TEMA 1.- ALMACENAMIENTO DE LA INFORMACIÓN

Contenido

TEMA 1.- ALMACENAMIENTO DE LA INFORMACIÓN

1 Dato, Información y Conocimiento

La necesidad de almacenar y procesar datos para obtener información útil ha existido siempre. Para comprender cómo gestionamos estos elementos, primero debemos diferenciarlos claramente.

1.1 Dato

Un dato es la unidad mínima de información sin procesar. Es una representación simbólica (numérica, alfabética, etc.) de un atributo o una variable, que por sí sola no tiene un significado completo ni nos permite tomar decisiones.

- **Ejemplo**: Si pensamos en los datos de los alumnos de un instituto, un dato individual podría ser "Ana", "16", "2º Bachillerato" o "9,5". Cada uno de estos elementos por separado es un dato. Por ejemplo, el número "16" no nos dice nada sin un contexto; podría ser una edad, una nota o un número de clase.

1.2 Información

La información es un conjunto de datos que han sido organizados y procesados de una manera significativa. A diferencia de los datos brutos, la información tiene relevancia y propósito. Constituye un mensaje que cambia el estado de conocimiento de quien lo recibe, permitiendo sacar conclusiones.

- **Ejemplo**: Siguiendo con el caso del instituto, si organizamos los datos anteriores, obtenemos información útil: "La alumna Ana, de 16 años, que cursa 2º de Bachillerato, tiene una nota media de 9,5". Esta frase ya no son datos aislados, sino un mensaje estructurado que nos informa sobre el rendimiento y la situación académica de una alumna específica. De la misma manera, en una oficina, el conjunto de datos sobre un empleado (nombre, nóminas pagadas, vacaciones restantes) se convierte en información cuando se organiza para saber si está al día con sus pagos o cuántos días libres le quedan.

1.3 Conocimiento

El conocimiento es el nivel más elevado y representa la capacidad de transformar los datos, la información y la pericia de las personas en acción. Implica combinar estos elementos con la experiencia acumulada para tomar decisiones y actuar.

- **Ejemplo**: El director del instituto, al analizar la información de que "el 30% de los alumnos de 2º de Bachillerato tiene una nota media inferior a 5 en Matemáticas" (información), puede aplicar su experiencia y pericia (conocimiento) para decidir implementar clases de refuerzo. Esta decisión no se basa solo en los datos, sino en la comprensión profunda del problema y sus posibles soluciones, lo que demuestra la aplicación del conocimiento.

Idea Principal: El proceso es secuencial. Almacenamos **datos** de forma segura, los procesamos para obtener **información** y, finalmente, un experto utiliza esa información, junto con su experiencia, para generar **conocimiento** y tener éxito en sus propósitos.

2 Sistemas Lógicos de Almacenamiento.

2.1 Ficheros

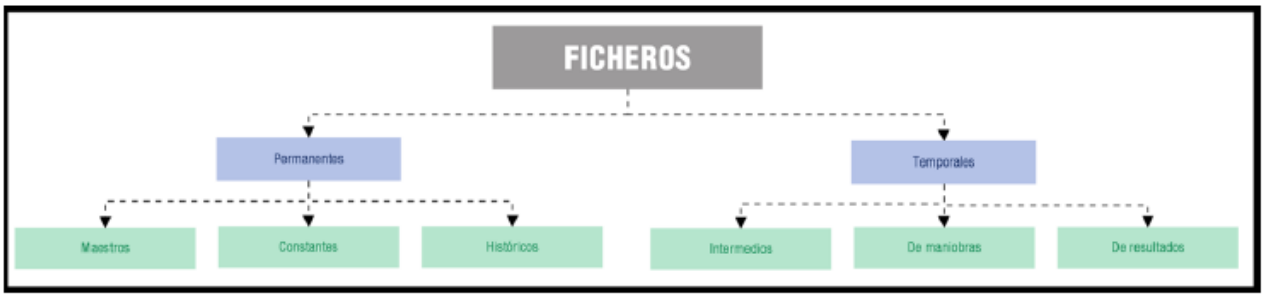
2.1.1 Concepto de Fichero

Un **fichero** o archivo es un conjunto de información relacionada sobre un mismo tema, tratada como una unidad y organizada de forma estructurada para su almacenamiento y posterior recuperación. Los ficheros están formados por **registros lógicos**, que contienen datos de un mismo elemento (como un alumno), y estos a su vez se dividen en **campos**, que son las informaciones elementales (como el nombre o el correo electrónico).

La cantidad de información transferida entre el soporte y la memoria principal en una sola operación se denomina **registro físico o bloque**. Generalmente, un bloque contiene varios registros lógicos, y al número de registros por bloque se le llama **factor de blocaje**.

Sobre los ficheros se realizan operaciones de consulta y mantenimiento (añadir, suprimir, modificar, etc.) a nivel de registro. La organización y el método de acceso son fundamentales para el rendimiento de estas operaciones.

