamos un cajero automático.
- Cuando acudimos a la consulta del médico.
- Al inscribirnos en un curso, plataforma online, etc...
- Si utilizas el GPS.
- Cuando reservamos unas localidades en un evento deportivo.
- Cuando consultamos cualquier información en internet.
- Al solicitar un certificado de un organismo oficial.

Como vemos, el gran volumen de datos que manejamos y sus innumerables posibilidades hacen necesaria la existencia de técnicos perfectamente formados y capaces de trabajar con ellos.

Este módulo profesional se centra, precisamente, en las **Bases de Datos** y su uso en el desarrollo de aplicaciones. En esta primera unidad, comenzaremos conociendo los primeros sistema basados en ficheros para el almacenamiento y gestión de la información. Seguidamente, se desarrollarán los conceptos y definiciones básicas relacionados con las bases de datos, viendo también sus modelos y tipos. Más adelante conocer los sistemas gestores de bases de datos y finalmente, veremos las herramientas reales con las que llevar a caso dicha gestión.

1.2 Los Ficheros de Información

En esta sección vamos a hablar de los ficheros de información, en que consiste, que tipos nos podemos encontrar, métodos de acceso y parámetros de utilización.

1.2.1 ¿Que es un Fichero?

En la década de los setenta, los procesos básicos relacionados con una empresa se centraban en la contabilidad y facturación. Las necesidades de almacenamiento y gestión de la información podían satisfacerse con un número relativamente reducido de archivos de papel agrupados y ordenados, los típicos ficheros clásicos.

Con la primera informatización, se paso del papel al ordenador, pudiendo acceder a los datos de forma mucho más rápida. Los ordenadores adaptaron sus herramientas para que se asemejaran a las que los usuarios utilizaban manualmente, de forma que en informática también empezó a hablarse de ficheros, carpetas, formularios, etc...

La información que empezó a tratarse en los ordenadores debía ser almacenada para su posterior recuperación, consulta y procesamiento. El elemento que se creo para almacenar esta información fue el **fichero** o **archivo**.

Podemos definir un **fichero** como el **conjunto de información relacionada**, tratada como un todo y organizada de **forma estructurada**. Es una secuencia de dígitos binarios que organiza información relacionada con el mismo aspecto.

Los fichero están formados por **registros lógicos** que contienen información relativa a un mismo elemento u objeto (por ejemplo, información de un usuario). A su vez, los registros están divididos en **campos** que tienen cada una de las informaciones elementales que forman un registro (por ejemplo, nombre de usuario, email,...).

Los datos están almacenados de forma que se pueda añadir, suprimir, actualizar y consultar, individualmente, en cualquier momento.

Como los ficheros suelen ser muy grandes, solo se puede llevar parte de ellos a la memoria principal para procesarlos. La cantidad de información que es transferida entre el soporte en el que se almacena el fichero y la memoria del ordenador, en solo una operación de lectura/escritura, se llama **registro físico** o **bloque**.

Normalmente en cada operación de lectura/escritura se transfieren varios registros de un fichero, es decir, un bloque suele contener varios registros lógicos. Al número de registros que entran en un bloque se le llama **factor de blocaje**, y a la operación de agrupar varios registros en un mismo bloque se conoce como **bloque de registros**.

1.2.2 Tipos de Ficheros

Según la función que vaya a desempeñar un fichero, estos pueden ser clasificados de varias maneras:

- (a) **Ficheros Permanentes:** contiene información relevante para una aplicación. Es decir, los datos necesarios para el funcionamiento de ésta. Tiene un período de permanencia en el sistema amplio. Se subdividen en:
 - **Ficheros Maestros:** contiene el estado actual de los datos que pueden modificarse desde la aplicación. Es la parte central de aplicación, su núcleo.

- **Ficheros Constantes:** son aquellos que incluyen datos fijos de la aplicación. No suelen ser modificados y se accede a ellos para la realización de consultas.
 - **Ficheros Históricos:** contiene datos que fueron considerados como actuales en un período o situación anterior. Se utilizan para la reconstrucción de situaciones o estados concretos.
- (b) **Ficheros Temporales:** se utilizan para almacenar datos que son útiles para una parte de la aplicación. Son generados a partir de datos de ficheros permanentes y tienen un período corto de existencia. Estos pueden ser:
- **Ficheros Intermedios:** almacenan resultados de una aplicación que serán usados por otra.
 - **Ficheros de Maniobras:** almacenan datos de una aplicación que no pueden ser mantenidos en memoria por falta de espacio.
 - **Ficheros de Resultados:** almacenan datos que van a ser transferidos a un dispositivo de salida.

En la siguiente figura podemos ver un esquema con esta clasificación.

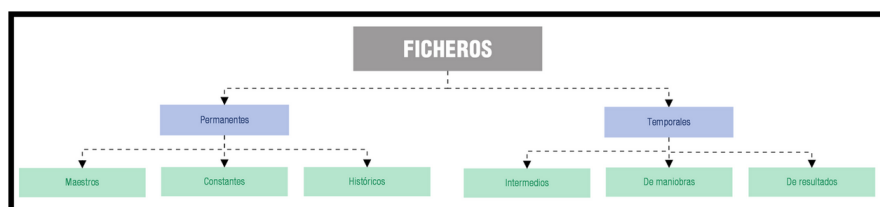


Figura 1.2.1.: Clasificación de ficheros según su función

1.2.3 Los Soportes de la Información

Los ficheros se almacenan en soportes de información manejados por periféricos del ordenador, que permiten leer y grabar datos en el soporte. Los soportes más utilizados son las **cintas magnéticas** y los **discos** (magnéticos, ópticos o magneto-ópticos).

Al principio se usaban tambores de cinta magnética, similares en tamaño a un disco de vinilo, funcionaban de manera similar a los antiguos casetes, pero al tener un tamaño mucho más grande permitían almacenar mucha más información, permitiendo el acceso a esta de forma secuencial.

Posteriormente los medios de almacenamiento fueron evolucionando a la par que el hardware, en concreto con la aparición de los disquetes y el disco duro. Estos dispositivos ya permitían el acceso aleatorio a los datos.

Por lo tanto, podemos distinguir dos tipos de dispositivos de almacenamiento de datos:

- **Soportes de Acceso Directo a Datos:** son los más empleados y el acceso a datos se hace de forma directa, pudiendo colocarlos en la posición que más nos interese.
- **Soporte de Acceso Secuencial:** se suele usar en copias de seguridad y si deseamos leer un dato que está a mitad de la cinta, tendremos que leer todo lo que hay hasta llegar a esa posición.

Si quieres aprender más sobre las características de cintas y discos, puedes consultar los enlaces siguientes:

- [Cintas magnéticas de almacenamiento de datos](#)
- [Discos magnéticos y Discos ópticos](#)

1.2.4 Métodos de Acceso a Ficheros

A medida que la tecnología ha ido evolucionando, el acceso a la información ha ido variando mucho. Los objetivos fundamentales de estas variaciones son los siguientes:

- Proporcionar acceso rápido a los registros.
- Conseguir economizar el almacenamiento.
- Facilitar la actualización de los registros.
- Permitir que la estructura refleje la organización real de la información.

Los ficheros se pueden clasificar según como se organiza en la memoria principal, o dicho de otra forma, los métodos de acceso al fichero, que podemos ver en la siguiente figura.

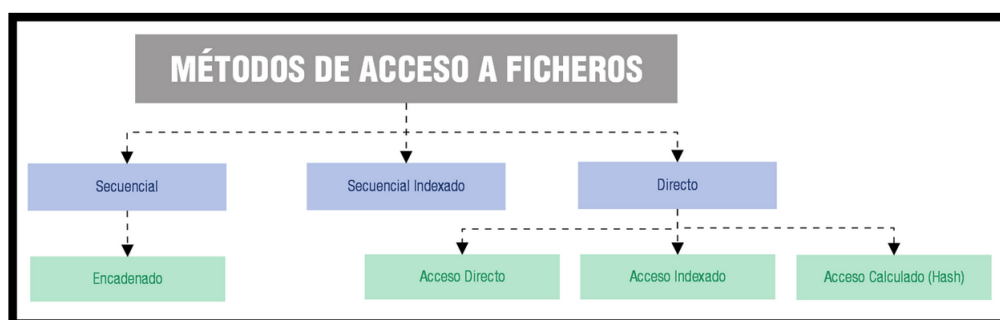


Figura 1.2.2.: Tipos de ficheros por método de acceso

Las organizaciones secuencia, de acceso aleatorio o directo e indexado, son las mas comunes. En esta sección vamos a explicar las características de cada uno de los métodos de acceso a ficheros.

1.2.4.1 Ficheros Secuenciales

Un **fichero secuencial** se caracteriza porque sus registros están almacenados de forma continua, de forma que la única manera de acceder a él, es leyendo un registros detrás de otro hasta el final. En los ficheros secuenciales hay una marca que indica el final del fichero, suele denominarse **EOF** (End Of File). Así, para detectar el final de fichero solo es necesario encontrar esta marca.

Este tipo de fichero puede usar dispositivos de almacenamiento de acceso secuencial, como cintas magnéticas, aunque también se utilizan en los CD de audio y DVD de vídeo, en los que la música y las imágenes se almacenan en un espiral continua.

Los registros almacenados se identifican por medio de la información ubicada en uno de sus campos, que se denomina **clave** o **llave**. Si se ordena un archivo secuencial por su clave es más rápido realizar operaciones sobre el.



Figura 1.2.3.: Estructura de un fichero secuencial

Algunas características de este tipo de ficheros son las siguientes:

- La **lectura** siempre se realiza **hacia adelante**.
- Son ficheros **monousuario**, no permiten el acceso simultáneo de varios usuarios.
- Tiene una **estructura rígida de campos**. Todos los registros deben aparecer en orden, es decir, la posición de los campos en el registros siempre debe ser la misma.
- El **modo de apertura** del fichero, condiciona la lectura o escritura.
- **Aprovechan al máximo** el soporte de **almacenamiento**, no dejando huecos vacíos.
- Se pueden **grabar** en **cualquier tipo** de **soporte**, tanto secuenciales como direccionales.
- Todos los **lenguajes de programación** contiene instrucciones para **trabajar** con este tipo de ficheros.
- No se pueden **insertar registros** en los que están **ya grabados**.

1.2.4.2 Ficheros de Acceso Directo

En este tipo de archivos se puede acceder a un registro indicando la posición relativa del registros dentro del archivo, o a través de una **clave** que forma parte del registro como un **campo** más. Estos archivos deben almacenarse en dispositivos de memoria masiva con acceso directo como los discos magnéticos.

Cada uno de los registros se guarda en una posición física, que dependerá del espacio disponible en memoria masiva, por lo que la distribución es aleatoria dentro del soporte de almacenamiento. Para acceder a la posición física del registros se utiliza una posición o índice, de forma que no es necesario recorrer todo el fichero para encontrar un determinado registros.

Esta **dirección física** se obtendrá tras la aplicación de una **transformación** específica a la **clave**. Según como sea esta transformación, existen tres modos de acceso diferente, como podemos ver en la siguiente figura.

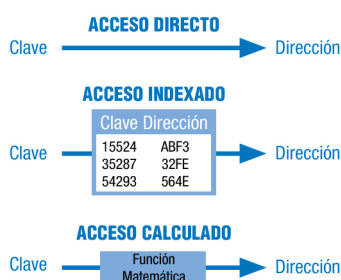


Figura 1.2.4.: Modos de acceso a un registro en archivos de acceso directo

El método más rápidos es el **acceso directo**, donde la clave coincide con la dirección física del registro, teniendo ésta que ser una posición valida dentro del rango de direcciones físicas.

La **medida de posicionamiento** básico del puntero en el fichero es el **byte**, dependiendo del tipo de codificación de caracteres que empleemos (**Unicode**, **ANSI**), se usarán 1 o 2 bytes por carácter respectivamente. Teniendo esto en cuenta, el puntero avanza de 1 en 1 o de 2 en 2 bytes para poder leer o escribir cada carácter.

Otras **características** de este tipo de ficheros son las siguientes:

- **Posicionamiento inmediato.**
- **Registros de longitud fija.**
- **Apertura** del fichero en **modo mixto**, para lectura y escritura.
- Permiten **múltiples usuarios** al mismo tiempo.
- Los **registros se borran** colocando un cero en la posición que ocupan.
- Permiten la utilización de **algoritmos de compactación** de huecos.
- Los archivos se **crean** con un **tamaño definido**, es decir, con un máximo de registros definidos durante su creación.
- Esta organización solo es posible en **soportes direccionales**.
- Se **usan** cuando el **acceso a datos** de un registro se hace siempre empleando la **misma clave** y la **velocidad de acceso** al registro es lo que más importa.
- Permiten la **actualización de registros** en el mismo fichero, sin necesidad de copiarlo.
- Permiten realizar **procesos de actualización en tiempo real**.

1.2.4.3 Ficheros Indexados

Se basan en el uso de **índices**, que permiten el acceso a un registros sin tener leer el fichero entero. Estos índices son similares a los de los libros, si nos interesa leer un capítulo podemos recurrir al índice donde se nos dice en que página comienza y acaba dicho capítulo.

Por lo tanto, deberá existir una **zona de registros** en los que se encuentren los datos del archivo y una **zona de índices**, que contiene la tabla con las claves de los registros y las posiciones donde se encuentran. La tabla de índices esta ordenada por campos clave.

La tabla de índices será cargada en la memoria principal para realizar en ella la búsqueda de la fila correspondiente a la clave del registros a encontrar, proporcionando así la dirección donde se encuentra el registro. Una vez localizada la dirección, solo es necesario acceder el dispositivo de almacenamiento y colocarlos en la dirección indicada.

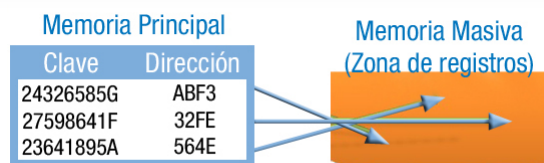


Figura 1.2.5.: Ficheros indexados

Las características mas relevantes de los ficheros indexados son las siguientes:

- El diseño de registros tiene que tener un campo o campos, que permita identificar cada registro de forma única, es decir, no puede haber 2 registros que tengan la misma información en él. A este campo se le llama **campo clave** y es el que va a servir de índice. Un mismo fichero puede tener varios campos clave, pero al menos uno de ellos **no puede tener valores duplicados** y se llama **clave principal**. A las restantes se les llama **claves alternativas**.

- Permite usar el modo de **acceso secuencial** y el modo de **acceso directo** para leer la información que guardan sus registros.
- Para acceder a estos ficheros usando el modo de **acceso directo**, se hace conociendo el **contenido del campo clave** que queremos localizar. Con esta información el sistema operativo puede consultar el índice y conocer la posición dentro del fichero.
- Para acceder a este tipo de ficheros usando el **modo secuencial**, los registros son **leídos** por el contenido del **campo clave**, independientemente del orden en el que fueron grabados, ya que el acceso se hace a través del índice, que para hacer más fácil la búsqueda de registros, permanece siempre ordenado por campos clave.
- Solamente puede **almacenarse** en un **medio direccionable**, ya que sino no podría usar el modo de acceso directo.

1.2.4.4 Otros Tipos de Organización

Además de los tipos de organización de ficheros que ya hemos visto, existen otros como los **ficheros secuenciales indexados** o los **ficheros de acceso calculado**, los cuales pasamos a describir a continuación.

(a) Ficheros Secuenciales Indexados:

También llamados parcialmente indexados, al igual que los ficheros indexados existe una **zona de índices** y otra **zona de registros de datos**, pero esta última se encuentra **dividida en segmentos** ordenados.

En la zona de índices, cada fila hace referencia a cada uno de los segmento. La clave corresponde al último registros del segmento y el índice al registro inicial. Una vez que se accede al primer registro del segmento, dentro de él se localiza (de forma secuencial) el registro buscado.

Las principales características de este tipo de ficheros son:

- Permite el **acceso secuencial**. Esto es muy interesante cuando la tasa de accesos es alta. En el acceso secuencial además los registros se leen ordenados por el campo clave.
- Permite el **acceso directo a registros**. Realmente **emula** este tipo de acceso, empleando para ello las tablas de índices. Primero busca la clave en el área de índices y luego va a leer al área de datos en la dirección que indica la tabla.
- Se pueden **actualizar** los **registros** en el **mismo fichero**, sin necesidad de crear uno nuevo de copia en el proceso de actualización.
- Ocupa **más espacio** que los **ficheros secuenciales**, debido al uso del área de índices.
- Solo se pueden utilizar **soportes direccionales**.
- Obliga a una **inversión económica mayor**, por la necesidad de **programas**, y a veces, **hardware más sofisticado**.

