# Unidad 1: Sistemas de Gestión de Bases de Datos

## Sistemas tradicionales de archivos VS Sistemas de BD

Antes de la llegada de los sistemas de gestión de bases de datos (SGBD), las organizaciones normalmente han almacenado la información usando estos sistemas, pero mantener la información en estos sistemas de archivos tiene una serie de inconvenientes importantes:

- **Redundancia e inconsistencia de datos:** Existen datos que pueden repetirse en diferentes lugares o archivos, esto provoca que, teniendo esa duplicidad de datos, el almacenamiento y el costo (en recursos del sistema) de acceso sean más altos. Inconsistencia de datos se presentará porque las copias de los mismos datos en diferentes archivos pueden no coincidir, pues si en un archivo se hicieron cambios de los datos, en los otros archivos donde estaban los mismos datos no son modificados automáticamente.

- **Dificultad en el acceso a los datos:** Cuando se requiere de ciertos datos diferentes de archivos diferentes, la obtención, consulta y modificación de los datos no puede hacerse directamente de forma práctica y eficiente. Tendrían que desarrollarse sistemas de recuperación de datos para realizar esa operación específica, o desarrollar un sistema de recuperación de datos para uso general y ajustarlo de acuerdo a las necesidades.

- **Aislamiento de datos:** Debido a que los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos apropiados.

- **Problemas de integridad:** Los valores de los datos almacenados en la BD deben satisfacer ciertas restricciones de consistencia. Los desarrolladores hacen cumplir estas restricciones en el sistema añadiendo código apropiado en las diversas aplicaciones. Sin embargo, cuando se añaden nuevas restricciones es difícil cambiar los programas para hacer que se cumplan. Esto se complica cuando las restricciones implican diferentes elementos de datos de diferentes archivos.

- **Problemas de atomicidad:** En muchas aplicaciones es crucial asegurar que, cuando ocurra un fallo y sea detectado, se restauren los datos a un estado de consistencia que existía antes del fallo. Es difícil asegurar esta propiedad en un sistema de archivos tradicional.

- **Anomalías en el acceso concurrente:** en estos sistemas un entorno en el que permita a múltiples usuarios actualizar los datos de un mismo archivo simultáneamente puede dar lugar a datos inconsistentes o un estado incorrecto.

- **Problemas de seguridad:** No todos los usuarios de un sistema de bases de datos deberían poder acceder a todos los datos. En estos sistemas es difícil garantizar tales restricciones de seguridad.

Estas dificultades, entre otras, han motivado el desarrollo de los sistemas de bases de datos para resolver estos problemas.

Los sistemas de archivos (file systems) se especializan en la administración de información desestructurada. Tradicionalmente, un sistema de archivos no tiene conocimiento del contenido interno de un archivo. Sí tiene alguna información que lo relaciona al archivo, como una ubicación (ruta del archivo), o timestamps (fechas de creación, modificación, etc). Algunos sistemas de archivos permiten almacenar meta-data extendida que puede ser leída por aplicaciones, pero no por el sistema de archivos en sí.

Las bases de datos se especializan en la información estructurada. Tradicionalmente, una base de datos sabe el formato de los contenidos internos de una tabla. Proveen un mecanismo de direccionamiento multi-dimensional (“filas” y “columnas”, por ejemplo) y un espacio de nombres optimizado (índices). También provee mecanismos para hallar información específica basándose en sus características.

Una debilidad en los sistemas de archivos es que ellos no realizan un buen trabajo de capturar las relaciones entre los archivos; una jerarquía de espacios de nombres funciona “más o menos”, pero le entregan una carga de mantener la relación entre los archivos al usuario. Algunas aplicaciones pueden absorber un poco de esa carga, pero entonces las relaciones serán específicas de las aplicaciones.

Una debilidad en los sistemas de base de datos es que dependen en que vos definas la estructura de datos de antemano. Añadir una característica a un conjunto de datos existente (como por ejemplo, agregar una nueva columna a una tabla que ya existe) es posible, pero no siempre fácil de lograr (¿de dónde sacás los datos para esa nueva columna?).

## Introducción a las Bases de Datos

Una base de datos es una colección de datos relacionados. Un dato es un hecho conocido que se puede grabar y que tiene un significado implícito (por ejemplo, tu DNI, tu número de teléfono, etc.).

A pesar de que esta definición pueda parecer genérica (un libro podría ser una base de datos bajo esta definición), el uso del término base de datos es más restringido.

Una BD tiene las siguientes propiedades implícitas:

- Representa algún aspecto del mundo real (lo que en ocasiones se denomina mini-mundo).

- Es una colección de datos lógicamente coherente con algún significado inherente. Un surtido de datos aleatorio no se puede denominar como una base de datos.

- Una base de datos se diseña, construye, y rellena con datos para un propósito específico. Dispone de un grupo pretendido de usuarios y algunas aplicaciones preconcebidas en las que esos usuarios están interesados.