2.1.2 Clasificación de Ficheros según su Función



Los ficheros pueden clasificarse según la función que desempeñan en una aplicación:

- **Ficheros Permanentes**: Contienen información relevante para una aplicación y tienen un largo periodo de permanencia.
    - **Maestros**: Guardan el estado actual de los datos (ej. los datos de los usuarios activos de una plataforma).
    - **Constantes**: Incluyen datos fijos que no suelen modificarse (ej. un archivo con códigos postales).
    - **Históricos**: Contienen datos que ya no son actuales pero se guardan para reconstruir situaciones pasadas (ej. usuarios dados de baja).

- **Ficheros Temporales**: Almacenan información útil solo para una parte de la aplicación y tienen una existencia corta.
    - **Intermedios**: Guardan resultados de una aplicación que serán usados por otra.
    - **De maniobras**: Almacenan datos que no caben en memoria principal.
    - **De resultados**: Contienen datos que se enviarán a un dispositivo de salida.

2.1.3 Soportes de Información

Los ficheros se almacenan en soportes físicos, que determinan el método de acceso.

- **Soportes de Acceso Secuencial**: Como las cintas magnéticas, obligan a leer toda la información anterior para llegar a un dato concreto. Se usan principalmente para copias de seguridad.
- **Soportes de Acceso Directo (o direccionables)**: Como los discos (magnéticos, ópticos), permiten posicionarse directamente en la ubicación del dato deseado. Son los más empleados hoy en día.

2.1.4 Tipos de Ficheros según su Organización y Acceso

La forma en que se organizan los registros en un soporte y cómo se accede a ellos define los principales tipos de ficheros.

2.1.4.1 Ficheros Planos o de Texto Simple

Un **archivo de texto plano** contiene únicamente texto formado por caracteres legibles por humanos, sin ningún tipo de formato tipográfico como negritas o cursivas. Estos archivos son universales y pueden ser leídos por una gran variedad de programas. A menudo se utilizan para almacenar datos de configuración de software, ya que un usuario puede modificarlos fácilmente. Su principal inconveniente es que no hay una forma segura de saber qué codificación de caracteres se utilizó (ej. UTF-8, ISO-8859), lo que puede causar errores. En Windows, suelen tener la extensión .txt.

2.1.4.2 Ficheros Secuenciales

En esta organización, los registros se graban en el soporte de almacenamiento en **posiciones de memoria físicamente contiguas**, uno detrás de otro, sin dejar huecos. Para acceder a un registro, es necesario leer todos los anteriores desde el principio del fichero.

COD	NOMBRE	CIUDAD	
10	Juan	Sevilla	*Registro 1*
35	Carlos	Sevilla	*Registro 2*
16	María	Málaga	*Registro 3*
7	Luís	Jaén	*Registro 4*

- **Ventajas**: Son rápidos para procesar un gran volumen de registros de forma consecutiva y aprovechan al máximo el espacio de almacenamiento.
- **Inconvenientes**: Son muy lentos para consultas de registros individuales y no permiten insertar o eliminar registros de forma intermedia.
- **Ejemplo**: Un fichero con el registro de transacciones bancarias de un día, que se procesará en bloque al final de la jornada.

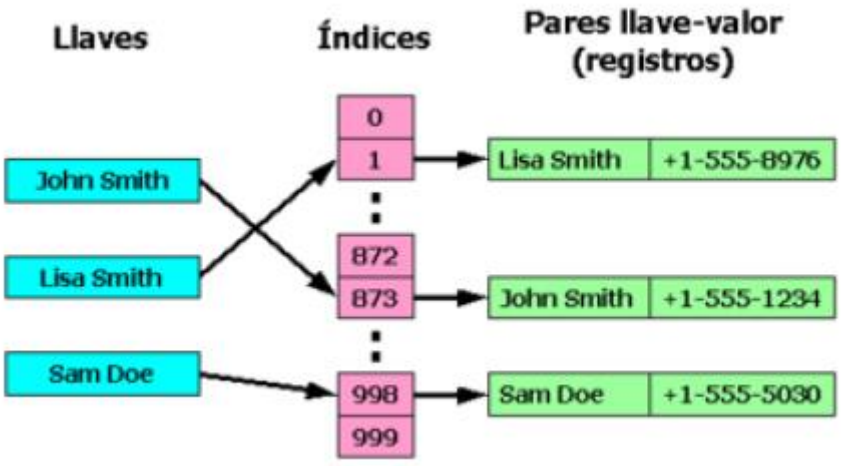
2.1.4.3 Ficheros de Acceso Directo o Aleatorio

Permiten acceder a un registro directamente a través de una **clave** que indica su posición dentro del fichero. Establece una **correspondencia directa entre la dirección lógica (la clave) y la dirección física (dónde está guardado)**. Cada dirección solo puede ser ocupada por un único registro.