(b) Ficheros de Acceso Calculado o Hash:

Cuando usamos ficheros indexados es necesario siempre consultar una tabla para obtener la dirección de almacenamiento a partir de la clave. La técnica de acceso calculado o **hash**, permite accesos más rápidos, ya que en vez de consultar una tabla utiliza una **función matemática** (función de hashing) conocida, que a partir de la clave genera la dirección conocida de cada registro. Si la clave es alfanumérica deberá previamente ser transformada en un número.

El mayor problema que ofrece esta aproximación es que a partir de diferentes claves se puedan obtener la misma dirección de almacenamiento al aplicar la función. A este problema se le denomina **colisión**, y las claves que generan la misma dirección se denominan **sinónimos**. Para resolver este problema se aplican diferentes métodos, como tener un bloque de excedentes o zona de sinónimos, o crear un archivo de sinónimos, etc...

Para llevar a cabo la transformación se existen multitud de métodos, aunque los mas empleados son:

- **Módulo:** la dirección será igual al resto de la división entera entre la clave y el número de registros.
- **Extracción:** la dirección será igual a una parte de las cifras que se extraen de la clave.

Una buena función hash, será aquella que produzca el menor número de colisiones. En este caso hay que buscar una función, a poder ser **biunívoca**, que relacione los posibles valores de la clave con el conjunto de números correlativos de la dirección. Esta función consistirá en realizar una serie de cálculos matemáticos con el valor de la clave hasta obtener un número entre 1 y n, siendo n el número de direcciones que tiene el fichero.

1.2.5 Parámetros de Utilización

En función del uso que se le vaya a dar al fichero, serán convenientes unos u otros métodos de organización. Mediante la utilización de **parámetros de referencia** podemos determinar el uso de un fichero. Estos parámetros son:

- (a) **Capacidad o Volumen:** es el espacio, en caracteres, que ocupa el fichero. La capacidad podrá calcularse multiplicando el número de registros por el tamaño medio de cada registro.
- (b) **Actividad:** permite conocer la cantidad de consultas y modificaciones que se realizan en un fichero. Para poder especificar la actividad hay que tener en cuenta:
 - **Tasa de Consulta o Modificación:** que es el número de registros consultados o modificados en cada tratamiento del fichero, respecto al número total de registros contenidos en él.
 - **Frecuencia de Consulta o Modificación:** número de veces que se accede a un fichero para realizar una consulta o modificación en un tiempo predeterminado.

Volatilidad: mide la cantidad de inserciones y borrados que se efectúan en un fichero. Para determinar la volatilidad es necesario saber:

- **Tasa de Renovación:** es el tanto por ciento de registros renovados en cada tratamiento del fichero respecto del número de registros totales.
- **Frecuencia de Renovación:** es el número de veces que se accede al fichero para renovarlo durante un período de tiempo determinado.
- (c) **Crecimiento:** es la variación de la capacidad del fichero y se mide con la tasa de crecimiento, que es el porcentaje de registros en los que aumenta el fichero en cada tratamiento.

Teniendo en cuenta estos valores, podremos hacernos una idea de cual es el método de organización que mejor se adapta a nuestras necesidades y cual es el que deberemos usar en nuestros ficheros.

1.3 Bases de Datos

Como hemos visto en la sección anterior, los ficheros permiten organizar y memorizar conjuntos de datos de un mismo tipo con una determinada estructura, siendo un medio para almacenar la información o resultados de una aplicación. El problema es que si las aplicaciones, al ser diseñadas, dependen directamente de sus archivos, se pierde independencia y surgen serios inconvenientes: como información duplicada, incoherencia de datos, etc...

Aquí es donde aparece el concepto de base de datos. Una **base de datos** permitirá reunir toda la información relacionada en un único sistema de almacenamiento, pudiendo cualquier aplicación utilizarla de forma independiente y ofreciendo una mejora en el tratamiento de la información.

La gestión de las bases de datos a experimentado gran cantidad de cambios, partiendo de aplicaciones especializadas hasta pasar a convertirse en el núcleo de los entornos informáticos modernos. Con la llegada de internet en los 90, el número de usuarios de bases de datos creció exponencialmente, y aunque muchos no sean conscientes de ello, las usan a diario.

Así, conocer los sistemas de gestión de bases de datos, sus conceptos fundamentales, el diseño, lenguajes e implementación de estas, es imprescindible para cualquiera que se este formando en el campo de la informática.

1.3.1 Conceptos Básicos

Una **base de datos** es una colección de datos relacionados lógicamente entre sí, con una definición y descripción comunes y que están estructurados de una determinada manera. Es un conjunto de datos que representa entidades y sus relaciones, almacenados con la mínima redundancia y posibilitando el acceso a ellos eficientemente por parte de varias aplicaciones.

Las bases de datos no contienen solo los datos de la organización, sino que también almacenan una descripción de dichos datos. Esta descripción es lo que se denomina **metadatos**, se almacenan en un **diccionario de datos** o **catálogo** y es lo que permite la **independencia de datos** lógico-física.

Una base de datos, constará de los siguientes **elementos**:

- **Entidades**: objeto real o abstracto, con características diferenciadas de otros, del que se almacena información en la base de datos. En una base de datos de una clínica veterinaria, diferentes entidades podrían ser: ejemplar, doctor, consulta, etc...
- **Atributos**: son los datos que se almacenan en la entidad. Cualquier propiedad o característica puede ser un atributo de una entidad. Continuando con el ejemplo, podrían ser atributos: raza, color, nombre, número de identificación, etc...
- **Registros**: es donde se almacena la información de cada entidad. Es un conjunto de atributos que contienen los datos que pertenecen a una misma repetición de identidad. En nuestro ejemplo un registro podría ser: Podenco, blanco, 121932911, etc...
- **Campos**: donde se almacenan los atributos de cada registro. Teniendo en cuenta el ejemplo anterior, un campo podría ser Podenco.

El uso de bases de datos ofrece muchas **ventajas**, entre las que podemos encontrar las siguientes:

- **Acceso Múltiple**: diversos usuarios y aplicaciones podrán acceder a la base de datos sin que existan problemas en el acceso o los datos.
- **Utilización Múltiple**: cada uno de los usuarios o aplicaciones podrá tener una visión única de la base de la estructura de la base de datos, accediendo solo a la parte que le corresponde.

- **Flexibilidad:** la forma de acceso de la información puede ser establecida de diferentes maneras, ofreciendo tiempos de respuesta muy reducidos.
- **Confidencialidad y Seguridad:** el control de acceso de los datos podrá ser establecido para que los usuarios y aplicaciones puedan acceder a unos datos y a otros no, impidiendo a los usuarios no autorizados el uso de la base de datos.
- **Protección Contra Fallos:** en casos de fallos en la información, existen mecanismos bien definidos que permiten la recuperación de los datos de forma fiable.
- **Independencia Física:** un cambio en el soporte físico, por ejemplo un disco duro, no afectaría a los datos o las aplicaciones que acceden a estos.
- **Independencia Lógica:** los datos realizados en la base de datos no afectan a las aplicaciones que acceden a ella.
- **Redundancia:** los datos se almacenan, por lo general, una única vez, aunque si fuera necesario podríamos repetir la información de manera controlada.
- **Interfaz de Alto Nivel:** mediante la utilización de lenguajes de alto nivel puede utilizarse la base de datos de forma sencilla y cómoda.
- **Consulta Directa:** existe una herramienta para poder acceder a los datos de forma interactiva.

1.3.2 Uso de las Bases de Datos

Ya sabemos en que consiste una base de datos, ahora veremos que usuarios son los que la utilizan y en que campos se utilizan éstas.

Existen principalmente cuatro **tipos de personas** que pueden hacer uso de las bases de datos y cada uno de ellos hace un uso diferente de éstas. Estas personas son:

- **El Administrador**

Es la persona encargada de la creación o implementación física de la base de datos. Es quien escoge los tipos de ficheros, los índices que se deben crear, la ubicación de estos, etc... En general, es quien toma las decisiones del funcionamiento físico del almacenamiento de la información. Además, establecerá la política de seguridad y acceso para garantizar el menos número de problemas.

- **Los Diseñadores**

Son los encargados de diseñar como será la base de datos. Llevarán a cabo la identificación de los datos, sus relaciones, sus restricciones, etc... Para ello, han de conocer a fondo los datos y procesos que deben representarse en la base de datos. Si estamos hablando de una empresa, deberán conocer la reglas de negocio de esta. Para obtener un buen resultado, el diseñador deberá implicar a todos los usuarios de la base de datos lo antes posible en el proceso.

- **Los Programadores de Aplicaciones**

Una vez diseñada y construida la base de datos, los programadores se encargarán de implementar los programas de aplicación que servirán a los usuarios finales. Estos programas permitirán la posibilidad de realizar inserciones, actualizaciones o eliminaciones de datos. Para desarrollar estas aplicaciones se utilizan lenguajes de tercera o cuarta generación, como C, FORTRAN, Smalltalk, Ada, Java, etc...

■ Los Usuarios Finales

Son los clientes finales de la base de datos. Al diseñar, implementar y mantener la base de datos se busca cumplir con los requisitos establecidos por el cliente para la gestión de su información.

Respecto a los **campos** en los que se **usan la bases de datos** y los usos que se les dan son innumerables, aunque en la siguiente lista tienes algunos ejemplos:

- Banca: información de clientes, cuentas, transacciones, ...
- Líneas Aéreas: información de clientes, vuelos, horarios, ...
- Universidades: información de alumnos, asignaturas, profesores, horarios,...
- Telecomunicaciones: guardar registros de llamadas realizadas, generar facturas mensuales, mantener saldo de las tarjetas telefónicas y almacenar información sobre las redes.
- Medicina: información hospitalaria, biomedicina, genética, ...
- Legislación: normativas, registros, etc...
- Organismos Públicos: registros de los ciudadanos, certificados, etc...
- Justicia y Seguridad: delincuentes, casos, sentencias, investigaciones...

Como vemos, prácticamente en cualquier campo en el que se necesite recopilar, almacenar y gestionar información se utilizan las bases de datos.

1.3.3 Ubicación de la Información

Las bases de datos pueden tener un tamaño muy pequeño o ser muy voluminosas, pero independientemente de esto todas se almacenan en discos duros y otros dispositivos de almacenamiento a los que se puede acceder a través de un ordenador. Una base de datos pequeña pueden existir en un archivo pequeño dentro de un disco duro, mientras que una gran base de datos puede necesitar decenas de servidores en diferentes localizaciones.

En esta sección, vamos a ver los tipos de dispositivos y tecnologías de almacenamiento mas utilizados para el despliegue de bases de datos. En la siguiente lista, se detallan estos dispositivos:

-
- **Disco SATA:** es una interfaz de transferencia de datos entre la placa base y algunos dispositivos de almacenamiento como discos duros, lectores o grabadores de CD/DVD, unidades de estado solido u otros dispositivos. La interfaz SATA proporciona mayores velocidades, cables de conexión mas largos, mejor aprovechamiento cuando hay varios dispositivos conectados y capacidad de conectar unidades sin necesidad de apagar el ordenador. La primera generación tenía una tasa de transferencia de 150 MB/s, denominada **SATA 150 MB/s** o **Serial-ATA-150**. Actualmente se comercializan dispositivos **SATA II**, con velocidades de transferencia de 300 MB/s y **SATA III**, con velocidades de 600 MB/s.
- **Discos SCSI:** son interfaces preparadas para discos de gran capacidad de almacenamiento y gran velocidad de rotación. Se presentan bajo tres especificaciones: **Standard SCSI**, **Fast SCSI** y **Fast-Wide SCSI**. Su velocidad de acceso puede llegar a los 7 ms y la velocidad de transmisión de información secuencia a 5 MB/s, 10 MB/s y 20 MB/s para las versiones Standard, Fast y Fast-Wide respectivamente. Un controlador puede manejar hasta 7 discos duros SCSI.
- **RAID:** acronimo de **Redundant Array of Independent Disks**, es un contenedor de almacenamiento redundante. Se basa en montar varios discos duros para que trabajen conjuntamente

obteniendo mejoras en el almacenamiento, la velocidad, la disponibilidad y la seguridad de la información. Según las características que queramos reforzar se usará una u otra configuración RAID.

- **Sistemas NAS:** es el acrónimo de **Network Attached Storage**. Estos sistemas de almacenamiento permiten compartir el almacenamiento de un computador (servidor), con ordenadores personales o servidores clientes a través de la red, haciendo uso de un sistema operativo optimizado para dar acceso a los datos a través de protocolos de comunicación específicos. Suelen ser dispositivos de almacenamiento de gran capacidad, varios TeraBytes, generalmente superiores a los discos duros externos y que están conectados con la red.
- **SAN:** acrónimo de **Storage Area Network**. Se trata de una red concebida, arrays de discos y librerías de soporte. La arquitectura de este tipo de sistema permite que los recursos de almacenamiento estén disponibles para varios servidores en una red de área local o amplia. Debido a que la información almacenada no reside directamente en ninguno de los equipos de la red, se optimiza el poder de procesamiento para aplicaciones comerciales y la capacidad de almacenamiento se puede proporcionar al servidor que más lo necesite.

Aunque no se mencionen en esta lista, en los últimos años la tendencia es usar **bases de datos en la nube**, que permiten la utilización de bases de datos desarrolladas, implementadas y a las que se accede en un entorno de nube, como una nube privada, pública o híbrida. [1] En la [página de Oracle](#) puedes encontrar información más detallada sobre este tipo de bases de datos.

1.4 Modelos de Bases de Datos

La clasificación tradicional de las bases de datos establece tres modelos de bases de datos: **jerárquico**, **en red** y **relacional**. En la actualidad el modelo de datos más empleado es el relacional, aunque hay que tener en cuenta que dos de sus variantes, **bases de datos distribuidas** y **orientadas a objetos** son las más empleadas.

En esta sección, vamos a analizar estos modelos de bases de datos así como algunos otros aquí no mencionados.

1.4.1 Modelo Jerárquico

Cuando IBM creó su Sistema Administrado de la Información o IMS, se establecieron las bases para que la gran mayoría de sistemas de gestión de información de los años 70 utilizaran un modelo jerárquico. También recibe el nombre de modelo árbol, ya que utiliza una estructura de árbol invertido para el almacenamiento de los datos.

En el **modelo jerárquico**, la información se organiza con un jerarquía en el que se establece una relación entre las entidades **padre/hijo**. De tal forma que existen nodos que contienen atributos o campos y que se relacionan con sus nodos hijos, pudiendo tener cada nodo **varios hijos**, pero un nodo solo puede tener **un nodo padre**.

Los **datos** de este modelo se **almacenan** en estructuras lógicas llamadas **segmentos**. Los segmentos se relacionan entre sí usando **arcos**. Visualmente, este modelo se puede representar como un árbol invertido, estando en la parte superior los padres y en la inferior los hijos.

Hoy en día, debido a sus limitaciones, este modelo está en desuso. En la siguiente figura podemos ver un ejemplo de estructura jerárquica.

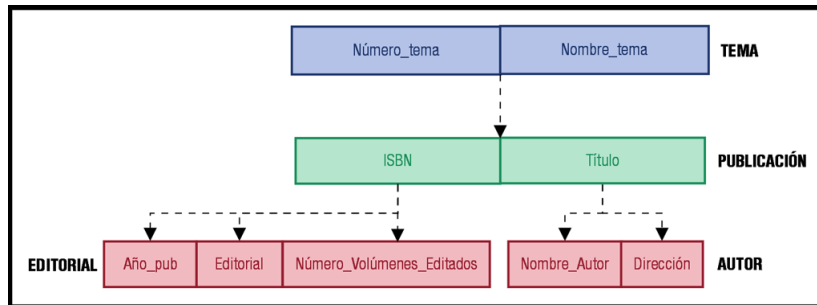


Figura 1.4.1.: Modelo Jerárquico de Bases de Datos

1.4.2 Modelo en Red

El modelo de datos en red aparece a mediados de los 60 como respuesta a las limitaciones del modelo jerárquico en cuanto a la representación de relaciones más complejas. Podemos considerar a **IDS** (Integrated Data Storage) de Bachman como el primer sistema de bases de datos en red. Más adelante, se intentó crear un modelo de red por parte de **CODASY**, siendo un modelo que tuvo una gran aceptación a principios de los 70.

El **modelo en red** organiza la información en **registros**, también llamados **nodos**, y **enlaces**. En los registros se almacenan los datos, mientras que los enlaces permiten relacionar los datos. Las bases de datos en red son parecidas a las jerárquicas, salvo que en éstas un nodo puede tener **más de un padre**.

En este modelo se puede representar prácticamente cualquier relación de datos, pero su manejo se hace muy complicado. Al no tener que duplicar la información se ahorra en espacio de almacenamiento. El sistema de gestión de la información más extendido es **IDMS**.

En la siguiente figura podemos ver un ejemplo de este tipo de modelo de bases de datos.