Un sistema de administración de base de datos (DBMS, database management system) es una colección de programas que permite a los usuarios crear y mantener una base de datos. El DBMS es un sistema de software de propósito general que facilita los procesos siguientes procesos de una BD:

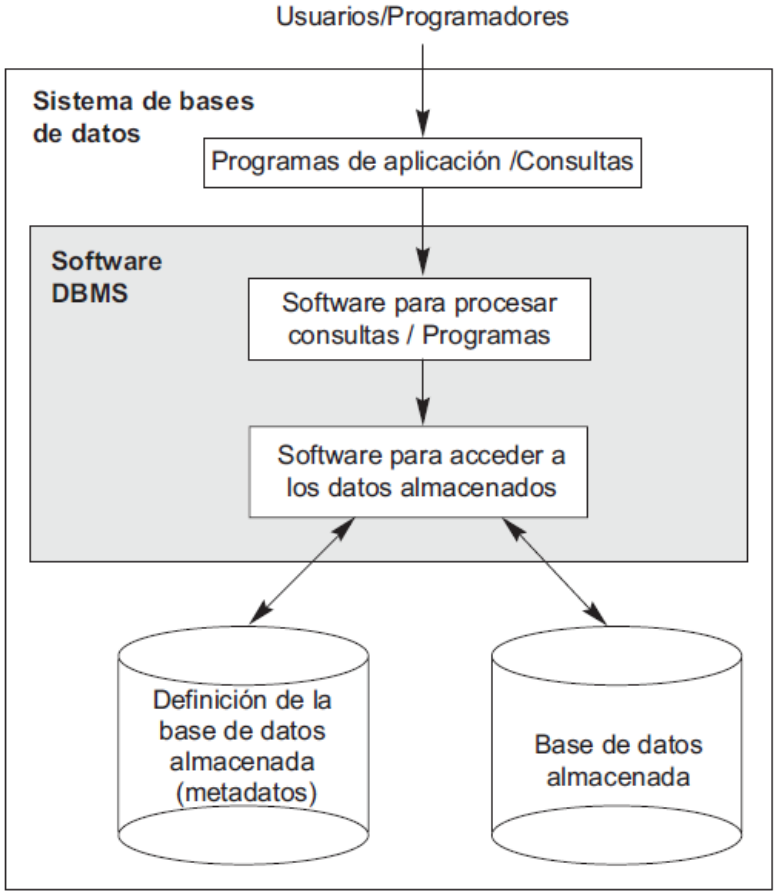
- **Definición:** implica especificar los tipos de datos, estructuras y restricciones de los datos que se almacenarán en la BD. La definición o información descriptiva de una base de datos también se almacena en esta última en forma de catálogo o diccionario de la base de datos; es lo que se conoce como metadatos.

- **Construcción:** es el proceso consistente en almacenar los datos en algún medio de almacenamiento controlado por el DBMS.

- **Manipulación:** incluye funciones como la consulta de la base de datos para recuperar datos específicos, actualizar la base de datos para reflejar los cambios introducidos en el minimundo y generar informes a partir de los datos.

- **Compartición:** compartir una base de datos permite que varios usuarios y programas accedan a la BD de forma simultánea.

Una aplicación accede a la BD enviando consultas o solicitudes de datos al DBMS. Una **consulta** normalmente provoca la recuperación de algunos datos; una **transacción** puede provocar la lectura o la escritura de algunos datos en la BD.




El DBMS tar tección contra
accesos no e tiempo.

**Un sistema tware DBMS.**

**<u>Característi</u>**

A diferencia rea y maneja
un archivo p anejo de los
sueldos de p único
almacén de

Las principales características de la metodología de bases de datos frente a la metodología de procesamiento de archivos son las siguientes:

- Naturaleza autodescriptiva de un sistema de bases de datos.

- Aislamiento entre programas y datos, y abstracción de datos.
- Soporte de varias vistas de los datos.
- Compartición de datos y procesamiento de transacciones multiusuario.

**Naturaleza autodescriptiva de un sistema de base de datos**

El sistema de base de datos no solo contiene la propia base de datos, sino también una completa definición o descripción de la estructura de la base de datos y sus restricciones. Esta estructura se almacena en el catálogo del DBMS. La información almacenada en el catálogo se denomina metadatos y describe la estructura de la base de datos principal.

Esto es debido a que un software de DBMS de propósito general no se escribe para una aplicación específica en mente, es por eso que debe referirse al catálogo para conocer la estructura de datos de una BD específica.

**Aislamiento entre programas y datos, y abstracción de datos**

Cuando hacés un programa en C y guardás un archivo, la estructura de ese archivo está atada a tu programa. Si necestiás hacer un cambio en esa estructura, tenés que realizar un cambio en todos los programas que accedan a ese archivo.