- **Ventajas**: Son muy rápidos para tratar registros individuales y permiten operaciones de lectura y escritura simultáneas.
- **Inconvenientes**: Desaprovechan el espacio de almacenamiento al dejar muchos huecos libres y el acceso secuencial es ineficiente. Además, pueden producirse **colisiones** si dos claves intentan ocupar la misma dirección.
- **Ejemplo**: Un sistema de gestión de clientes donde se necesita consultar la ficha de un cliente específico introduciendo su número de identificación (la clave).

2.1.4.3.1 Acceso Calculado o Hash

Esta técnica se emplea en los ficheros de acceso directo. Se utiliza una **función matemática (función de hashing)** que, a partir de la clave del registro, calcula directamente su dirección de almacenamiento.



- **Ventajas**: Es el método más rápido para el acceso directo.

- **Inconvenientes**: El principal problema es la **colisión** o la generación de **sinónimos**, que ocurre cuando la función hash produce la misma dirección para dos claves diferentes. Se necesitan técnicas adicionales para gestionar estas colisiones.
- **Ejemplo**: Un sistema que almacena datos de usuarios usando su número de DNI como clave. Una función hash podría tomar los últimos dígitos del DNI para calcular la dirección donde se guardará el registro del usuario.

2.1.4.4 Ficheros Indexados

Son una solución que combina las ventajas de los modelos secuencial y directo. Un fichero indexado se acompaña de un **índice**, que es una tabla que relaciona una clave con la dirección física del registro correspondiente, permitiendo el acceso directo.

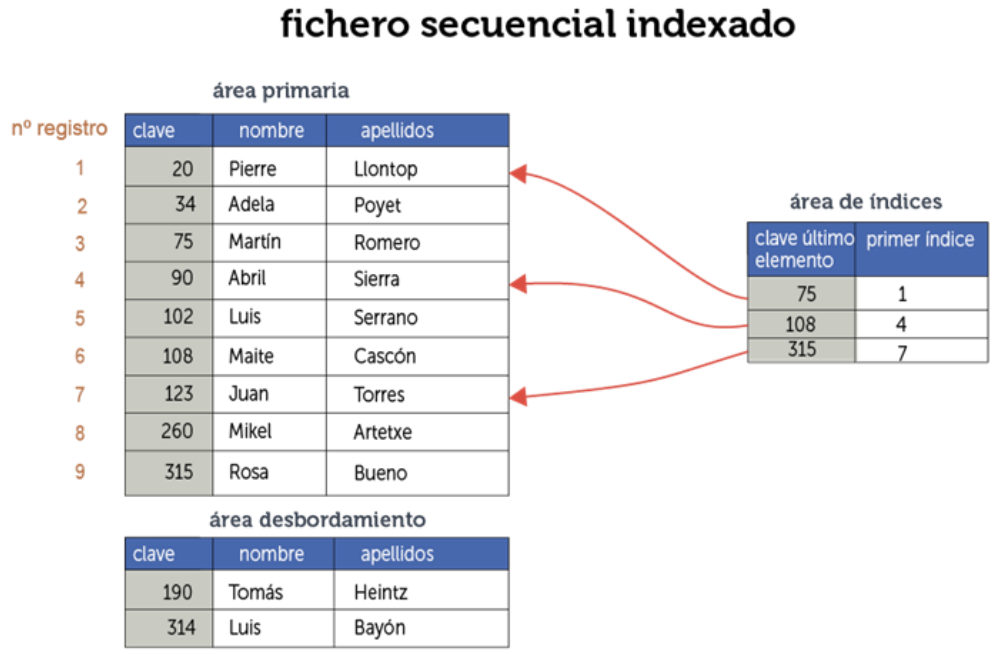
- **Ventajas**: Permiten tanto el acceso secuencial como el directo, y la actualización de registros se puede hacer en el mismo fichero.
- **Inconvenientes**: Ocupan más espacio en disco debido al área de índices y el tiempo de acceso puede degradarse con muchas inserciones, ya que aumenta un área de desbordamiento (*overflow*).
- **Ejemplo**: Un catálogo de una biblioteca. Para buscar un libro por su título (clave), se consulta primero el índice para encontrar su ubicación y acceder directamente a su ficha.

2.1.4.4.1 Estructura de un Fichero Secuencial Indexado (ISAM)

Esta organización es una de las más utilizadas y se basa en tres áreas:

1. **Área Primaria**: Contiene los registros de datos, ordenados por su clave y divididos en segmentos. Dentro de cada segmento, los registros están almacenados secuencialmente.
2. **Área de Índices**: Es un fichero secuencial donde cada registro contiene dos campos: la clave del último registro de un segmento del área primaria y un puntero a la dirección de inicio de dicho segmento.
3. **Área de Excedentes (Overflow)**: Alberga los registros que no caben en el área primaria, por ejemplo, debido a inserciones.

El acceso a un registro se realiza consultando primero el área de índices para localizar el segmento correcto, luego se accede directamente a ese segmento en el área primaria y se busca secuencialmente dentro de él.