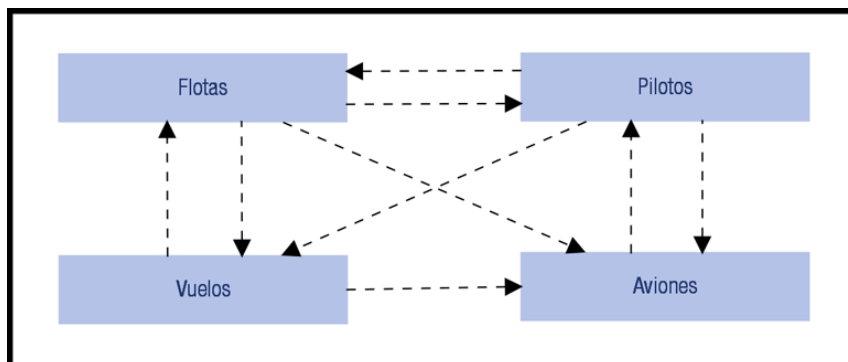


Figura 1.4.2.: Modelo en Red de Bases de Datos

1.4.3 Modelo Relacional

Este modelo es posterior a los dos anteriores y fue desarrollado por Codd en 1970. Hoy en día, este tipo de base de datos es la más utilizada.

El **modelo relacional** es percibida por los usuarios como un conjunto de tablas. Esta percepción es solo a nivel lógico, ya que a nivel físico puede estar implementada mediante diferentes estructuras de almacenamiento. Este modelo utiliza **tablas bidimensionales** (relaciones) para la representación lógica

de los datos y las relaciones entre ellos. Cada relación (tabla) posee un nombre único y contiene un conjunto de columnas.

Cada tabla estará compuesta de los siguientes elementos:

- **Registro/Entidad/Tupla:** es el nombre que recibe cada fila.
- **Campo/Atributo:** es el nombre que recibe cada columna.
- **Clave:** es el atributo o conjunto de estos que identifica de forma única a cada tupla.

Al conjunto de valores que puede tomar un atributo se le conoce como **dominio**. Además, las tablas deben cumplir un conjunto de **requisitos** para que se consideren correctas, que son los siguientes:

- Todos los registros son del mismo tipo
- La tabla solo puede tener un tipo de registro.
- No existen campos o atributos repetidos.
- No existen registros duplicados.
- No existe orden de almacenamiento de los registros.
- Cada registro o tupla debe estar identificada por una clave formada por uno o varios atributos.

En la siguiente imagen puedes ver como se relacionan las tuplas y atributos en una base de datos con el modelo relacional.

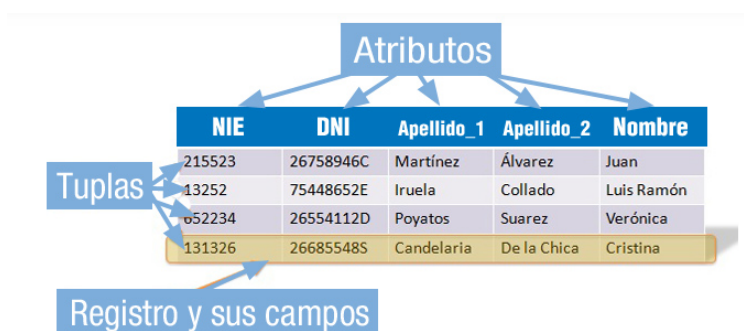


Figura 1.4.3.: Modelo Relacional de Bases de Datos

El lenguaje mas habitual para construir consultas en este tipo de bases de datos es **SQL** (Structured Query Language), un estándar implementado por los principales motores o sistemas de gestión de bases de datos.

Durante su diseño, una base de datos relacional para por un proceso que se conoce como **normalización**, que consiste en definir las reglas que determinan las dependencias entre los datos. Si definimos esta dependencia en una base de datos de la forma mas sencilla posible, conseguiremos que la cantidad de espacio necesaria para almacenar los datos sea la menor posible y la facilidad para actualizar la relación sea la mayor posible. Es decir, optimizaremos su funcionamiento.

1.4.4 Modelo Orientado a Objetos

El **modelo orientado a objetos** define una base de datos en términos de **objetos**, sus **propiedades** y sus **operaciones**. Los objetos con las misma estructura y comportamiento pertenecen a una misma **clase**, y las clases se organizan en jerarquías. Las operaciones de cada clase se especifican en términos

de procedimientos predefinidos denominados **métodos**. Algunos sistemas existentes en el mercado, basados en el modelo relacional, han sufrido evoluciones incorporando conceptos de la programación orientada a objetos. A estos modelos se les conoce como **modelos objeto-relacionales**.

El objetivo de este modelo es cubrir las limitaciones del modelo relacional. Gracias a este modelo se incorporan ventajas como la herencia entre tablas, los tipos definidos por el usuario, disparadores almacenables en la base de datos (triggers), soporte multimedia, etc..

En la siguiente imagen, podemos ver un ejemplo del modelo orientado a objetos.

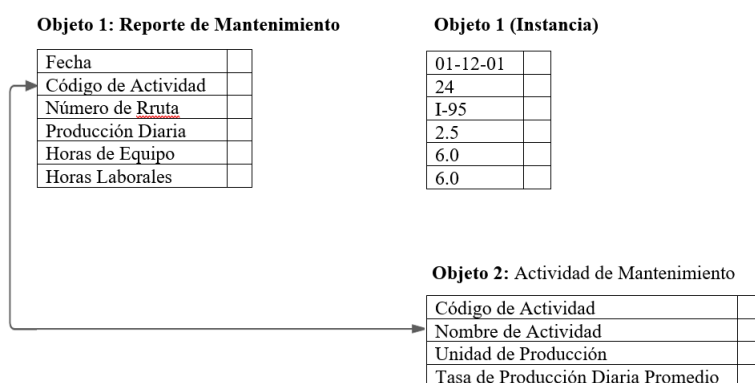


Figura 1.4.4.: Modelo Orientado a Objetos de Bases de Datos

Los conceptos más importantes del paradigma de objetos que incorpora el modelo orientado a objetos son los siguientes:

- **Encapsulación:** propiedad que permite ocultar la información al resto de objetos impidiendo así el acceso incorrecto o conflictos.
- **Herencia:** propiedad a través de la cual objetos heredan comportamientos dentro de la jerarquía de clases.
- **Polimorfismo:** propiedad de una operación mediante la cual puede ser aplicada a diferentes tipos de objetos.

Desde la aparición de la programación orientados a objetos (OOP) se empezó a pensar en bases de datos adaptadas a estos lenguajes. Este modelo es considerado como el fundamento de las bases de datos de tercera generación, siendo considerara las bases de datos en red como la primera y las relacionales como la segunda generación. Aunque no ha reemplazado a estas últimas, si es el tipo de base de datos que más esta creciendo en los últimos años.

1.4.5 Modelo NoSQL

Las **bases de datos NoSQL** son bases de datos que no cumplen con el esquema entidad-relación. Tampoco utilizan una estructura de datos de datos en forma de tabla donde se van almacenando los datos, sino que el para el almacenamiento se usan otros formatos como clave-valor, mapeo de columnas, graos, etc...

Esta forma de almacenar los datos tiene ciertas **ventajas** respecto a los modelos relacionales, las cuales son las siguientes:

- Se pueden **ejecutar** en máquinas con **pocos recursos**, no requieren apenas computación.

- **Escalabilidad horizontal:** para mejorar el rendimiento de estos sistemas simplemente se consigue añadiendo mas nodos, con la única operación de indicar al sistema cuales son los nodos que están disponibles.
- Puede manejar **gran cantidad de datos**, debido a que usan una **estructura distribuida** en muchos casos mediante **tablas Hash**.
- **No generan cuellos de botella:** el principal problema de los sistemas SQL es que necesitan transcribir cada sentencia para poder ser ejecutada, lo que constituye a un punto de entrada común, que ante muchas peticiones puede ralentizar el sistema.

Después de ver las principales ventajas que tiene el uso de este modelo, en la siguiente lista vemos las principales **diferencias** que nos podemos encontrar entre las bases de datos **NoSQL** y los sistemas **SQL** son las siguientes:

- **No utilizan SQL** como lenguaje de consulta. La mayoría de bases de datos NoSQL evitan usar este lenguaje o, como mucho, lo usan como apoyo.
- **No utilizan** estructuras fijas como **tablas** para el almacenamiento de datos. Permiten hacer uso de otros sistemas de almacenamiento de información como clave-valor, objetos o grafos.
- **No** suelen permitir **operaciones JOIN**. Al disponer de un volumen de datos tan extremadamente grande cuando la operación no es la búsqueda de una clave, la sobrecarga puede llegar a ser muy costosa. La solución en este caso sería desnormalizar los datos, o bien realizar el JOIN mediante software en la capa de aplicación.
- **Arquitectura distribuida:** las bases de datos relacionales suelen estar centralizadas en una misma máquina o bien en una estructura master-slave, sin embargo en los casos NoSQL la información puede estar compartida por varias máquinas mediante mecanismos de tablas Hash distribuidos.

Como hemos comentado, este modelo de bases de datos no usa tablas para almacenar datos, sino que se emplean otros tipos de almacenamiento que no suelen usar en bases de datos relacionales. Así, dependiendo del **tipo de almacenamiento** que se use en una base de datos NoSQL, estas pueden clasificarse en:

- (a) **Bases de datos clave-valor:** es el modelo de bases de datos NoSQL más popular, además de ser el más sencillo en cuanto a funcionalidad. En este tipo de sistemas, cada elemento está identificado por una **llave única**, lo que permite la recuperación de información de una forma muy rápida. Información que habitualmente esta almacenada como un **objeto binario (BLOB)**. Se caracterizan por ser **muy eficientes** tanto para las **lecturas** y las **escrituras**. Algunos ejemplos de estas bases de datos son Cassandra, BigTable o HBase.
- (b) **Bases de datos documentales:** estas bases de datos almacenan la información como un documento, por norma general con una estructura simple como **JSON** o **XML** y donde se utiliza una **clave única** para cada registro. Este tipo permite, además de realizar búsquedas por clave-valor, realizar consultas más avanzadas sobre el contenido del documento. Son las bases de datos NoSQL **más versátiles**. Se pueden utilizar en muchos tipos de proyectos, incluso en muchos que tradicionalmente usarían bases de datos relacionales. Algunas de las más utilizadas son MongoDB o CouchDB.
- (c) **Bases de datos en grafo:** en este tipo de bases de datos, la información se presenta como nodos en un grafo y sus relaciones como aristas, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Este tipo de bases de datos ofrece una navegación más eficiente que las bases de datos relacionales. Algunos ejemplos de estas bases de datos son Neo4j, InfoGrid o Virtuoso.

Como vemos, hay diferentes clases de bases de datos NoSQL, cada una con sus características propias, con sus ventajas e inconveniente. Dependiendo de nuestras necesidades, podemos elegir una u otra, pero en la actualidad hay unas pocas **bases de datos NoSQL** que son las **más usada** y que son las siguientes:

- **Cassandra**: se trata de una base de datos creada por **Apache** y que esta basada en el modelo **clave-valor**. Dispone de un lenguaje propio para realizar la consultas, **CQL** (Cassandra Query Language). Cassandra esta desarrollada en Java, por lo que puede correr en cualquier sistema que tenga una JVM.
- **Redis**: se trata de una base de datos tipo **clave-valor**. Se puede imaginar como un array gigante en memoria para almacenar datos, los cuales pueden ser cadenas, hashes, conjuntos de datos o listas.
- **Mongo DB**: se trata de una base de datos **orientada a documentos** de **esquema libre**, es decir, cada entrada puede tener un esquema de datos diferente que nada tenga que ver con el resto de registros almacenados. Es bastante rápida a la hora de ejecutar sus operaciones ya que esta desarrollada en C++. Es una de las bases NoSQL favoritas de los desarrolladores.
- **CouchDB**: se trata de un sistema desarrollado por **Apache** y que funciona sobre sistemas **Linux** y **OSX**, pero no sobre Windows. Utiliza Javascript como principal lenguaje de interacción. Permite la creación de **vistas**, un mecanismo para crear combinaciones para retornar valores de varios documentos, es decir, CouchDB permite operaciones **JOIN** típicas de las bases de datos relacionales.

Como vemos hay mucha variedad de bases de datos NoSQL, si quieres ampliar información puedes consultar este blog sobre [Bases de Datos NoSQL](#) o este artículo sobre las [claves para elegir tu BD NoSQL](#).

1.4.6 Otros Modelos

Además de las bases de datos que ya hemos visto, existen otros modelos, que aunque no vamos a ver con tanta profundidad, son dignos de mención. Estos modelos, que en algunos casos son un evolución de los ya vistos, son los siguientes:

- **Modelo Objeto-Relacional**

Las bases de datos que pertenecen a este modelo, son un híbrido entre el modelo de bases de datos relacional y el orientado a objetos. El principal inconveniente de las bases de datos orientadas a objetos es el coste de la conversión de las bases de datos relacionales a estas.

En una **base de datos objeto-relacional** (BDOR) siempre se busca obtener lo mejor del modelo relacional, incorporando las mejoras ofrecidas por la orientación a objetos. En este modelo se siguen almacenando tuplas, aunque la estructura de las tuplas no esta restringida sino que las relaciones pueden ser definidas en función de otras, que es lo que denominamos herencia directa.

El estándar en el que se basa este modelo es el **SQL99**. Este estándar ofrece la posibilidad de añadir a las bases de datos relacionales procedimientos almacenados de usuarios, triggers, tipos definidos por el usuario, consultar recursivas, bases de datos **OLAP**, etc...

También permite añadir funciones que tengan código en algún lenguaje de programación como SQL, Java, C, etc...

La mayoría de bases de datos relacionales clásicas de gran tamaño, como Oracle, SQL Server, etc..., son objeto-relacionales.

■ **Modelo de Bases de Datos Deductiva**

Es modelo de bases de datos almacena la información y permite hacer deducciones a través de **inferencias**. Es decir, se derivan nuevas informaciones a partir de las que se han introducido explícitamente en la base de datos por parte del usuario.

Las **bases de datos deductivas** son también llamadas bases de datos lógicas, al basarse en lógica matemática. Surgieron para contrarrestar las limitaciones del modelo relacional para las respuesta a consultas recursivas y la deducción de relaciones indirectas entre los datos almacenados.

■ **Modelo de Bases de Datos Multidimensionales**

Son bases de datos ideadas para desarrollar aplicaciones muy concretas. Básicamente almacena sus datos con varias dimensiones, es decir que en vez de un valor, encontramos varios dependiendo de los ejes definidos o una base de datos de estructura basada en dimensiones orientada a consultas complejas y alto rendimiento. En una base de datos multidimensionales, la información se representa como matrices multidimensionales, cuadros de múltiples entradas o funciones de varias variables sobre conjuntos finitos. Cada una de estas matrices se denomina cubo. Eso facilita el manejo de grandes cantidades de datos dentro de las empresas, dándole a esto una amplia aplicación dentro de varias áreas y diferentes campos del conocimiento humano.

■ **Modelo de Bases de Datos Transaccionales**

Son bases de datos caracterizadas por su velocidad para gestionar el intercambio de información, se utiliza sobre todo en sistemas bancarios, análisis de calidad y datos de producción industrial. Son bases de datos muy fiables, ya que en ellas cada una de las operaciones de inserción, actualización o borrado se realizan completamente o se descartan.

1.5 Tipos de Bases de Datos

Como hemos visto, según el modelo de datos las bases de datos se pueden clasificar en diferentes tipos. Pero este no es la única clasificación de bases de datos que existe. Atendiendo a diferentes características y criterios, las bases de datos se pueden clasificar en los siguientes tipos:

(a) **Bases de datos según su contenido**

Las bases de datos se pueden clasificar según el tipo de contenido que albergan, en este caso las bases de datos pueden ser:

- **Bases de datos con información actual:** contienen información muy concreta y actualizada, normalmente, de tipo numérico: estadísticas, series históricas, resultados de encuestas, convocatorias de becas, ofertas de empleo, etc...
- **Directorios:** recogen datos sobre personas o instituciones especializadas en una actividad o materia concreta. Hay directorios de profesionales, de investigadores, de centros de investigación, de bibliotecas, etc...
- **Bases de datos documentales:** en este tipo, cada registro se corresponde con un documento, sea este de cualquier tipo: publicación impresa, documento audiovisual, gráfico, etc... Dependiendo de si incluye o no el contenido completo de los documentos que describen, podemos tener:
 - **Bases de datos de texto completo:** constituidas por los propios documentos en formato electrónico, por un volcado completo de los datos.

- **Archivos electrónicos e imágenes:** constituidas por referencias que permiten el enlace directo con la imagen del documento original, sea este un documento iconográfico o un documento impreso digitalizado en forma de imagen.
- **Bases de datos referenciales:** sus elementos no contienen el texto original, sino tan sólo información fundamental para describir y permitir la localización de los documentos impresos, sonoros, iconográficos, audiovisuales o electrónicos. En este sistema de información solo se pueden obtener referencias sobre documentos que habrá que localizar en otro servicio.