2.1.5 Inconvenientes de los sistemas de ficheros

Los sistemas de ficheros, aunque fueron la primera solución, presentan graves desventajas que impulsaron la creación de las bases de datos:

- **Alta Redundancia e Inconsistencia**: El mismo dato puede repetirse en varios ficheros, lo que aumenta el coste de almacenamiento y el riesgo de que los datos se vuelvan contradictorios si no se actualizan en todas partes.
- **Dependencia entre Datos y Programas**: La estructura física de los datos está ligada al programa que los lee. Un cambio en el fichero (ej. añadir un campo) obliga a modificar todos los programas que lo usan.
- **Dificultad de Acceso y Concurrencia**: El acceso a los datos es lento y complicado para consultas complejas, y es muy difícil que varios usuarios modifiquen los datos simultáneamente sin problemas.
- **Problemas de Seguridad**: El control de acceso lo gestiona el programa, no el sistema de ficheros, lo que facilita que alguien con conocimientos pueda acceder o modificar los datos sin autorización.

2.2 Bases de Datos.

Las bases de datos (BD) son el núcleo de los sistemas informáticos modernos y su uso está ampliamente extendido en nuestra vida cotidiana.

2.2.1 Concepto de Base de Datos

Una base de datos es una **colección de datos lógicamente relacionados entre sí, con una definición y descripción comunes, almacenados con la mínima redundancia**. Permite que múltiples aplicaciones y usuarios accedan a ellos de forma eficiente. Una BD no solo contiene los datos, sino también una descripción de los mismos, conocida como **metadatos**, que se almacena en el diccionario de datos.

2.2.2 Ventajas e inconvenientes de las bases de datos

2.2.2.1 Ventajas

- **Independencia de datos y programas**: Se puede modificar la estructura de los datos sin alterar el código de las aplicaciones.
- **Menor redundancia e mayor integridad**: Al centralizar los datos, se evita la duplicación y es más fácil mantener la consistencia.
- **Mayor seguridad**: Permiten un control de acceso centralizado.
- **Acceso más eficiente y concurrente**: Varios usuarios pueden acceder a los datos de forma simultánea y optimizada.

2.2.2.2 Desventajas

- **Coste elevado**: Requieren software y hardware potentes.
- **Personal cualificado**: Necesitan administradores expertos.
- **Implantación compleja**: Su puesta en marcha puede ser un proceso largo y difícil.

2.2.3 Usos de las Bases de Datos

Las bases de datos se utilizan en prácticamente todos los ámbitos:

- **Banca**: Información de clientes, cuentas, transacciones.
- **Líneas aéreas**: Reservas, horarios, información de vuelos.
- **Universidades**: Datos de estudiantes, carreras, materias.
- **Telecomunicaciones**: Registros de llamadas, facturación.
- **Medicina**: Historiales clínicos, gestión hospitalaria.
- **Internet**: Redes sociales, juegos online, bibliotecas virtuales, etc.

2.2.4 Sistemas Gestores de Bases de Datos (SGBD)

Si una base de datos es como una biblioteca, el **Sistema Gestor de Bases de Datos (SGBD)** es el bibliotecario.

- **Definición**: Un SGBD es un **conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos**. Actúa como intermediario entre los usuarios (o las aplicaciones) y la base de datos física.

2.2.5 Funciones de un SGBD

Un SGBD realiza tres funciones principales:

1. **Descripción o Definición (DDL - Data Definition Language)**: Permite crear y definir la estructura de la base de datos (tablas, campos, relaciones, restricciones).
2. **Manipulación (DML - Data Manipulation Language)**: Facilita la inserción, borrado, modificación y consulta de los datos.
3. **Control (DCL - Data Control Language)**: Permite al administrador gestionar la seguridad, los permisos de los usuarios, las copias de seguridad y la concurrencia.

El lenguaje estándar que engloba estas tres funciones es **SQL (Structured Query Language)**.

2.2.6 Componentes de un SGBD

Un SGBD está compuesto por:

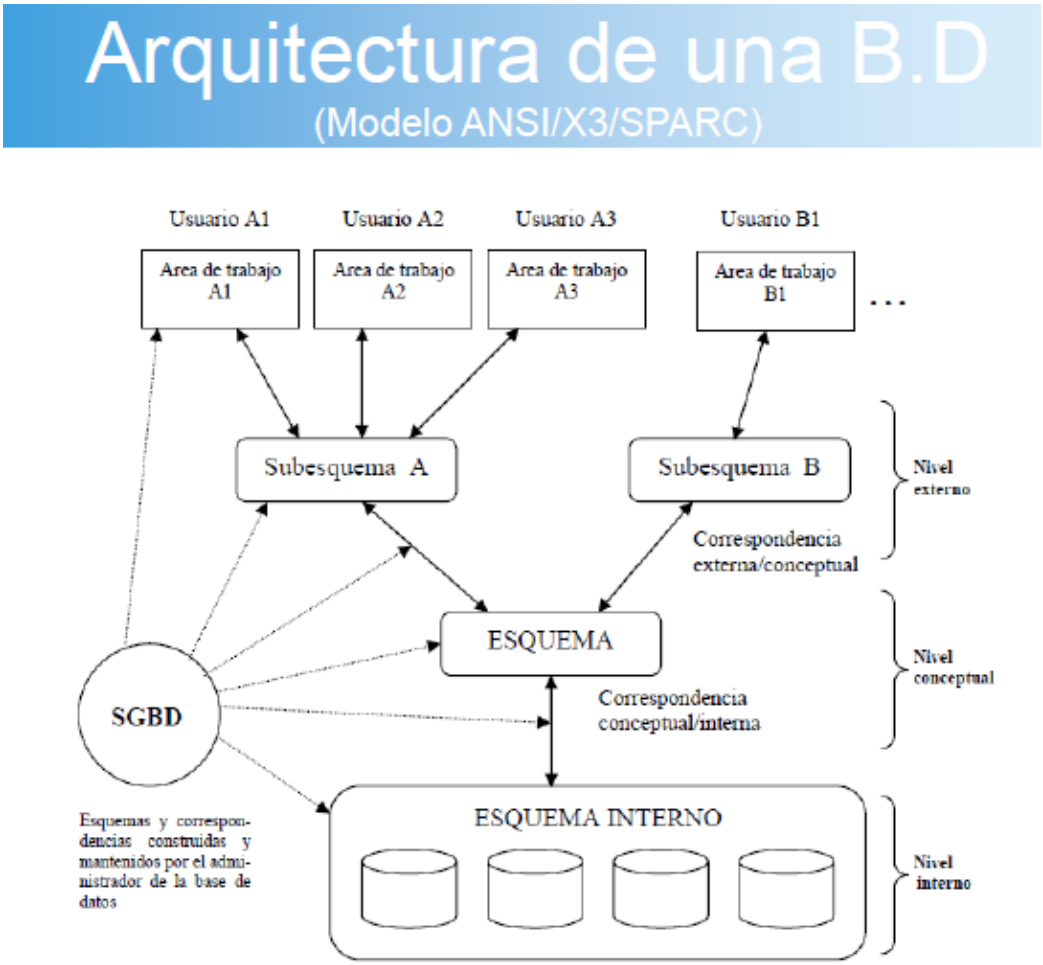
- **Gestor de la Base de Datos**: El software principal que interactúa con los datos y el sistema operativo.
- **Diccionario de Datos**: Un catálogo que almacena los metadatos.
- **Lenguajes de la Base de Datos**: DDL, DML y DCL.
- **Usuarios**: Se distinguen diferentes perfiles, como el **Administrador de la Base de Datos (DBA)**, diseñadores, programadores y usuarios finales.
- **Herramientas**: Aplicaciones adicionales para la gestión, generación de informes, etc..

2.2.7 Arquitectura de un SGBD

Para proporcionar independencia entre los datos físicos y la visión que tienen los usuarios, los SGBD utilizan una **arquitectura de tres niveles**:

1. **Nivel Interno o Físico**: Es el nivel más bajo y describe **cómo se almacenan físicamente los datos** en los dispositivos de almacenamiento. Define ficheros, métodos de acceso, índices, etc..
2. **Nivel Lógico o Conceptual**: Describe la **estructura completa de la base de datos** para toda la comunidad de usuarios. Define las entidades, atributos, relaciones y restricciones de integridad, ocultando los detalles físicos.
3. **Nivel Externo o de Visión**: Es el nivel más alto y describe la **parte de la base de datos que es relevante para un usuario o grupo de usuarios en particular**. Cada usuario tiene su propia "vista" de los datos.

Esta arquitectura permite la **independencia lógica** y la **independencia física**.



2.2.8 Tipos de SGBD

Los SGBD se pueden clasificar según varios criterios:

- **Según el modelo lógico**: Relacionales, Orientados a Objetos, etc..
- **Según el número de usuarios**: Monousuario o multiusuario.
- **Según la distribución**: Centralizados o distribuidos.
- **Según el coste y la licencia**: Comerciales (privativos, requieren pago de licencia) o libres (open source).

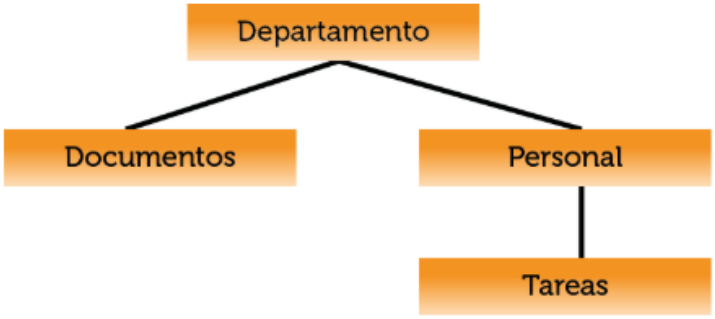
2.2.8.1 SGBD según su modelo lógico

Los principales modelos son:

2.2.8.1.1 Modelo Jerárquico

Este es uno de los modelos más antiguos. Organiza los datos siguiendo una **estructura de árbol**, donde los datos se relacionan en una jerarquía de "padre-hijo".