(b) **Bases de datos según su uso**

Las bases de datos también se puede clasificar según el número de usuario que la usen. Así, una base de datos puede ser:

- **Base de datos individual:** es una base de datos utilizada, básicamente, por una sola persona. El sistema administrador de base de datos y los datos son controlados por el mismo usuario. Puede estar almacenada en la unidad de disco duro del usuario o en el servidor de archivos de una red de área local. Por ejemplo, un gerente podría tener una base de datos para el control de sus vendedores y su desempeño.
- **Bases de datos compartida:** son bases de datos con múltiples usuarios y que probablemente pertenezcan a la misma organización, como la base de datos de una compañía. Se encuentra alojada en una computadora potente y bajo el control de un profesional en el área, el administrador de bases de datos. Los usuarios tienen acceso a la base de datos mediante una red de área local o área extensa.
- **Bases de datos de acceso público:** son bases de datos accesibles por cualquier persona. Puede no ser necesario pagar un canon para hacer uso de los datos contenidos en ellas.
- **Bases de datos propietarias o bancos de datos:** se tratan de bases de datos de gran tamaño y desarrolladas por una organización para contener información sobre temas especializados o de carácter particular. El público general puede tener acceso a estas bases de datos a veces de forma gratuita y otras pagando una cuota. Pueden ofrecer información que va desde negocios, economía, inversión, técnica y científica hasta servicios de entretenimiento.

(c) **Bases de datos según la variabilidad de la información**

La variabilidad de la información, es decir, si la información almacenada es estática o dinámica, es otro criterio por el que podemos clasificar las bases de datos. Siendo estas:

- **Bases de datos estáticas:** son bases de datos de sólo lectura. Se utilizan para el almacenamiento de datos históricos que pueden ser analizados y utilizados para el estudio del comportamiento de un conjunto de datos a través del tiempo. Permite realizar proyecciones y toma de decisiones.
- **Bases de datos dinámicas:** son bases de datos donde la información almacenada se modifica con el tiempo, permitiendo operaciones como actualización y adición de datos, además de operaciones fundamentales de consulta.

(d) **Bases de datos según la localización de la información**

Las bases de datos se pueden clasificar también según como tienen distribuida su información, pudiendo ser:

- **Bases de datos centralizadas:** son bases de datos ubicada en un único lugar, un único computador. Pueden ser sistemas monousuario que se ejecutan en un ordenador o sistemas de bases de datos de alto rendimiento que se ejecutan en grandes sistemas. Este tipo de organización facilita el mantenimiento, sin embargo, hace que la información almacenada en dicha base de datos sea más vulnerable a fallos y limita su acceso. Este tipo de bases de datos puede ofrecer dentro de la arquitectura Cliente/Servidor dos configuraciones:
 - **Basada en el anfitrión:** ocurre cuando la máquina cliente y servidor son la misma. Los usuarios se conectarán directamente a la máquina donde se encuentra la base de datos.
 - **Basa en cliente/servidor:** ocurre cuando la base de datos reside en una máquina servidor y los usuarios acceden a la base de datos desde su máquina cliente a través de la red.
- **Bases de datos distribuidas:** una base de datos distribuida consiste en un sistema de bases de datos formado por diferentes bases de datos situadas en diferentes espacios físicos o lógicos y conectadas mediante redes. Los usuarios utilizan estas bases de datos mediante una variedad de redes de comunicación. Un ejemplo, sería una empresa con diferentes centros regionales, cada uno con su base de datos, pero donde los gerentes pueden acceder a estas como si fuera una única base de datos.

(e) **Bases de datos según el órgano productos**

También se pueden clasificar las bases de datos según quien sea el productos de la información alojada en ésta. Podemos encontrar los siguientes tipos:

- **Bases de datos de organismos públicos y de la administración:** las bibliotecas y centros de documentación de los ministerios, instituciones públicas, universidades y organismos de investigación públicos, entre otros, generan gran cantidad de información. Estos sistemas pueden ser:
 - Bases de datos de acceso público, sean gratuitas o no.
 - Bases de datos de uso interno, con información de acceso restringido.
- **Bases de datos de instituciones sin ánimos de lucro:** fundaciones, asociaciones, sindicatos y organizaciones no gubernamentales elaboran frecuentemente sus propios sistemas de información especializados.
- **Bases de datos de entidades privadas o comerciales:** los centros de documentación, bibliotecas y archivos pueden elaborar diferentes tipos de sistemas de información:
 - Bases de datos de uso interno para facilitar la circulación de la información dentro de la empresa.
 - Bases de datos de uso interno que ocasionalmente ofrecen servicios hacia el exterior.
 - Bases de datos comerciales, diseñadas específicamente para ser usadas por usuarios externos.
- **Bases de datos realizadas por cooperación en red:** se trata de sistema de información cuya elaboración es compartida por diversas instituciones, bases de datos internacionales que se elaboran a través de este sistema de trabajo, con diversos centros nacionales responsables de la información perteneciente a cada país.

(f) **Bases de datos según su modo de acceso**

Otra forma de clasificar las bases de datos es atendiendo al modo en el que se accede a ellas, como puede ser:

- **Bases de datos de acceso local:** para consultar es necesario acudir al organismo productor, a su biblioteca o archivo. Pueden ser consultadas en monopuesto en varios puntos de la red local.
- **Bases de datos en CD-ROM:** pueden adquirirse por compra o suscripción bien directamente por un particular o por una biblioteca o centro de documentación que permita su consulta a sus usuarios. En algunas instituciones se instalan diferentes CD-ROMs en una red local para permitir su consulta desde cualquier ordenador conectado a dicha red.
- **Bases de datos en línea:** pueden consultarse desde cualquier ordenador conectado a internet. La consulta puede ser libre o exigir la solicitud previa de una clave personal de entrada, llamado password. Hay diferentes tipos de acceso en línea:
 - **Acceso vía telnet o mediante internet:** el usuario realiza una conexión estable al host (anfitrión) en donde se halla la base de datos, a través de internet. La interfaz de usuario instalada en el host determinará si la consulta debe hacerse por menús, comandos o expresión en un lenguaje determinado. Cuando un usuario accede vía telnet, inicia una sesión interactiva con el programa que gestiona la base de datos, que le permite realizar todas las posibles consultas que soporte el sistema: selección, combinación y visualización o impresión de resultados. En cualquier momento podrá visualizar todas las consultas realizadas hasta ese momento y establecer combinaciones entre ellas.
 - **Acceso vía Web:** conexión a través de un formulario existente en una página web, diseñado para realizar consultas en la base de datos.

Una misma **base de datos** puede tener **diferentes formas de acceso**, por ejemplo, puede tener acceso local y además una edición en CD-ROM y un acceso en línea. Sin embargo, puede haber diferencias entre el contenido al que se puede acceder por cada vía o el grado de actualización de la información.

(g) **Bases de datos según cobertura temática**

Por último, podemos clasificar las bases de datos según el tema de la información que contiene y la cobertura de este. Atendiendo a este criterio podemos encontrar los siguientes tipos de bases de datos:

- **Bases de datos científico-tecnológicas:** contienen información destinada a los investigadores de cualquier ámbito científico o técnico. A su vez, este grupo puede subdividirse en:
 - **Bases de datos multidisciplinarias.** abarcan varias disciplinas científicas o técnicas.
 - **Bases de datos especializadas:** recopilan y analizan documentos pertenecientes a una disciplina concreta: biomedicina, farmacéutica, química, etc...
- **Bases de datos económico-empresariales:** contienen información de interés para empresas y entidades financieras...
- **Bases de datos de medios de comunicación:** contienen información de interés para profesionales de los medios de comunicación de masas: prensa, radio, televisión...
- **Bases de datos de ámbito político-administrativo y jurídico:** contienen información para

los organismos de la administración y los profesionales de derecho: legislación, jurisprudencia, etc...

- **Bases de datos de ámbito sanitario:** además de las propias pertenecientes al primera grupo de bases de datos científicas, existen otras de interés en este sector: historiales médicos, archivos hospitalarios, etc...
- **Bases de datos para el gran público:** contienen información destinada a cubrir necesidades de información general, de interés para gran número de usuarios.

1.6 Sistemas Gestores de Bases de Datos

Para poder tratar la información contenida en las bases de datos se utilizan sistemas gestores de bases de datos, o SGBD, también llamados DBMS (DataBase Management System), que ofrece un conjunto de aplicaciones que permiten acceder y gestionar dichos datos.

Un **Sistema Gestos de Bases de Datos**, es el conjunto coordinado de programas, procedimientos, lenguajes, etc., que suministra, tanto a los usuarios no informáticos como a los analistas programadores, o al administrador, los medios necesarios para describir y manipular los datos contenidos en una base de datos, manteniendo su integridad, confidencialidad y seguridad.

El SGBD permite a los usuarios la creación y mantenimiento de una base de datos, facilitando la definición construcción y manipulación de la información contenida en éstas. La **definición** de una base de datos consistirá en especificar los tipos de datos, las estructuras y las restricciones que los datos han de cumplir a la hora de almacenarse en dicha base de datos. Por otro lado, la **construcción** una base de datos será el proceso de almacenamiento de datos concreto en un medio o soporte de almacenamiento que esté supervisado por el SGBD. Finalmente, la **manipulación** de la base de datos incluirá la posibilidad de realización de consultas para recuperar documentación específica, la actualización de datos y la generación de informes a partir de su contenido.

El uso de un SGBD ofrece muchas suscripción **ventajas** a la hora de gestionar una base de datos, algunas de ellas son las siguientes:

- Proporcionar al usuario una **visión abstracta** de los **datos**, ocultando la complejidad relacionada con como se almacenan dichos datos.
- Ofrecen **independencia física**, es decir, la visión que tiene el usuario de la información y la manipulación de los datos almacenados en la base de datos, es independiente de como estén almacenados físicamente.
- Disminuye la **redundancia** y la **inconsistencia** de datos.
- Asegura la **integridad** de los datos.
- **Facilitan** el **acceso a datos**, aportando rapidez y evitando la pérdida de datos.
- **Aumenta** la **seguridad** y **privacidad** de los datos.
- Mejoran la **eficiencia**.
- Permite compartir datos y **accesos concurrentes**.
- Facilita el **intercambio de datos** entre **diferentes sistemas**.
- Incorpora mecanismos de **copia de seguridad** y **recuperación** de datos para restablecer la información en caso de fallo.

El SGBD interacciona con otros elementos software del sistema, en concreto con el sistema operativo. Los datos almacenados de forma estructurada en la base de datos son usados por otras aplicaciones, será el SGBD quien ofrecerá una serie de facilidades a éstas para el acceso y manipulación de la información, basándose en métodos y funciones propias del sistema operativo.

1.6.1 Funciones de SGBD

Un SGBD realiza tres funciones principales, como es la descripción, manipulación y utilización de los datos. A continuación se detalla cada una de estas funciones.

1. Descripción

Permite al diseñador de base de datos crear las estructuras adecuadas para integrar los datos. Esta función es la que permite definir las tres estructuras de la base de datos: **Estructura interna**, **Estructura conceptual** y **Estructura externa**.

Esta función se lleva a cabo mediante el **lenguaje de descripción de datos** o **DDL**. Mediante este lenguaje, se definen la estructuras de datos, relaciones entre estos y las restricciones que deben cumplir.

Se especificarán las características de los datos a cada uno de los tres niveles:

- **A nivel interno** (estructura interna), se ha de indicar el espacio de disco reservado para la base de datos, la longitud de los campos, su modo de presentación, etc...
- **A nivel conceptual** (estructura conceptual), se proporcionan herramientas para la definición de identidades y su identificación, atributos de las mismas, interrelación entre ellas, restricciones de integridad, etc..., es decir, el esquema de la base de datos.
- **A nivel externo** (estructura externa), se deben definir la vistas de los diferentes usuarios través del lenguaje para la definición de estructuras externas. Además el SGBD se encargará de la transformación de las estructuras externas orientadas al usuario a las estructura conceptuales y de la relación de esta con la estructura física.

2. Manipulación

Permite a los usuarios de la base de datos buscar, modificar, añadir o suprimir los datos de la misma, siempre de acuerdo con las normas de seguridad dictadas por el administrador. Se llevará a cabo mediante un **lenguaje de manipulación de datos** o **DML** que facilita los instrumento necesarios para la realización de estas tareas.

También se encarga de definir **la vista externa** de todos lo usuarios de la base de datos o vistas parciales que cada usuario tiene de los datos definidos con el DDL.

Por manipulación podemos entender:

- La recuperación de información almacenada en la base de datos, conocida como **consulta**.
- La **inserción** de nueva información.
- El **borrado** de información almacenada.
- La **modificación** de la información de la base de datos.

3. Control

Permite al administrador de la base de datos establecer mecanismos de protección de las diferentes visiones de los datos asociadas a cada usuario, proporcionando elementos de creación y

modificación de dichos usuarios. Adicionalmente incorpora sistema de creación de copias de seguridad, carga de ficheros, auditoría, protección contra ataques, configuración del sistema, etc...

El lenguaje que implementa esta función es el **lenguaje de control de datos** o **DCL**.

Para desarrollar todas estas funciones, usaremos el **Lenguaje Estructurado de Consultas** o **SQL** (Structured Query Language). Este lenguaje proporciona sentencias para la realización de operaciones DDL, DML, y DCL.

SQL fue publicado por ANSI en 1986 y ha ido evolucionando a lo largo del tiempo. Además, los SGBD suelen proporcionar otras herramientas que complementan a estos lenguajes como generadores de formularios, informes, interfaces gráficas, generadores de aplicaciones, etc...

1.6.2 Componente de un SGBD

Un SGBD es un paquete de software complejo que ha de proporcionar servicios relacionados con el almacenamiento y la explotación de los datos de forma eficiente. Para realizar esto, incluyendo las funciones vistas en el punto anterior, cuenta con un conjunto de componentes, que son los siguientes:

1. Lenguajes de la base de datos

Cualquier SGBD ofrece la posibilidad de usar lenguajes e interfaces adecuadas para los diferentes usuarios. A través de los lenguajes se pueden especificar los datos que componen la base de datos, su estructura, relaciones, reglas de integridad, control de acceso, características físicas y vistas externas de los usuarios.

Los lenguajes del SGBD son:

- Lenguaje de Definición de Datos (**DDL**)
- Lenguaje de Manejo de Datos (**DML**)
- Lenguaje de Control de Datos (**DCL**)

2. Diccionario de Datos

Descripción de los datos almacenados. Se trata de información útil para los programadores de aplicaciones. Contiene la información lógica de las estructuras que almacenan los datos, su nombre, descripción, contenido y organización. En una base de datos relacional, el diccionario de datos aportará información sobre:

- Estructura lógica y física de la BD.
- Definición de tablas, vistas, índices, disparadores, procedimientos, funciones, etc...
- Cantidad de espacio asignado y utilizado por los elementos de la BD.
- Descripción de las restricciones de integridad.
- Información sobre los permisos asociados a cada perfil de usuario.
- Auditoría de acceso a datos, utilización, etc...

3. El Gestor de Base de Datos

Es la parte del software encargada de garantizar el correcto, íntegro, seguro y eficiente acceso y almacenamiento de los datos. Es componente es el encargado de proporcionar una interfaz entre los datos almacenados y los programas de aplicación que los manejan. Es un intermediario, entre

el usuario y los datos. También es el encargado de garantizar la seguridad, privacidad e integridad de los datos, controlando los accesos concurrentes e interactuando con el sistema operativo.

4. Usuarios de la Base de Datos

En un SGBD existen diferentes tipos de usuarios, cada uno de ellos con una serie de permisos sobre los objetos de la base de datos. Generalmente, existirán los siguientes usuarios:

- El **administrador** de la base de datos o **Database Administrator (DBA)**, que será la persona o conjunto de ellas encargadas de la administración de la base de datos. Tiene el control centralizado de la base de datos y es el responsable de su buen funcionamiento. Se encarga de autorizar el acceso a la base de datos, de coordinar y vigilar su utilización y de adquirir los recursos software y hardware necesarios.
- Los **usuarios** de la base de datos, que serán de diferentes tipos y con diferentes necesidades sobre los datos, así como con diferentes accesos y privilegios. Podemos hacer la siguiente clasificación:
 - Diseñadores
 - Operadores y personal de mantenimiento.
 - Analistas y programadores de aplicaciones.
 - Usuarios finales: casuales, simples, avanzados y autónomos.

5. Herramientas de la Base de Datos

Son un conjunto de aplicaciones que permiten al administrador la gestión de la base de datos, de los usuarios y permisos, generadores de formularios, informes, interfaces gráficas, etc...

1.6.3 Arquitectura de un SGBD

Un SGBD tiene una arquitectura que simplifica a los diferentes usuarios de la base de datos su labor. El objetivo fundamental es separar los programas de aplicación de la base de datos física.