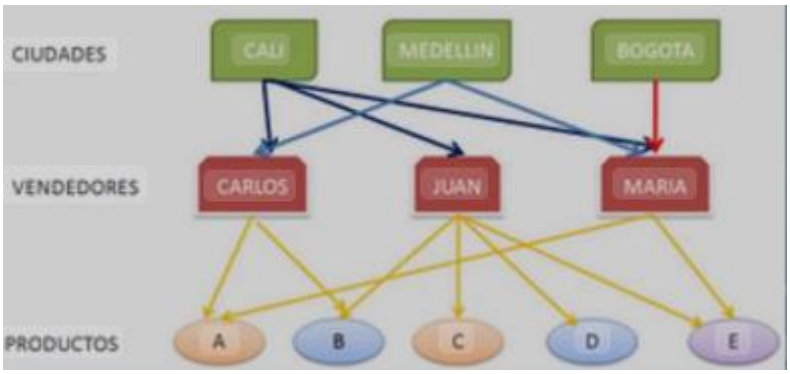
- **Estructura clave**: Un registro "hijo" solo puede tener **un único registro "padre"**. Esto crea una dependencia estricta y una ruta de acceso única desde la raíz (el nodo superior) hasta cualquier dato.
- **Analogía**: Podríamos imaginarlo como el sistema de carpetas de un ordenador. Tienes una carpeta principal (padre) y dentro puedes tener varios archivos o subcarpetas (hijos), pero cada una de esas subcarpetas pertenece únicamente a su carpeta contenedora superior.



2.2.8.1.2 Modelo en Red

Este modelo surgió como una evolución del jerárquico para superar sus limitaciones. Aunque también utiliza una estructura de nodos interconectados, introduce una flexibilidad crucial.

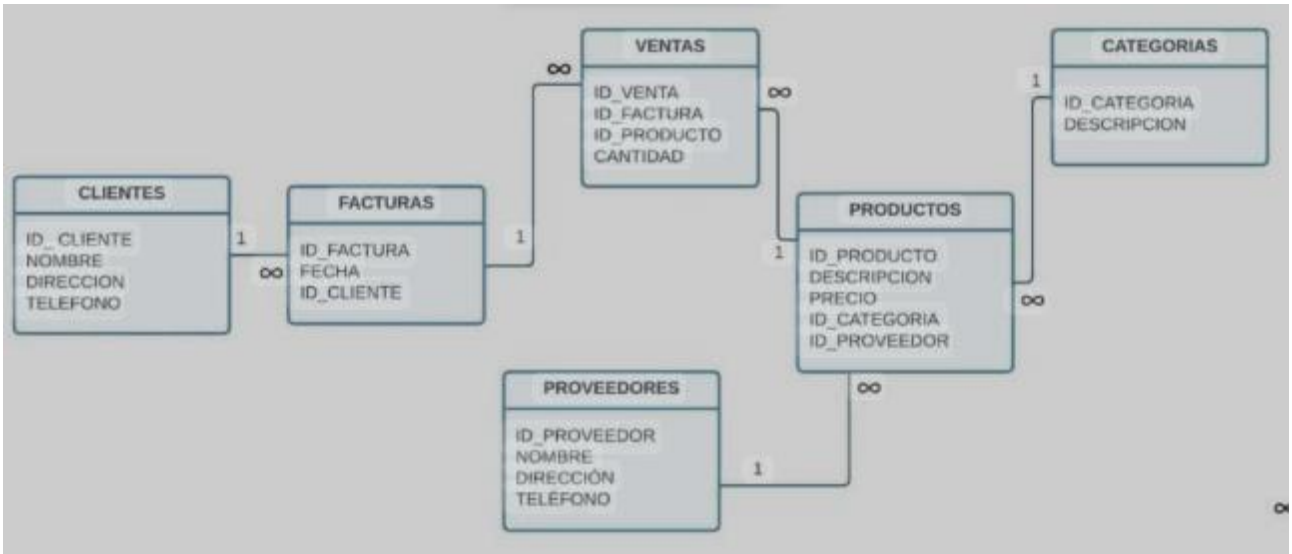
- **Estructura clave**: A diferencia del modelo jerárquico, aquí un registro "hijo" **puede tener varios "padres"**. Esto permite modelar relaciones mucho más complejas y realistas, donde un elemento puede estar asociado a múltiples categorías o entidades superiores.
- **Analogía**: Si en una biblioteca un libro (hijo) puede pertenecer a varios géneros literarios (padres) como "Ciencia Ficción" y "Novela de Aventuras", el modelo en red podría representar esta relación múltiple, algo que el modelo jerárquico no podría hacer fácilmente.



2.2.8.1.3 Modelo Relacional

Es el modelo más extendido y utilizado en la actualidad. Su gran éxito se debe a su simplicidad conceptual y su sólida base matemática.

- **Estructura clave**: Organiza toda la información en **tablas bidimensionales**, también llamadas **relaciones**.
  - Cada tabla está compuesta por **filas** (conocidas como *tuplas* o *registros*), que representan un elemento único (por ejemplo, un cliente).
  - Las columnas (conocidas como *atributos* o *campos*) describen las propiedades de esos elementos (por ejemplo, nombre, dirección, teléfono).
- **Funcionamiento**: Las relaciones entre los datos de diferentes tablas se establecen mediante claves (campos comunes), lo que permite combinar y consultar la información de manera muy flexible y potente sin necesidad de punteros físicos como en los modelos anteriores.
- **Ejemplos de SGBD**: La mayoría de los sistemas populares como Oracle, Microsoft SQL Server, MySQL y PostgreSQL se basan en este modelo.



2.2.8.1.4 Modelo Orientado a Objetos

Este modelo traslada los conceptos de la programación orientada a objetos (POO) al mundo de las bases de datos.

La base de datos se define en términos de **objetos**, que son unidades que encapsulan tanto datos (*propiedades*) como el comportamiento asociado a esos datos (*métodos*).

Es especialmente útil en aplicaciones complejas donde los datos y las operaciones que se realizan sobre ellos están íntimamente ligados.

2.2.8.1.5 Otros Modelos

Las fuentes también citan brevemente la existencia de otros modelos que a menudo son extensiones o combinaciones de los anteriores, como:

- **Objeto-Relacional**: Un modelo híbrido que busca combinar la simplicidad y robustez del modelo relacional con las capacidades avanzadas del modelo orientado a objetos.
- **Deductivas, Multidimensionales y Transaccionales**: Son otros enfoques más especializados para casos de uso específicos como la inteligencia artificial, el análisis de datos (data warehousing) o el procesamiento de transacciones a gran escala

2.2.8.2 SGBD Comerciales

Son sistemas que requieren el pago de una licencia, aunque a menudo ofrecen versiones gratuitas para desarrolladores o con funcionalidades limitadas. Los más destacados son:

- **Oracle Database**: Reconocido como uno de los líderes a nivel mundial. Es multiplataforma, potente, seguro y escalable, utilizado principalmente en grandes empresas. Ofrece una versión gratuita llamada Express Edition.
- **Microsoft SQL Server**: La alternativa de Microsoft, que funciona exclusivamente en sistemas Windows. Es un SGBD relacional con arquitectura cliente/servidor.
- **DB2**: El SGBD de IBM, multiplataforma y con un fuerte soporte para la integración nativa de XML.
- **Informix**: Otra opción de IBM, conocido por consumir menos recursos que otros grandes SGBD.

2.2.8.3 SGBD Libres (Open Source)

Son sistemas cuyo código fuente es abierto y pueden ser utilizados, modificados y distribuidos libremente, representando una alternativa a coste cero frente a los SGBD comerciales. Los más populares son:

- **MySQL**: Es el SGBD de código abierto más extendido, especialmente en aplicaciones web. Es relacional, multihilo, multiusuario y multiplataforma. Aunque es libre, también ofrece una licencia comercial.

- **PostgreSQL**: Considerado el SGBD de código abierto más avanzado. Es un sistema relacional orientado a objetos.
- **SQLite**: Es una biblioteca en C que implementa un motor de base de datos relacional. No funciona como un servidor, sino que se integra directamente en la aplicación, lo que lo hace extremadamente rápido.

2.2.9 Bases de Datos Centralizadas

En una arquitectura centralizada, **la totalidad de la base de datos y el SGBD residen en un único sistema informático (un solo ordenador o sitio)**. Los usuarios acceden a este punto central, tradicionalmente a través de terminales.

- **Características**: Todos los componentes (datos, SGBD, dispositivos) están en una sola ubicación.

- **Ventajas**:
  - **Simplicidad y control**: La gestión y seguridad son más sencillas.
  - **Evita redundancia e inconsistencia**: Al haber una única copia de los datos, no pueden producirse discrepancias.
  - ◦**Mantenimiento más económico**: Requiere menos recursos humanos y técnicos.

- **Inconvenientes**:
  - **Punto único de fallo**: Si el sistema central falla, se pierde por completo el acceso a la información.
  - **Escalabilidad limitada**: El poder de cómputo está limitado a una sola máquina.
  - **Vulnerabilidad**: En caso de desastre, la recuperación de la información es más difícil.

2.2.10 Bases de Datos Distribuidas

La evolución de las redes de comunicación y la necesidad de integrar información de múltiples fuentes dieron lugar a los sistemas de bases de datos distribuidas.