Encontrar un estándar para esta arquitectura no es una tarea sencilla, aunque los tres estándares más importantes en el campo de las bases de datos son **ANSI/SPARC/X3**, **CODASYL** y **ODMG**, este último solo para bases de datos orientadas a objetos. Tanto ANSI (EEUU) como ISO (internacional), son el referente en cuanto a estandarización de bases de datos, conformando un único modelo de bases de datos.

La arquitectura propuesta proporciona tres niveles de abstracción: **nivel físico**, **nivel lógico o conceptual** y **nivel externo o visión del usuario**. Las características de cada uno de estos niveles son las siguientes:

- **Nivel físico:** En este nivel se describen la estructura física de la base de datos a través de un esquema interno encargado de detallar el sistema de la base de datos y sus métodos de acceso. Es el nivel más cercano al almacenamiento físico. A través del esquema físico se indica, entre otras cosas, los archivos que contienen la información, su organización, los métodos de acceso a registros, los tipos de registros, la longitud, etc...
- **Nivel lógico:** En este nivel se describe la estructura completa de la base de datos a través de un esquema que detalla entidades, atributos, relaciones, operaciones de los usuarios y restricciones. Los detalles del almacenamiento se ocultan, permitiendo una abstracción a más nivel.

- **Nivel externo:** En este nivel se describen las diferentes vistas que los usuarios tendrán de la base de datos. Cada grupo de usuarios verá sólo la parte de la base de datos que le interesa, ocultando el resto.

Para una base de datos, existirá solo un esquema interno, uno lógico y podrían existir varios esquemas externos definidos para uno o varios grupos de usuarios.

Gracias a esta arquitectura se consigue **independencia de datos** a dos niveles:

- **Independencia Lógica:** podemos modificar el esquema conceptual sin alterar los esquemas externos ni los programas de aplicación.
- **Independencia Física:** podemos modificar el esquema interno sin alterar el esquema lógico ni los externos. Es decir, se puede cambiar el sistema de almacenamiento, reorganizar ficheros, añadir nuevos, etc., si que esto afecte al resto de esquemas.

Esta arquitectura, como hemos visto, tiene muchas ventajas, siendo la empleada en la mayoría de SGBD. En la siguiente figura tenemos un esquema para poder visualizar de forma más gráfica esta arquitectura.

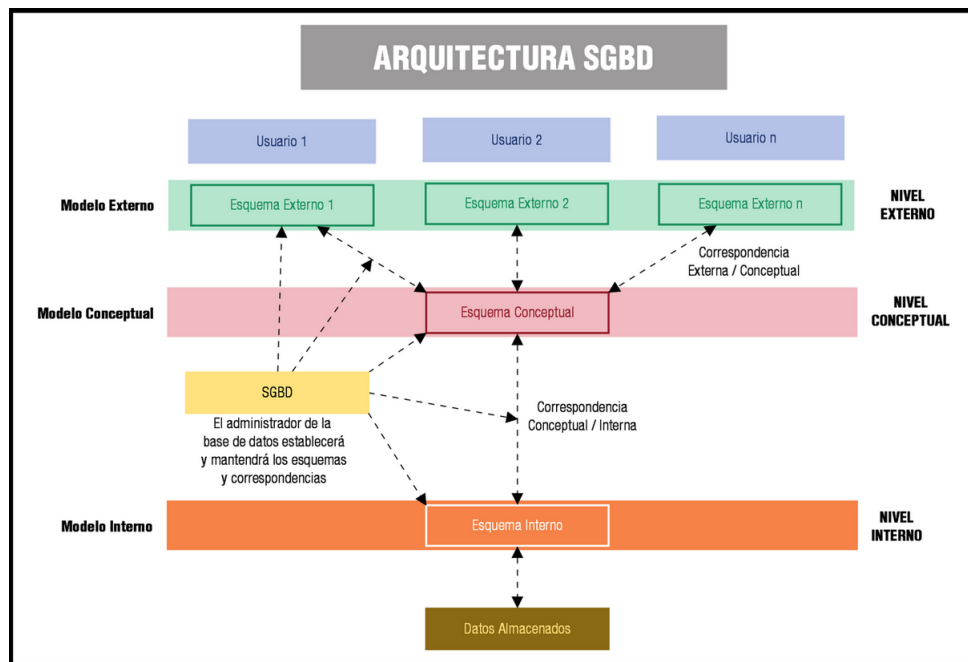


Figura 1.6.1.: Arquitectura de SGBD

1.6.4 Tipos de SGBD

Una vez que hemos visto las funciones y los componentes de un SGBD, vamos a ver los tipos que podemos encontrarnos atendiendo a diferentes criterios en la siguiente clasificación:

- Modelo Lógico:** este es el primer criterio que nos solemos encontrar. Actualmente, el modelo lógico más empleado es el **relacional**. Los modelos de red y jerárquicos han quedado obsoletos, pero si hay otros modelos que podemos encontrar como el objeto-relacional. Incluyendo todos los modelos, tenemos:
 - Modelo Jerárquico.

- Modelo en Red.
- Modelo Relacional.
- Modelo Orientado a Objetos.

En la sección 1.4, puede encontrar más información sobre los diferentes modelos y sus características.

- (b) **Número de Usuarios:** también podemos clasificar un SGBD por el número de usuarios al que prestan servicio, pudiendo ser éstos:
- **Monousuario:** solo atienden a un usuario a la vez y su principal uso es en ordenadores personales.
 - **Multiusuario:** atienden a varios usuarios a la vez y entre estos se encuentran la mayoría de SGBD.
- (c) **Distribución de la Base de Datos:** el tercer criterio es el número de sitios en los que esta distribuida la base de datos. Así un SGBD puede ser:
- **Centralizado:** sus datos se almacenan en un solo computador. Los SGBD centralizados pueden atender a varios usuarios, pero la base de datos y el propio sistema de gestión se encuentran alojados en un único computador.
 - **Distribuidos (Homogéneos, Heterogéneos):** la base de datos real y el propio SGBD pueden estar distribuidos en varios computadores conectados por red. Dentro de los sistemas distribuidos podemos encontrar dos tipos:
 - **Homogéneos:** utilizan el mismo SGBD en varios computadores diferentes.
 - **Heterogéneos:** una tendencia es crear software para acceder a varias bases de datos autónomas preexistentes almacenadas en sistemas distribuidos heterogéneos. Esta da lugar a SGBD federados o **sistemas multibase de datos**, en los que los SGBD participantes tienen cierto grado de autonomía.
- (d) **Coste del SGBD:** otro criterio de clasificación es el coste del paquete de software, estando en rangos desde los 0 a los 100.000 euros. Mientras que los sistemas más económicos monousuario para microcomputadoras pueden costar entre 0 y 3.000 euros, los paquetes más completos pueden llegar a los 100.000 euros.
- (e) **Propósito:** el último criterio de clasificación es el propósito que tiene el SGBD, es decir, si es de propósito general o específico, siendo estos:
- **Propósito general:** estos sistemas pueden ser utilizados para el tratamiento de cualquier tipo de base de datos.
 - **Propósito específico:** cuando el rendimiento es esencial se puede diseñar y construir un software de propósito especial para una aplicación específica, y este sistema solo servirá para esa aplicación. Por ejemplo, muchos sistemas de reserva de líneas aéreas son de propósito especial y pertenecen a la categoría de **sistemas de procesamiento de transacciones en línea**, que deben atender un gran número de transacciones concurrentes sin imponer excesivos retrasos.

Como vemos, hay muchos tipos de SGBD, ya dependerá de nuestras necesidades y de las restricciones del problema que queremos solventar cual es la elección más adecuada.

1.7 SGBD Comerciales

La elección de un SGBD es una decisión muy importante a la hora de desarrollar proyectos. A veces el sistema más avanzado, el “mejor” según los entendidos, puede no ser el más adecuado para el tipo de proyectos que estemos desarrollando. Hemos de tener en cuenta que volumen de carga puede soportar la base de datos, que sistema operativo utilizaremos como soporte, cual es el presupuesto, etc...

Actualmente en el mercado de software existen una gran variedad de SGBD comerciales. En la siguiente figura, mostramos los mas utilizados y cuales son sus características.

SGBD	Descripción
ORACLE	Reconocido como uno de los mejores a nivel mundial. Es multiplataforma, confiable y seguro. Es Cliente/Servidor. Basado en el modelo de datos Relacional. De gran potencia, aunque con un precio elevado hace que sólo se vea en empresas muy grandes y multinacionales. Ofrece una versión gratuita Oracle Database Express Edition 11g Release 2 .
MYSQL	Sistema muy extendido que se ofrece bajo dos tipos de licencia, comercial o libre. Para aquellas empresas que deseen incorporarlo en productos privativos, deben comprar una licencia específica. Es Relacional, 🗑️ Multihilo, Multiusuario y Multiplataforma. Su gran velocidad lo hace ideal para consulta de bases de datos y plataformas web.
DB2	Multiplataforma, el motor de base de datos relacional integra XML de manera nativa, lo que IBM ha llamado pureXML, que permite almacenar documentos completos para realizar operaciones y búsquedas de manera jerárquica dentro de éste, e integrarlo con búsquedas relacionales.
Microsoft SQL SERVER	Sistema Gestor de Base de Datos producido por Microsoft. Es relacional, sólo funciona bajo Microsoft Windows, utiliza arquitectura Cliente/Servidor. Constituye la alternativa a otros potentes SGBD como son Oracle, PostgreSQL o MySQL.
SYBASE	Un DBMS con bastantes años en el mercado, tiene 3 versiones para ajustarse a las necesidades reales de cada empresa. Es un sistema relacional, altamente escalable, de alto rendimiento, con soporte a grandes volúmenes de datos, transacciones y usuarios, y de bajo costo.

Figura 1.7.1.: SGBD comerciales

Aunque estos son los más importantes, podemos encontrar otros ampliamente usados como DBASE, ACCESS, INTERBASE y FOXPRO.

1.8 SGBD Libres

La alternativa a los SGBD comerciales la tenemos en los SGBD de código abierto o libres, también llamados **Open Source**. Son sistemas distribuidos y desarrollador libremente. En la siguiente figura se muestran los más importantes así como sus principales características.

SGBD	Descripción
MySQL	Es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. Distribuido bajo dos tipos de licencias, comercial y libre. Multiplataforma, posee varios motores de almacenamiento, accesible a través de múltiples lenguajes de programación y muy ligado a aplicaciones web.
PostgreSQL	Sistema Relacional Orientado a Objetos. Considerado como la base de datos de código abierto más avanzada del mundo. Desarrollado por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyados por organizaciones comerciales. Es multiplataforma y accesible desde múltiples lenguajes de programación.
Firebird	Sistema Gestor de Base de Datos relacional, multiplataforma, con bajo consumo de recursos, excelente gestión de la concurrencia, alto rendimiento y potente soporte para diferentes lenguajes.
Apache Derby	Sistema Gestor escrito en Java, de reducido tamaño, con soporte multilenguaje, multiplataforma, altamente portable, puede funcionar embebido o en modo cliente/servidor.
SQLite	Sistema relacional, basado en una biblioteca escrita en C que interactúa directamente con los programas, reduce los tiempos de acceso siendo más rápido que MySQL o PostgreSQL, es multiplataforma y con soporte para varios lenguajes de programación.

Figura 1.8.1.: SGBD libres

Cabe también mencionar **MariaDB**, por la relevancia y uso que esta adquiriendo en aplicaciones web en los últimos años.

1.9 Bases de Datos Centralizadas

Un **sistema de bases de datos centralizado** es aquel en el que el SGBD esta implantado en una sola plataforma u ordenador desde donde se gestiona directamente, de forma centralizada, la totalidad de los recursos. Es la arquitectura de los centros de procesado de datos tradicionales. Se basa en tecnologías sencillas, muy experimentadas y de gran robustez.

Los sistemas de los años sesenta y setenta era totalmente centralizados, como corresponde a los sistemas operativos de aquellos años, así como al hardware para el que estaban hechos: un gran ordenador para toda la empresa y una red de terminales sin memoria.

Las principales **características** de los sistemas centralizados son las siguientes:

- Se almacena completamente en la ubicación central, es decir, todos los componentes del sistema residen en un único sitio o computador.
- No posee múltiples elementos de procesamiento o mecanismos de comunicación como en las bases de datos distribuidas.
- Los componentes de la base de datos centralizada son: los datos, el software de gestión de bases de datos y los dispositivos de almacenamiento secundario.
- Son sistemas en los que su seguridad puede verse comprometida fácilmente.

Estos sistemas, tiene sus ventajas e inconvenientes respecto a los sistemas distribuidos. En la siguiente tabla, puedes ver una lista con los más comunes.

Ventajas	Inconvenientes
Se evita la redundancia debido a la posibilidad de inconsistencias y al desperdicio de espacio.	Un mainframe en comparación de un sistema distribuido no tiene mayor poder de cómputo.
Se evita la inconsistencia. Ya que si un hecho específico se representa por una sola entrada, la no-concordancia de datos no puede ocurrir.	Cuando un sistema de bases de datos centralizado falla, se pierde toda disponibilidad de procesamiento y sobre todo de información confiada al sistema.
La seguridad se centraliza.	En caso de un desastre o catástrofe, la recuperación es difícil de sincronizar.
Puede conservarse la integridad.	Las cargas de trabajo no se pueden difundir entre varias computadoras, ya que los trabajos siempre se ejecutarán en la misma máquina.
El procesamiento de los datos ofrece un mejor rendimiento.	Los departamentos de sistemas retienen el control de toda la organización.
Mantenimiento más barato. Mejor uso de los recursos y menores recursos humanos.	Los sistemas centralizados requieren un mantenimiento central de datos.

Figura 1.9.1.: Ventajas e Inconvenientes de las bases de datos centralizadas

1.10 Bases de Datos Distribuidas

La necesidad de integrar información de varias de fuentes y la evolución de las tecnologías de la comunicación, han producido cambios muy importantes en los sistemas de bases de datos. La respuesta a estas necesidades y evoluciones se materializa en los sistemas de bases de datos distribuidos.

Dentro de el paradigma de las bases de datos distribuidas, tenemos que tener claras las siguientes definiciones:

- **Base de Datos Distribuida (BDD):** es un conjunto de múltiples bases de datos, lógicamente relacionadas, las cuales se encuentra distribuidas entre diferentes nodos interconectados por una red de comunicaciones.
- **Sistema de Bases de Datos Distribuida (SBDD):** es un sistema en el que múltiples sitios de bases de datos están ligados por un sistema de comunicaciones, de tal forma que, un usuario en cualquier sitio puede acceder a los datos en cualquier parte de la red como si los datos estuvieran almacenados en su propio sitio.
- **Sistema Gestor de Base de Datos Distribuida (SGBDD):** es aquel que se encarga de manejar las BDD y proporciona un mecanismo de acceso que hace que la distribución sea transparente a los usuarios. El término transparente se refiere a que la aplicación trabajaría, desde un punto de vista lógico, como si un solo SGBD, ejecutado en una sola máquina, administrara esos datos.

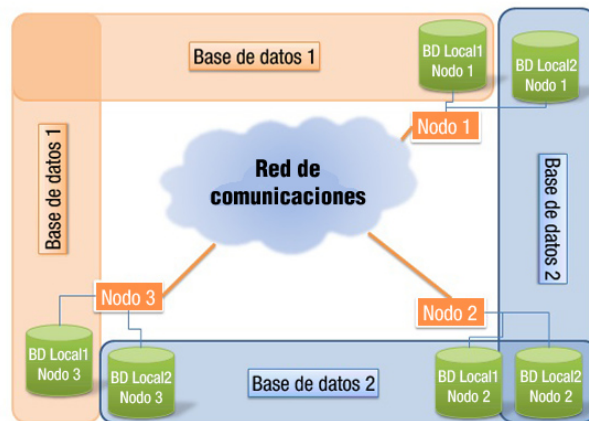


Figura 1.10.1.: Base de datos distribuida

Así, un SGBDD desarrollará su trabajo a través de un conjunto de nodos, que poseen un sistema de procesamiento de datos completo con un base de datos local, un sistema gestor de bases de datos e interconectados entre sí. Si estos nodos están dispersos geográficamente, se interconectarán mediante una red de área extensa o WAN (Wide Area Network), pero si se encuentran en edificios relativamente cercanos, puede estar conectados por una red de área local o LAN (Local Area Network).

Este tipo de sistemas es utilizado en organizaciones con estructura descentralizada, industrias de manufactura con múltiples sedes, aplicaciones militares, líneas aéreas, servicios bancarios, etc...

Al igual que las BD centralizadas, las distribuidas tiene sus ventajas e inconvenientes, que podemos ver en la siguiente figura.

1.10.1 Fragmentación en Bases de Datos Distribuidas

Como hemos visto en el punto anterior, la información en las BDD están repartida en diferentes nodos. La forma de extraer los datos consultados se puede hacer mediante la **fragmentación** de distintas tablas pertenecientes a distintas bases de datos que se encuentran en diferentes servidores. El problema de fragmentación se refiere al particionamiento de la información para distribuir cada parte a los diferentes sitios de la red.

Hay que tener en cuenta el **grado de fragmentación** que habrá, ya que es un factor clave a la hora de ejecutar las consultas. Si no existe fragmentación, se tomarán las relaciones o tablas como unidad de fragmentación. Pero también puede fragmentarse a nivel de tupla (fila o registro) o a nivel de atributo


Ventajas	Inconvenientes
El acceso y procesamiento de los datos es más rápido ya que varios nodos comparten carga de trabajo.	La probabilidad de violaciones de seguridad es creciente si no se toman las precauciones debidas.
Desde una ubicación puede accederse a información alojada en diferentes lugares.	Existe una complejidad añadida que es necesaria para garantizar la coordinación apropiada entre los nodos.
Los costes son inferiores a los de las bases centralizadas.	La inversión inicial es menor, pero el mantenimiento y control puede resultar costoso.
Existe cierta tolerancia a fallos. Mediante la  replicación, si un nodo deja de funcionar el sistema completo no deja de funcionar.	Dado que los datos pueden estar replicados, el control de concurrencia y los mecanismos de recuperación son mucho más complejos que en un sistema centralizado.
El enfoque distribuido de las bases de datos se adapta más naturalmente a la estructura de las organizaciones. Permiten la incorporación de nodos de forma flexible y fácil.	El intercambio de mensajes y el cómputo adicional necesario para conseguir la coordinación entre los distintos nodos constituyen una forma de sobrecarga que no surge en los sistemas centralizados.
Aunque los nodos están interconectados, tienen independencia local.	Dada la complejidad del procesamiento entre nodos es difícil asegurar la corrección de los algoritmos, el funcionamiento correcto durante un fallo o la recuperación.

Figura 1.10.2.: Ventajas e Inconvenientes de las bases de datos distribuidas

de una tabla. No será adecuado un grado de fragmentación nulo, ni tampoco un grado de fragmentación demasiado alto. El grado de fragmentación deberá estar equilibrado y responder a las particularidades de las aplicaciones que utilicen dicha base de datos.

Cuando se lleva a cabo la fragmentación, existen **tres reglas fundamentales** que se deben seguir. A saber:

- **Completitud:** si una relación R se descompone en fragmentos R_1, R_2, \dots, R_n , cada elemento de datos que pueda encontrarse en R , deberá encontrarse en uno o varios fragmentos de R_i .
- **Reconstrucción:** si una relación R se descompone en fragmentos R_1, R_2, \dots, R_n , la reconstrucción de la relación a través de sus fragmentos asegura que se preservan las restricciones definidas sobre los datos.
- **Disyunción:** si una relación R se descompone verticalmente, sus atributos primarios clave se repiten en todos sus fragmentos.

Así, cada vez que vayamos a diseñar la fragmentación de la información en una BDD, deberemos tener en cuenta estas tres reglas y asegurarnos de que nuestro diseño las cumpla.

Respecto a los **tipos de fragmentación**, podemos encontrarnos con tres tipos principales:

- **Fragmentación Horizontal:** este tipo de fragmentación se realiza sobre las tuplas de la relación, dividiendo la relación en subrelaciones que contienen un subconjunto de las tuplas que alberga la primera. Dentro de la fragmentación horizontal podemos encontrar la **primaria** y la **secundaria**.
- **Fragmentación Vertical:** la fragmentación vertical se basa en los atributos de la relación para efectuar la división. Una relación R produce fragmentos R_1, R_2, \dots, R_n , cada uno de los cuales contiene un subconjunto de atributos de R así como la clave primaria de R . El objetivo de la fragmentación es particionar una relación en un conjunto de relaciones mas pequeñas de forma que varias aplicaciones de usuario se ejecutarán sobre un fragmento. En este contexto, una fragmentación óptima es aquella que produce un esquema de fragmentación que minimiza el tiempo de ejecución de las consultas de usuario. La fragmentación vertical es más complicada que la horizontal, ya que existe un número mayor de alternativas para realizarla.
- **Fragmentación Híbrida o Mixta:** podemos combinar ambas aproximaciones utilizando la denominada fragmentación mixta. Si tras una fragmentación horizontal se lleva a cabo una vertical, se habla de fragmentación mixta Horizontal-Vertical. Para el caso contrario, estaremos ante una

fragmentación Vertical-Horizontal. Estos dos tipos de fragmentación se representan mediante arboles.

1.11 Enlaces de Ampliación

En este tema hemos visto los conceptos básicos sobre el almacenamiento de la información, centrándonos en los conceptos de ficheros y sus características, en que es una base de datos y las diferentes modelos de organización que se suelen emplear, viendo por último en que consiste un sistema gestos de bases de datos y que tipos existen.

A continuación, mostramos una lista con enlaces para ampliar y profundizar más en esta información, principalmente en los aspectos relacionados con las bases de datos y los SGBD.

- SGBD MariaDB
<https://es.wikipedia.org/wiki/MariaDB>
- SGBD MongoDB
<https://es.wikipedia.org/wiki/MongoDB>
- SGBD PostgreSQL
<https://es.wikipedia.org/wiki/PostgreSQL>
- Formato JSON para el intercambio de datos
<https://es.wikipedia.org/wiki/JSON>
- Formato BSON para el intercambio de datos
<https://es.wikipedia.org/wiki/BSON>
- Información sobre sistemas de gestión tipo NoSQL
<https://es.wikipedia.org/wiki/NoSQL>

Además de estos enlaces, podemos encontrar muchísima más información sobre los diferentes SGBD en sus correspondientes páginas oficiales, que podemos encontrar con una búsqueda rápida en cualquier buscador web, ya que la mayoría contienen una extensa documentación.

Tema 2

Unidad 2: Bases de Datos Relacionales - TODO

Tema 3

Unidad 3: Implantación de Bases de Datos Relacionales - TODO

Tema 4

Unidad 4: Realización de Consultas

En esta unidad vamos a ver como realizar consultas a una base de datos usando **SQL**. Usaremos como bases de datos para realizar la consultas **MySQL** y **Oracle**, por ello hay que tener en cuenta que, aunque hay **aspectos comunes** de SQL a ambos sistemas de gestión de bases de datos, también hay aspectos específicos, en cuyo caso se indicará explícitamente.

4.1 Introducción

El lenguaje **SQL** nació a partir de la publicación “A relational model of data for large shared banks”, publicado por **Edgar Frank Codd**. IBM aprovecho el modelo propuesto por Codd para crear un lenguaje acorde al recién creado **modelo relacional**, llamándolo **SEQUEL** (Structured English Query Language). Con el tiempo, SEQUEL se convertiría en SQL (Structured Query Language). En 1979, la empresa **Relational Software** lanzaría la primera versión comercial de SQL. Esta empresa, la conocemos hoy en día como **Oracle**.

En 1992, **ANSI** e **ISO** completaron la estandarización de SQL y se definieron las sentencias básicas que debía contener SQL para que fuera estándar. A este se le conocería como ANSI-SQL o SQL92.

Hoy en día, SQL sigue siendo el lenguaje estándar de acceso a bases de datos relacionales. La mayoría de bases de datos cumplen con el estándar, aunque cada fabricante añade sus mejoras al lenguaje SQL.

La **primera fase de trabajo** con cualquier base de datos consiste en la utilización de un lenguaje **DDL** (Data Defining Language), que nos permitirá crear la bases de datos y sus tablas, algo necesario antes de comenzar a trabajar con los datos que queremos alojar en esta. Esta fase la desarrollamos en la unidad anterior.

La **siguiente fase** será manipular los datos, trabajando con sentencias **DML** (Data Management Language). Este lenguaje esta orientado a consultas y manejo de datos de los datos creados. Las principales sentencias que nos proporciona son **SELECT**, **INSERT**, **DELETE** y **UPDATE**.

En esta unidad nos centraremos en las consultas, es decir, en la sentencia **SELECT**.

4.1.1 En Oracle

Las sentencias que veremos a continuación pueden ser ejecutadas desde el entorno web **Application Express** de Oracle utilizando el botón **SQL Workshop** en la página de inicio o home y eligiendo la opción **SQL Commands**.

También se pueden indicar las sentencias desde el entorno **SQL*Plus** que ofrece Oracle y que puede encontrarse en **Inicio >Todos los Programas >Oracle Database 11g Express Edition >Run SQL Command Line**, en el menú inicio de Windows.

Si optas por usar esta última aplicación, el primera paso que debes realizar para poder manipular los datos de una determinada tabla, es conectarse utilizando un nombre de usuario con los permisos necesarios para acceder a ese tipo de operaciones en la tabla deseada. Utiliza para ello la orden **CONNECT** seguida del nombre de usuario. Posteriormente solicitará la contraseña de usuario correspondiente.

Para ejecutar cualquier de las sentencias SQL que aprenderás en los siguiente puntos, simplemente debes **escribirla completa** y pulsar **Intro** para que se inicie su ejecución.

4.1.2 MySQL

En el caso de **MySQL**, las sentencias que veremos a continuación pueden ser ejecutadas:

- Desde cualquier **cliente gráfico** como por ejemplo **MySQL Workbench**. En este caso, redactaremos la sentencia en la ventana **SQL** y la ejecutaremos accediendo a **Query >Execute Current Statement**, o mediante el botón con un símbolo de un **rayo amarillo**.
- Desde la **línea de comandos** de MySQL.

En ambos casos, lo primero que habrá que hacer es iniciar o arrancar el servidor de bases de datos MySQL, si no esta iniciado por defecto, y conectar o abrir una conexión con el servidor utilizando un nombre de usuario con los permisos necesarios para hacer este tipo de operaciones sobre la bases de datos en la que queremos trabajar.

4.1.3 Modelo de la Base de Datos de Ejemplo

Aquí vamos a exponer los diferentes diagramas de la base de datos que vamos a usar como ejemplo en este tema, tanto para Oracle como para MySQL.

- **Modelo para Oracle:** en primero lugar vamos a mostrar el esquema entidad-relación de la base de datos con la que vamos a trabajar en Oracle.

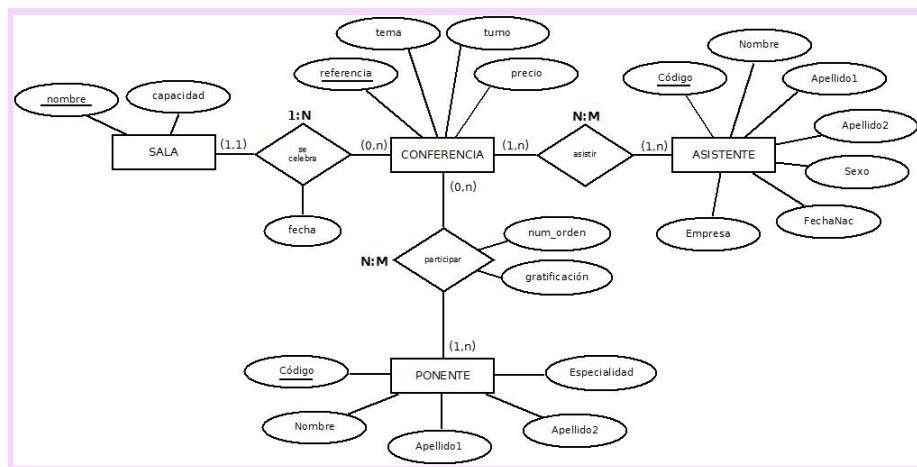


Figura 4.1.1.: Esquema entidad relación para la base de datos Oracle

El **modelo relacional** de paso de tablas resultante del anterior esquema sería el siguiente:

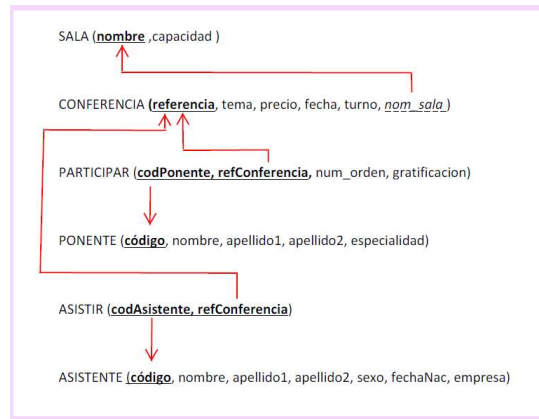


Figura 4.1.2.: Esquema de paso a tablas para Oracle

- **Modelo para MySQL:** para MySQL vamos a emplear otra base de datos. En primer lugar, aquí tenemos el **grafo relacional** para el paso de tablas, en modo texto, de esta base de datos.

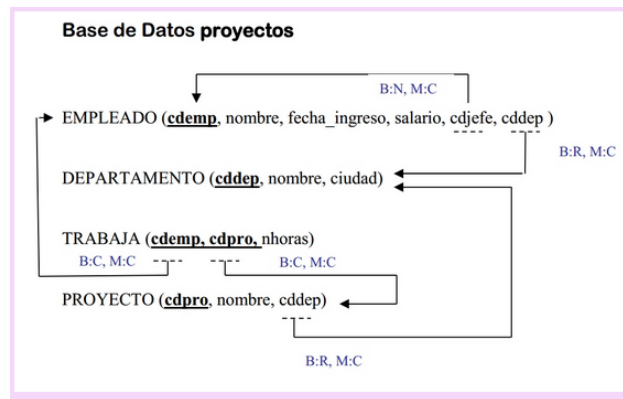


Figura 4.1.3.: Grafo relacional de la base de datos para MySQL

A partir de este modelo, se ha creado uno dentro de la aplicación **WorkBench**, el cual podemos ver en la siguiente imagen:

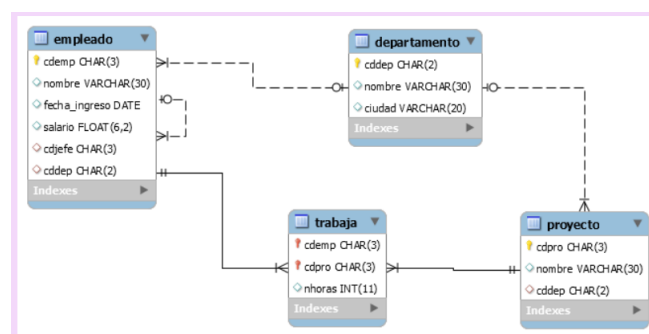


Figura 4.1.4.: Esquema relacional en Workbench para MySQL

Además os recomendamos que consultéis el apéndice **Anexos Tema 4** donde se explica más detalladamente como crear las bases de datos, tanto en Oracle como MySQL, que vamos a emplear en esta

unidad.

4.2 La Sentencia SELECT

En esta unidad, como hemos comentado, vamos a estudiar la realización de **consultas** a una base de datos. El lenguaje SQL nos proporciona la sentencia **SELECT** para realizar dichas consultas, permitiéndonos recuperar o seleccionar datos de una o varias tablas.

Esta sentencia consta de 4 partes básicas:

- **Cláusula SELECT**, seguida de la descripción de lo que se desea ver, es decir, de los **nombres de la columnas** que quieres que se muestren, separados por comas simples. Esta parte es **obligatoria**.
- **Cláusula FROM**, seguida del **nombre de la tabla o tablas** de las que queremos consultar las columnas, es decir, de donde quieres extraer los datos. Esta parte también es **obligatoria**.
- **Cláusula WHERE**, seguida de un **criterio de selección o condición**. Esta parte es **opcional**.
- **Cláusula ORDER BY**, seguida por un criterio de ordenación. Esta cláusula también es **opcional**.

Por lo tanto, una sentencia SELECT completa quedaría de la siguiente forma:

```
SELECT [ALL | DISTINCT] columna1, columna2, ... FROM tabla1, tabla2, ... WHERE condición1, ... ORDER BY ordenación;
```

Las cláusulas **ALL** y **DISTINCT** son opcionales, y su funcionalidad es la siguiente:

- **ALL**: esta cláusula indica que se quieren seleccionar todas las filas, **estén o no repetidas**. Es el **valor por defecto** y no se suele especificar.
- **DISTINCT**: esta cláusula indica que las filas repetidas se deben suprimir del resultado.

En los siguientes puntos, veremos más a fondo cada una de las cláusulas que acabamos de ver.