2.2.10.1 Concepto de Base de Datos Distribuida (BDD)

Una BDD es un **conjunto de múltiples bases de datos lógicamente relacionadas que se encuentran distribuidas físicamente en diferentes ordenadores (nodos) interconectados por una red de comunicaciones**.

El **Sistema Gestor de Bases de Datos Distribuida (SGBDD)** se encarga de manejar esta distribución de forma **transparente** para el usuario. Esto significa que el usuario interactúa con el sistema como si se tratara de una única base de datos centralizada, sin necesidad de saber dónde están almacenados realmente los datos.

- **Ejemplo**: Una cadena hotelera con sedes en varias ciudades. Cada hotel tiene su propia base de datos local, pero la dirección central puede realizar consultas sobre la ocupación total como si todos los datos estuvieran en un solo lugar.

2.2.10.2 Ventajas e Inconvenientes de las BDD

- **Ventajas**:
  - **Mayor disponibilidad y fiabilidad**: Si un nodo falla, el resto del sistema puede seguir funcionando.
  - **Mejor rendimiento**: El acceso a los datos locales es más rápido y las consultas pueden procesarse en paralelo.
  - **Escalabilidad**: Es fácil añadir nuevos nodos a la red.

- **Inconvenientes**:
  - **Complejidad**: La gestión de la concurrencia, seguridad y recuperación es mucho más compleja.
  - **Coste de comunicación**: El intercambio de mensajes entre nodos introduce una sobrecarga de rendimiento.
  - **Diseño complejo**: El diseño del software y de la distribución de los datos es un desafío técnico.

2.2.10.3 Fragmentación de la Información

Para distribuir una base de datos, es necesario dividir sus tablas en piezas más pequeñas llamadas **fragmentos**. Cada fragmento se asigna a uno o más nodos de la red. El objetivo es encontrar un nivel de particionamiento que optimice el rendimiento de las aplicaciones.

Al fragmentar, se deben cumplir tres reglas:

1. **Completitud**: Todos los datos de la tabla original deben estar en algún fragmento.
2. **Reconstrucción**: Debe ser posible reconstruir la tabla original a partir de sus fragmentos.
3. **Disyunción**: Si no hay replicación, un dato solo debe pertenecer a un único fragmento (excepto la clave primaria en fragmentación vertical).

2.2.10.4 Políticas de Fragmentación

Existen tres tipos principales de estrategias para dividir los datos en una base de datos distribuida:

1. Fragmentación Horizontal

Esta política **divide una tabla por sus filas (tuplas)**. Cada fragmento contiene un subconjunto de las filas de la tabla original, pero mantiene todas las columnas. La división se basa en una condición sobre los valores de uno o más atributos.

- **Ejemplo**: Consideremos una tabla EMPLEADOS de una empresa multinacional con columnas ID_Empleado, Nombre, Salario y Sede. Se puede fragmentar horizontalmente según la sede:
  - **Fragmento 1 (en el servidor de Madrid)**: Contendría todas las filas (empleados) donde Sede = 'Madrid'.
  - **Fragmento 2 (en el servidor de México)**: Contendría todas las filas donde Sede = 'México'.
  - Esta estrategia es útil porque las consultas sobre los empleados de Madrid se ejecutarán localmente en el servidor de Madrid, mejorando la velocidad de acceso.

2. Fragmentación Vertical

Esta política **divide una tabla por sus columnas (atributos)**. Cada fragmento contiene un subconjunto de las columnas de la tabla original para todas las filas. Es crucial que cada fragmento incluya la clave primaria de la tabla para poder reconstruir el registro completo.

- **Ejemplo**: En la misma tabla EMPLEADOS, se podría realizar una fragmentación vertical para separar la información de uso frecuente de la confidencial:
  - **Fragmento 1 (Datos Generales)**: Contendría las columnas ID_Empleado, Nombre y Sede.
  - **Fragmento 2 (Datos Salariales)**: Contendría las columnas ID_Empleado y Salario.
  - ID_Empleado (la clave primaria) se repite en ambos fragmentos. Esto permite que una aplicación de RRHH que solo necesita listar nombres no tenga que acceder a los datos salariales, mejorando la seguridad y la eficiencia.

3. Fragmentación Mixta o Híbrida

Esta política **combina las dos anteriores**. Se puede aplicar primero una fragmentación (por ejemplo, vertical) y luego aplicar otra (horizontal) sobre uno o más de los fragmentos resultantes, o viceversa.

- **Ejemplo**: Siguiendo con la tabla EMPLEADOS, podríamos primero aplicar la **fragmentación vertical** para separar datos generales de salariales. Después, aplicar una **fragmentación horizontal** sobre el fragmento de "Datos Generales" según la Sede, distribuyendo los datos de cada sede en su servidor local. El resultado sería una distribución de datos muy optimizada tanto para la localidad como para la seguridad.

Estas políticas de fragmentación son esenciales para diseñar sistemas de bases de datos distribuidas que sean eficientes, escalables y seguros.