4.2.1 Cláusula SELECT

Esta cláusula es la principal de la sentencia **SELECT**, que nos permitirá seleccionar la información que queremos obtener. Además de ser obligatoria, debemos tener en cuenta lo siguiente:

- **Se pueden nombrar** a las columnas anteponiendo el nombre de la tabla de la que proceden, aunque esto es opcional, el resultado sería así: **nombretabla.nombrecolumna**.
- Si queremos **incluir todas las columnas** podemos usar el comodín **asterisco (*)**. Por ejemplo: **SELECT * FROM nombretabla**.
- También podemos **ponerle alias a los nombres** de las columnas. Cuando se consulta una base de datos, los nombres de las columnas se usan como cabeceras de presentación. Si este resulta largo, corto o poco descriptivo, podemos usar un alias. Para ello, ponemos a continuación del nombre de la columna, el alias que le demos a esta entre comillas. Por ejemplo:
 - **Oracle**:

```
SELECT nombre "Nombre de la Sala" FROM SALA;
```

- **MySQL:**

```
SELECT nombre "Nombre de la Sala" FROM SALA;  
  
SELECT nombre as "Nombre empleado" FROM empleado;
```

- También podemos **sustituir** el nombre de las columnas por **constantes, expresiones o funciones SQL**, como vemos en los siguientes ejemplos.

- **Oracle:**

```
SELECT 4*3/100 "MiExpresión", capacidad FROM SALA;
```

- **MySQL:**

```
SELECT 4*3/100 "MiExpresión", nombre FROM empleado;
```

4.2.2 Cláusula FROM

Al realizar una sentencia SELECT debemos especificar de donde vamos a extraer los datos, es decir, de que tablas vamos a recuperar la información, para ello disponemos de la cláusula **FROM**.

En esta cláusula es donde se **definen** los **nombres de las tablas**. Si aparecen más de una tabla, éstas irán separadas por comas. A este tipo de consulta se denomina **consulta combinada** o **join**. Más adelante veremos que para que este tipo de consultas puedan realizarse deberemos aplicar una condición de combinación a través de la cláusula **WHERE**.

En el caso de **Oracle**, también puedes añadir el nombre de usuario del **propietario** de estas tablas, indicándolo así **USUARIO.TABLA**, pudiendo así distinguir entre tablas de un usuario y otro.

Tanto en **Oracle** como en **MySQL** pueden utilizarse un alias en las tablas para abreviar, en este caso, **no es necesario** encerrar el alias **entre comillas**, como en el caso de SELECT.

- **Oracle:**

```
SELECT * FROM CONFERENCIA C;
```

- **MySQL:**

```
SELECT * FROM empleado e;
```

4.2.3 Cláusula WHERE

Hasta ahora hemos podido usar la sentencia **SELECT** para obtener todas o un subconjunto de columnas de una o varias tablas, pero esta selección afectaba a todas las filas de la tabla. Si queremos restringir esta selección a un subconjunto de filas, debemos especificar alguna condición que deban cumplir aquellos registros que queremos seleccionar. Para esto, debemos emplear la cláusula **WHERE**.

A continuación de la palabra **WHERE** será donde pongamos la condición que han de cumplir las filas para salir como resultados de dicha consulta.

El criterio de búsqueda o condición puede ser más o menos sencillo y para crearlo se pueden combinar operadores de diversos tipos, funciones o expresiones más o menos complejas.

- **Oracle:**

```
SELECT nombre, apellido1, apellido2 FROM ASISTENTE WHERE sexo = 'H';
```

- **MySQL:**

```
SELECT nombre FROM empleado WHERE salario < 2000;
```

4.2.4 Cláusula HAVING

En el apartado anterior veíamos las funciones de **WHERE**, el problema es que si en la condición queremos incluir operadores aritméticos, como **SUM**, **MAX**, **COUNT**..., ya no sería posible utilizarlo. Aquí es cuando viene en nuestra ayuda **HAVING**.

Si cuando hacemos una consulta queremos restringir la selección a un subconjunto de filas especificando una condición aritmética debemos usar la cláusula **HAVING**.

Por lo tanto, si queremos saber cuando debemos usar **HAVING** o **WHERE** debemos tener en cuenta esta condición, además de que **WHERE** se usa para operar sobre **registros individuales** mientras **HAVING** se usa para operar sobre conjuntos de registros.

- **Oracle:** si en nuestra tabla **ASISTENTE**, necesitáramos un listado de los asistentes a más de dos películas, bastaría con crear la siguiente consulta.

```
SELECT nombre, apellido1, apellido2  
FROM ASISTENTE  
GROUP BY nombre  
HAVING count(peliculas) > 2;
```

- **MySQL:** si en nuestra tabla **empleado**, necesitáramos un listado de los empleados con un salario inferior a 200€, bastaría con crear la siguiente consulta:

```
SELECT nombre
FROM empleado
GROUP BY nombre
HAVING AVG(salario) < 200;
```

Normalmente **HAVING** suele ir acompañado de **GROUP BY**, ya que éste opera sobre los grupos que devuelve la la sentencia **GROUP BY**.

4.2.5 Cláusula **ORDE BY**

En la consulta del punto anterior hemos obtenido una lista de nombres y apellidos de los empleados. Sería conveniente que quedara ordenador por apellidos ya que es más práctico. Para ello, vamos a usar la cláusula **ORDER BY**.

Con esta cláusula, podemos **ordenar** la respuesta que recibimos a nuestra consulta especificando un **criterio de ordenación**. Su sintaxis sería la siguiente:

```
SELECT [ALL | DISTINCT] columnal, columna2, ...
FROM tabla1, tabla2, ...
WHERE condición1, condición2, ...
ORDER BY columnal [ASC | DESC], columna2 [ASC | DESC], ..., columnaN [ASC | DESC];
```

Después de cada columna de ordenación se puede incluir el **tipo de ordenación** (ascendente o descendente) utilizando las palabras reservadas **ASC** o **DESC**. Por defecto, la ordenación es de tipo ascendente.

Debes saber que es posible **ordenar** por **mas de una columna**. Es más, puedes ordenar no solo por columnas sino a través de una expresión creada con columnas, una constante o funciones SQL.

También se puede referenciar el campo por su número de orden en lugar de su nombre, es decir, referenciarlo por su posición en la lista de selección.

■ Oracle:

```
SELECT nombre, apellido1, apellido2,
FROM ASISTENTE
ORDER BY apellido1, apellido2, nombre;

-- O bien seleccionandolo por el número de campo

SELECT nombre, apellido1, apellido2, empresa FROM ASISTENTE ORDER BY 4;
```

■ MySQL:

```
SELECT nombre, cddep FROM empleado ORDER BY cddep, nombre;S

SELECT nombre, cddep, salario FROM empleado ORDER BY 3;
```

Hay que tener en cuenta que si colocamos un **número mayor que la cantidad de campos** de la lista de selección, aparecerá un **mensaje de error** y la consulta no se ejecutará. Además, no todos los **tipos de datos** sirven para ordenar, solo servirán aquellos de tipo **carácter, número y fecha**.

4.2.6 Tipos de JOIN

La palabra **JOIN** significa unir, y precisamente ese es su uso en SQL. Esta sentencia se usa para **combinar filas** de distintas tablas con campos comunes entre ellas. Hay diferentes tipos de **JOIN**, los cuales podemos ver en la siguiente figura.

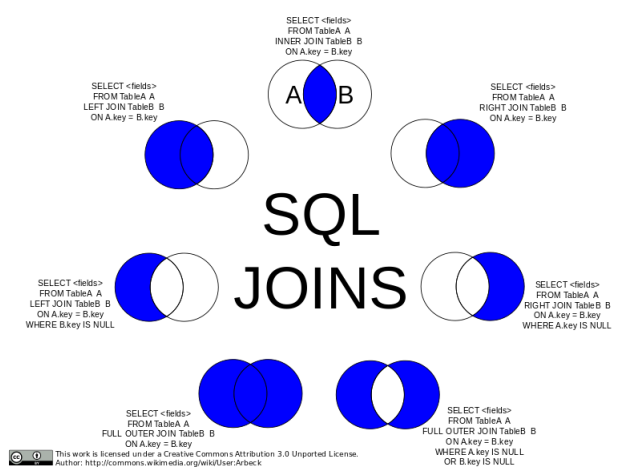


Figura 4.2.1.: Tipos de JOIN

A continuación vamos a explicar un poco cada tipo y a poner un ejemplo de cada uno de ellos.

- **INNER JOIN:** devuelve como resultado la selección de las distintas filas que tengan columnas en ambas tablas. Por ejemplo, si tenemos dos tablas, Empleados y Departamentos, y queremos listar los departamentos al cual pertenece cada empleado, la sentencia sería la siguiente:

```
SELECT de.nombre, e.nombre
FROM departamento de
INNER JOIN empleados e
ON de.id = e.departamento;
```

- **RIGHT JOIN:** devuelve como resultado todas las filas de la tabla de la derecha más las coincidencias con la tabla de la izquierda. Por ejemplo, si tenemos dos tablas, Empleados y Proyectos, vamos a listar los empleados que están asignados a algún proyecto.

```
SELECT p.nombre, e.nombre FROM proyectos p RIGHT JOIN empleados e ON p.id = e.proyecto;
```

- **LEFT JOIN:** devuelve como resultado todas las filas de la tabla izquierda más las coincidencias con la tabla de la derecha. Por ejemplo, si tenemos dos tablas Empleados y Proyectos, vamos a

listar los empleados y aparecerán los proyectos que están asignados a algún empleado:

```
SELECT p.nombre, e.nombre FROM proyectos p LEFT JOIN empleados e ON p.id = e.proyecto;
```

- **OUTER JOIN** o **FULL OUTER JOIN**: devuelve como resultado todas las filas de la tabla de la derecha y todas las filas de la tabla de la izquierda. Aparecerá **NULL** cuando no haya una coincidencia. Por ejemplo, si tenemos dos tablas Empleados y Proyectos, vamos a listar todos los proyectos y todos los empleados:

```
SELECT p.nombre, e.nombre FROM proyectos p OUTER JOIN empleados e ON p.id = e.proyecto;
```

La opción **OUTER JOIN** no existe en **MySQL**, por lo que cuando queramos usar una sentencia similar deberemos usar la cláusula **UNION**, entre dos consultas de tipo **LEFT JOIN** y **RIGHT JOIN**, como se muestra a continuación.

```
SELECT p.nombre, e.nombre FROM proyectos p LEFT JOIN empleados e ON p.id = e.proyecto  
  
UNION  
  
SELECT p.nombre, e.nombre FROM proyectos p RIGHT JOIN empleados e ON p.id = e.proyecto;
```

4.3 Operadores

Como hemos visto en el punto anterior, con la cláusula **WHERE** podíamos incluir expresiones para filtrar el conjunto de datos que queríamos consultar. Para crear estas expresiones, necesitamos usar diferentes operadores de modo que se puedan comparar, utilizar la lógica o elegir en función de la suma, etc...

Los **operadores** son símbolos que permiten la realización de operaciones matemáticas, lógicas, concatenar cadenas..., entre otras cosas. **Oracle** reconoce 4 tipos diferentes de operadores. A saber:

1. **Relacionales** o de **comparación**.
2. **Aritméticos**.
3. De **concatenación**.
4. **Lógicos**.

En los siguientes apartados vamos a ver cuales son cada uno de los operadores dentro de estas categorías, para que sirven y como se usan.

4.3.1 Operadores Relacionales

Los **operadores relacionales** nos permitirán **comparar expresiones**, que pueden ser valores concretos de campo, variables, etc...

Son símbolos que se usan para comparar dos valores. Si el **resultado** de la comparación es **correcto**, la **expresión** considerada es **verdadera**, en **caso contrario**, es **falsa**.

En la siguiente figura, puedes ver una tabla con todos los operadores relacionales que podemos usar en SQL y su significado.

OPERADOR	SIGNIFICADO
=	Igualdad.
!=, <, >, ^=	Desigualdad. (En el caso de MySQL, sólo != y < >)
<	Menor que.
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
IN	Igual que cualquiera de los miembros entre paréntesis. <code>IN (miembro1, miembro2,...)</code>
NOT IN	Distinto que cualquiera de los miembros entre paréntesis. <code>NOT IN (miembro1, miembro2,...)</code>
BETWEEN	Entre. Contenido dentro del rango, incluidos valorMin y valorMax. <code>BETWEEN valorMin AND valorMax</code>
NOT BETWEEN	Fuera del rango. <code>NOT BETWEEN valorMin AND valorMax</code>
LIKE '_abc%'	Se utiliza sobre todo con textos y permite obtener columnas cuyo valor en un campo cumpla una condición textual. Utiliza una cadena que puede contener los símbolos "%" que sustituye a un conjunto de caracteres o "_" que sustituye a un carácter.
IS NULL	Devuelve verdadero si el valor del campo de la fila que examina es nulo.

Figura 4.3.1.: Tabla de operadores relacionales

El **valor NULL** significa valor inexistente o desconocido y por tanto es tratado de forma diferente al resto de valores. Si queremos verificar que un valor es **NULL** los operadores que acabamos de ver no serán válidos. Debemos usar el operador **IS NULL** o **IS NOT NULL** como se indica en la tabla, que nos devolverá verdadero si el valor es NULL o no respectivamente.

Además, cuando se usa **ORDER BY**, los valores se NULL se muestran en primer lugar si se usa el modo ascendente o el último si se usa el descendente.

A continuación, vemos dos ejemplo del uso de operadores tanto en Oracle como MySQL.

- **Oracle:** si queremos obtener el nombre completo de los ponentes cuyo código comienza por ESP ordenados alfabéticamente por el primer apellido seria:

```
SELECT nombre, apellido1, apellido2 FROM PONENTE WHERE codigo LIKE  
'ESP%' ORDER BY apellido1;
```

- **MySQL:** si queremos obtener aquellos empleados cuyo salario es superior a 1000€ podemos crear la siguiente consulta:


```
SELECT nombre, salario FROM empleado WHERE salario > 1000;
```

4.3.2 Operadores Aritméticos

Los **operadores aritméticos** son aquellos que nos permiten realizar cálculos matemáticos con valores numéricos. Con estos operadores, **es posible obtener** salidas en la cuales **una columna** sea el **resultado de un cálculo** y no un campo de una tabla

En la siguiente tabla vemos los principales operadores aritméticos.

OPERADOR	SIGNIFICADO
+	Suma
-	Resta
*	Multiplicación
/	División

Figura 4.3.2.: Operadores aritméticos en SQL

Ejemplo de usos de operadores aritméticos son los siguientes:

- **Oracle:** si queremos obtener el precio de una conferencia aumentado en un 5 % en aquellas conferencia inferiores a 15€, podemos realizar la siguiente consulta:

```
SELECT precio*1,05 FROM CONFERENCIA WHERE precio<15;
```

- **MySQL:** si queremos obtener el salario aumentado un 5 % de aquellos trabajadores que cobran menos de 2000€, la consulta será la siguiente:

```
SELECT salario*1.05 FROM empleado WHERE salario<=2000;
```

Hay que tener en cuenta que cuando una expresión se calcula sobre un valor **NULL** el resultado es el propio valor **NULL**.

Además de estos operadores, en **Oracle** tenemos el operador '||', que nos permite **concatenar cadenas de caracteres**. Oracle puede, además, convertir automáticamente valores numéricos en cadenas de caracteres para una concatenación. En **MySQL**, en cambio, se usa la función **CONCAT(cadena1, cadena2,...)** para la concatenación de cadenas.

- **Oracle:** si queremos mostrar juntos el primer y segundo apellido de un ponente, podemos usar la siguiente consulta:

```
SELECT nombre, apellido1 || apellido2 FROM PONENTE;

-- Si queremos dejar un espacio entre una apellido y otro...

SELECT nombre, apellido1 || " " || apellido2 FROM PONENTE;
```

- **MySQL:** para obtener en una sola columna el nombre y el código de empleado, utilizamos la función CONCAT de la siguiente manera:

```
SELECT CONCAT(cdemp, " ", nombre) FROM empleado;
```

4.3.3 Operadores Lógicos

Habrán ocasiones en las que tengamos que evaluar más de una expresión o necesitar verificar que se cumple una condición, varias o solo alguna de ellas. Para esto, tenemos los **operadores lógicos**. En la siguiente tabla, podemos ver los diferentes operadores lógicos que podemos usar en SQL:

OPERADOR	SIGNIFICADO
AND	Devuelve verdadero si sus expresiones a derecha e izquierda son ambas verdaderas.
OR	Devuelve verdadero si alguna de sus expresiones a derecha o izquierda son verdaderas.
NOT	Invierte la lógica de la expresión que le precede, si la expresión es verdadera devuelve falsa y si es falsa devuelve verdadera.

Figura 4.3.3.: Operadores lógicos en SQL

A continuación vemos dos ejemplos de uso de los operadores lógicos en Oracle y MySQL.

- **Oracle:** si queremos obtener aquellas salas cuya capacidad esté entre 100 y 300 personas sería:

```
SELECT nombre, capacidad FROM SALA WHERE capacidad>=100 AND capacidad<=300;
```

- **MySQL:** si queremos obtener aquellos empleados cuyo salario sea menor o igual a 2000€ o superior a 2500€.

4.3.4 Precedencias

Con frecuencia utilizaremos la sentencia **SELECT** con expresiones muy extensas y resultará difícil saber que parte de dicha expresión se evaluará primero. Por ello, es conveniente saber el orden de precedencia de los diferentes operadores que hemos visto. Así, esto se evaluarán con el siguiente orden:

1. Se evalúa la multiplicación (*) y la división (/) al mismo nivel.
2. A continuación sumas (+) y restas (-).
3. Concatenación (||) (solo en **Oracle**)
4. Todas las comparaciones (<, >, ...)
5. Los operadores **IS NULL**, **IS NOT NULL**, **LIKE** y **BETWEEN**.
6. **NOT**.
7. **AND**.
8. **OR**.

En el caso de que queramos variar este orden, siempre podemos **usar paréntesis** para agrupar las expresiones que queremos que se evalúen en primer lugar.

4.4 Consultar Calculadas

En ocasiones es interesante realizar operaciones con algunos campos para obtener información derivada de estos.

Por ejemplo, si tenemos un campo precio, nos podría interesar mostrar los precios con el IVA añadido, o si tuviéramos un campo Sueldo, podríamos mostrar este más la suma de una Paga Extra. Estos son ejemplos simples, pero podemos construir expresiones mucho más complejas. Para ello, haremos uso de la **creación de campos calculados**.

Los **operadores aritméticos** se pueden emplear para realizar cálculos en las consultas. Estos **campos calculados** se obtiene haciendo uso de la sentencia **SELECT** poniendo a continuación la expresión que queramos. Este tipo de consultas, **no modifica los valores originales**, de las columnas de la tabla/s en la que se esta realizando dicha consulta, únicamente mostrará una nueva columna con los valores calculados. Por ejemplo:

- **Oracle:** podemos crear una columna con el valor de precio + 10, como podemos ver en la siguiente consulta.

```
SELECT tema, precio, precio + 10 FROM CONFERENCIA;
```

También podríamos asignar un alias a la columna para que nos muestre un nuevo nombre en esta, añadiendo **AS** después de la expresión y el **nombre del alias**.

```
SELECT tema, precio, precio + 10 AS PrecioNuevo FROM CONFERENCIA;
```

- **MySQL:** ahora vamos a crear una nueva columna llamada SalarioNuevo en MySQL:

```
SELECT nombre, salario, salario + 25 AS SalarioNuevo FROM empleado;
```

4.5 Funciones

En casi todos los sistemas de gestiones de datos existen **funciones** que nos ayudan a crear consultas más complejas. Dichas funciones varían según el SGBD. Aquí, veremos en concreto las de **Oracle**.

Las **funciones** son realmente operaciones que se realizan sobre los datos y que realizan un cálculo determinado. Para ello necesitamos uno datos de entrada, llamados **parámetros** en función de los cuales se realizarán los cálculos adecuados. Normalmente, estos parámetros se especifican entre paréntesis.

Las funciones se pueden incluir en la cláusulas **SELECT**, **WHERE** y **ORDER BY**.

A. Anexos Tema 4

A.1 Creación de Tablas y Preparación de Oracle y MySQL

En este apéndice vamos a indicar el proceso para crear las bases de datos que vamos a usar en la unidad 4, tanto en Oracle como en MySQL.

A.1.1 Creación de las Tablas en Oracle

Para la creación de las tablas en el SGBD de Oracle debemos seguir los siguiente pasos:

1. En primer lugar crearemos un espacio de trabajo o Workbench llamado **TAREA** con un usuario llamado **ANA** al que pondremos también la contraseña que queramos. Este usuario será por tanto el administrador de este espacio de trabajo y tendrá permisos para crear tablas.

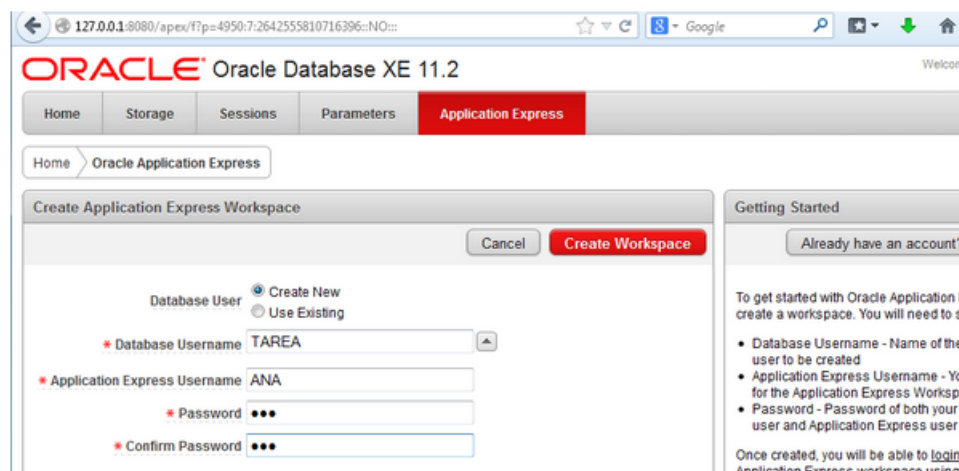


Figura A.1.1.: Creación del espacio de trabajo y el usuario

2. A continuación, vamos a descargar el archivo con el **script sql** para la creación de tablas. El archivo podemos encontrarlo [este enlace](#) de github, y pulsando en **download** lo descargamos.

Una vez descargado, vamos a utilizar la **línea de comandos de SQL** para ejecutar el archivo descargado. Para acceder a la línea de comandos debemos hacer lo siguiente:

- a) Vamos a **Inicio > Todos los Programas > Oracle Database 11g Express Edition**.
- b) Pulsando en **Run SQL Command Line** aparecerá la siguiente pantalla:

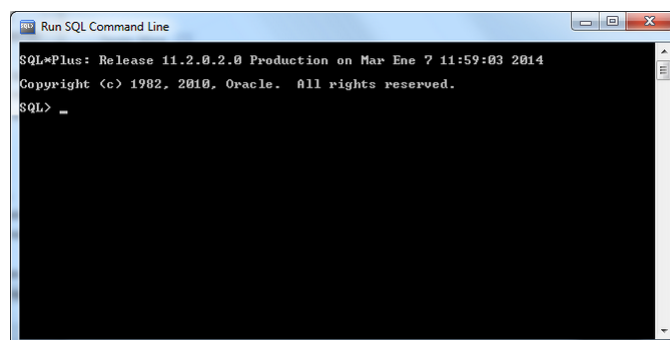


Figura A.1.2.: Línea de comandos de SQL en Oracle

3. Una vez abierta la línea de comandos de SQL, debemos ejecutar la instrucción **connect TAREA\ANA**, que corresponde al espacio de trabajo creado anteriormente con la contraseña del usuario administrador en mayúsculas.

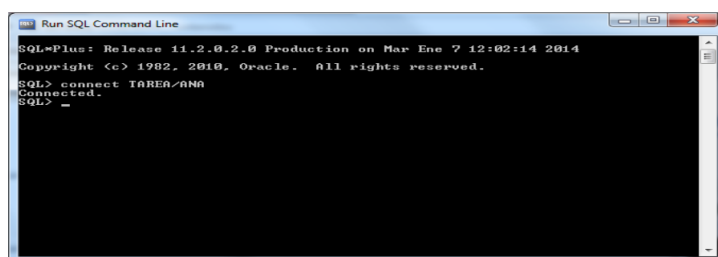


Figura A.1.3.: Conexión al espacio de trabajo creado

Una vez ejecutada la instrucción, si se ha realizado correctamente, se nos mostrará el mensaje **Connected**, indicándonos que se ha realizado la conexión.

4. Una vez realizada la conexión, podemos proceder a ejecutar el archivo descargado anteriormente, indicándolo del de la siguiente manera: **@ruta_archivo/BD04_CONT_R07_02.sql**, como se muestra en la siguiente captura.

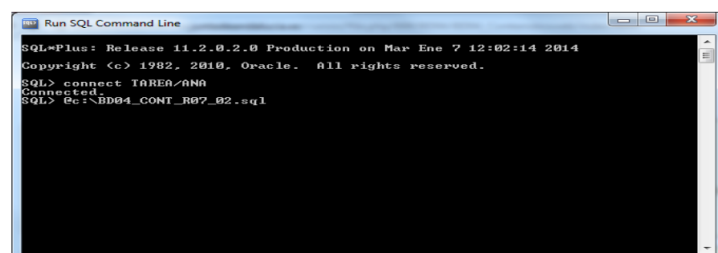


Figura A.1.4.: Ejecución del script proporcionado

Si todo se ha realizado correctamente, la aplicación nos mostrará un mensaje **Table created**, por cada tabla nueva que se haya creado, así como una mensaje **1 row created** por cada dato creado.

5. Por último, ya solo nos queda desconectar de la base de datos para poder consultar las tablas y datos creados desde la aplicación gráfica. Para ello, introducimos el comando **DISCONNECT** y pulsamos **Enter**.

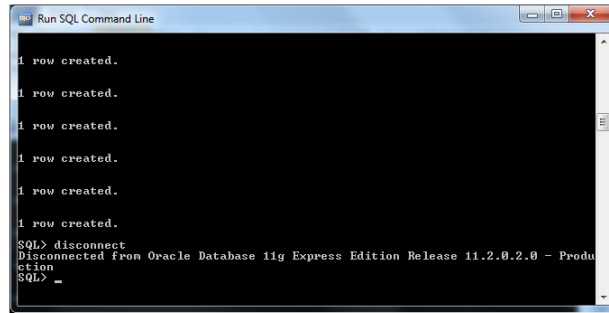


Figura A.1.5.: Desconexión de la base de datos

A.1.2 Creación de las Tablas en MySQL

Ahora vamos a ver como realizar el mismo procedimiento para crear la base de datos necesaria para este tema en MySQL.

1. En primer lugar vamos a descargar el **script** para **generar las tablas** en MySQL, el cual podemos encontrar en [este enlace](#). Una vez descargado el script, iniciamos sesión en nuestro servidor MySQL. Para ello, asegúrate de tener iniciado el servidor de bases de datos MySQL e inicia sesión como el usuario **root**, indicando su contraseña.

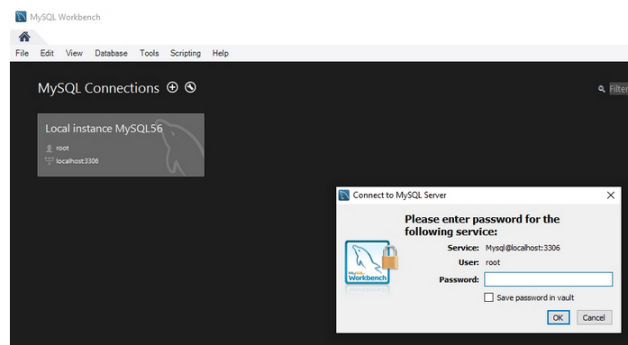


Figura A.1.6.: Inicio de sesión en el servidor MySQL

2. El siguiente paso es cargar el script, **bd_proyectosx.sql**, que hemos descargado anteriormente. Para ello, pulsamos en la opción **File > Open SQL Script...** en el menú de Workbench y buscamos el archivo en nuestras carpetas. Lo seleccionamos y al pulsar **Abrir** el script aparecerá en pantalla.

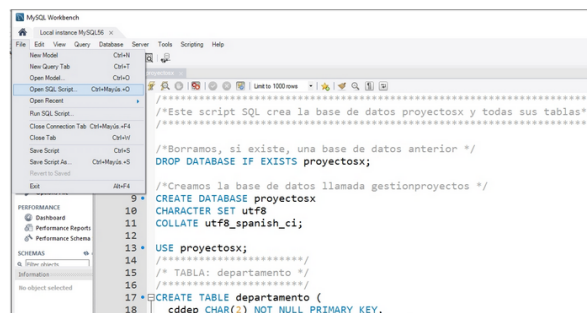


Figura A.1.7.: Carga del script en MySQL Workbench

3. Con el script ya cargado y en pantalla, pulsamos sobre **File >Run SQL Script** o bien en el icono de un “**rayo amarillo**”. Una vez realizado esto, el script se ejecutara y la base de datos quedará implantada.

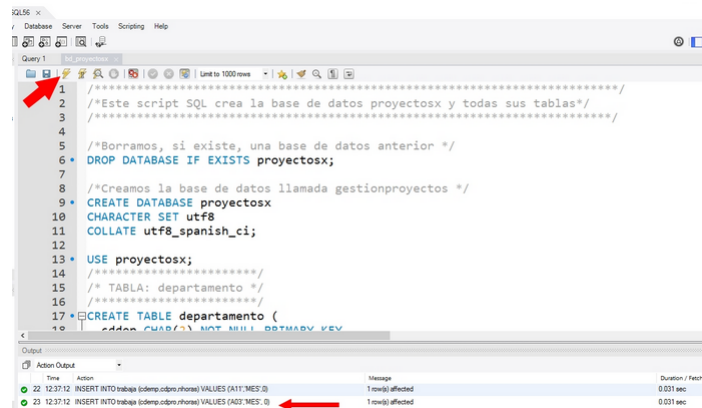


Figura A.1.8.: Ejecución del script y creación de la base de datos

4. Ya solo tenemos que actualizar la información en lista de la izquierda de Workbench pulsando en el **icono de las dos flechas** o pulsando con el botón derecho sobre ese menú y seleccionando **Refresh All**, para que así se nos muestre la nueva base de datos creada.

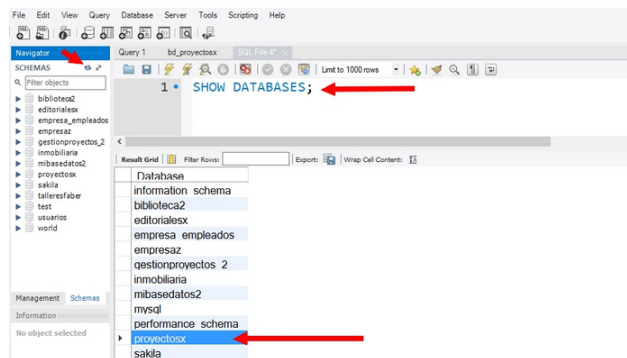


Figura A.1.9.: Actualización de la lista de bases de datos en Workbench

Tras estos pasos, ya tendremos nuestra base de datos creada en el servidor MySQL. Si queremos podemos mostrar las bases de datos que este tiene usando la sentencia **SHOW databases**, o realizar algunas consultas genéricas, como **SELECT * from empleados**, para comprobar los diferentes elementos que tiene cada tabla.

Glosario

ANSI El Instituto Nacional Estadounidense de Estándares, más conocido como ANSI (por sus siglas en inglés: American National Standards Institute), es una organización sin fines de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.. 9, 47

biunívoca Una correspondencia biunívoca, o correspondencia uno-a-uno, es simplemente una correspondencia unívoca cuya correspondencia inversa también es unívoca. En otras palabras, la relación biunívoca se establece cuando para cada elemento del primer conjunto que se corresponde con solo un elemento del segundo conjunto, tal elemento del segundo conjunto se corresponde con solo aquel elemento del primer conjunto.. 12, 47

CODASY Conference on Data System Languages. 17, 47

constantes Cantidad que tiene un valor fijo en un calculo, proceso, etc.... 43, 47

expresiones Combinación de constantes, variables o funciones, que es interpretada (evaluada) de acuerdo a las normas particulares de precedencia y asociación para un lenguaje de programación en particular. Como en matemáticas, la expresión es su "valor evaluado", es decir, la expresión es una representación de ese valor.. 43, 47

funciones Es un grupo de instrucciones con un objetivo en particular y que se ejecuta al ser llamada desde otra función o procedimiento. Una función puede llamarse múltiples veces e incluso llamarse a sí misma (función recurrente). Las funciones pueden recibir datos desde afuera al ser llamadas a través de los parámetros y deben entregar un resultado. Se diferencian de los procedimientos porque estos no devuelven un resultado.. 43, 47

IDMS Integrated Data Management System. 17, 47

inconsistencia Se produce inconsistencia en los datos cuando datos iguales hacen referencia a distintas cosas. Es decir, cuando distintas copias de los mismos datos no coinciden.. 26, 47

integridad Consiste en la veracidad de los datos almacenados con respecto a la información esperada.. 26, 47

OLAP OLAP es el acrónimo en inglés de procesamiento analítico en línea (On-Line Analytical Processing). Su objetivo es agilizar la consulta de grandes cantidades de datos.. 21, 47

redundancia Repetición de un mismo dato dentro de una base de datos.. 26, 47

SQL99 Es la definición estandar del lenguaje de consulta de bases de datos, también denominado SQL-3. Fue creado en 1999.. 21, 47

Unicode El Estándar Unicode es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas además de textos clásicos de lenguas muertas.. 9, 47

Bibliografía

[1] Bases de datos en la nube.

<https://www.oracle.com/es/database/what-is-a-cloud-database/>.