Ese es el enfoque tradicional del procesamiento de archivos. Sin embargo, los programas que acceden a un DBMS no necesitan esos cambios en la mayoría de los casos. La estructura de los archivos se almacena en el catálogo DBMS, independientemente de los programas de acceso. Esta propiedad se llama **independencia programa-datos.**

En algunos tipos de sistemas de bases de datos, como los sistemas orientados a objetos y los de objetos relacionales, los usuarios pueden definir operaciones sobre los datos como parte de las definiciones de la base de datos. Una operación (también denominada función o método) se puede especificar de dos formas. La interfaz (o firma) de una operación incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La implementación (o método) de la operación se especifica separadamente y puede modificarse sin que la interfaz se vea afectada. Las aplicaciones de usuario pueden operar sobre los datos invocando estas operaciones por sus nombres y argumentos, independientemente de cómo estén implementadas las operaciones. Esto puede recibir el nombre de **independencia programa-operación.**

La característica que permite la independencia programa-datos y la independencia programa-operación se denomina **abstracción de datos**. Un DBMS proporciona a los usuarios una representación conceptual de los datos que no incluye muchos de los detalles de cómo están almacenados los datos o de cómo están implementadas las operaciones. Informalmente, un modelo de datos es un tipo de abstracción de datos que se utiliza para proporcionar esa representación conceptual. El modelo de datos utiliza conceptos lógicos, como objetos, sus propiedades y sus relaciones, lo que para la mayoría de los usuarios es más fácil de entender que los conceptos de almacenamiento en el computador. Por ello, el modelo de datos oculta los detalles del almacenamiento y de la implementación que no resultan interesantes a la mayoría de los usuarios de bases de datos.

**Soporte de varias vistas de los datos**

El DBMS puede mostrar cierta vista de los datos según las necesidades del usuario actual. Una vista puede ser un subconjunto de la base de datos o puede contener datos virtuales derivados de los archivos de la base de datos pero que no están explícitamente almacenados. Algunos

usuarios no tienen la necesidad de preocuparse por si los datos a los que se refieren están almacenados o son derivados.

**Compartición de datos y procesamiento de transacciones multiusuario**

El DBMS debe incluir software de control de la concurrencia para que varios usuarios que intenten actualizar los mismos datos, lo hagan de un modo controlado para que el resultado de la actualización sea correcto (¡condiciones de carrera!).

El concepto de transacción es cada vez más importante para las aplicaciones de bases de datos. Una transacción es un programa en ejecución o proceso que incluye uno o más accesos a la base de datos, como la lectura o la actualización de los registros de la misma. Se supone que una transacción ejecuta un acceso lógicamente correcto a la base de datos si lo ejecutó íntegramente sin interferencia de otras transacciones.

El DBMS debe implementar varias propiedades de transacción. La propiedad aislamiento garantiza que parezca que cada transacción se ejecuta de forma aislada de otras transacciones, aunque puedan estar ejecutándose cientos de transacciones al mismo tiempo. La propiedad de atomicidad garantiza que se ejecuten o todas o ninguna de las operaciones de bases de datos de una transacción.

**Mercado actual**

El mercado actual está plagado de diferentes tipos de software DBMS. No solo por la competencia entre empresas, sino porque existen diferentes tipos de bases de datos, las cuales requieren un software adecuado para su administración.

Por ejemplo, MySQL es un DBMS open source para base de datos relacionales (las que más comúnmente son enseñadas, aquellas que disponen tablas y relaciones entre las mismas).

Pero también existen otros DBMS como MongoDB, utilizado para base de datos no relacionales (orientadas a documentos, en el caso de MongoDB).

U otros como PostgreSQL, que es una DMBS orientada a objetos relacionales (utiliza el paradigma orientado a objetos como base, permitiendo por ejemplo el uso de clases, herencia, etc).

SQLServer es el DBMS que ofrece Microsoft, y es generalmente usado dentro de su ecosistema de productos/servicios (C#, .Net Framework, etc.).

## Sistema de bases de datos

**Registro o tupla**

En el contexto de una base de datos relacional, un registro (también llamado fila o tupla) representa un objeto único de datos implícitamente estructurados en una tabla. En términos simples, una tabla de una base de datos puede imaginarse formada de filas y columnas. Cada fila de una tabla representa un conjunto de datos relacionados, y todas las filas de la misma tabla tienen la misma estructura.

Un registro es un conjunto de campos que contienen los datos que pertenecen a una misma repetición de entidad. Se le asigna automáticamente un número consecutivo (número de registro) que en ocasiones es usado como índice aunque lo normal y práctico es asignarle a cada registro un campo clave para su búsqueda.

**Estructura de datos relacional**

El modelo relacional, para el modelado y la gestión de bases de datos, es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos.

Su idea fundamental es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas. La mayoría de las veces se conceptualiza de una manera fácil de imaginar, pensando en cada relación como si fuese una tabla que está compuesta por registros (cada fila de la tabla sería un registro o "tupla") y columnas (también llamadas "campos").

## Relaciones

En una BDR, todos los datos se almacenan y se accede a ellos por medio de relaciones previamente establecidas.

- **Relaciones base: l**as relaciones que almacenan datos son llamadas relaciones base y su implementación es llamada "tabla".

- **Relaciones derivadas:** son calculadas al aplicar operaciones relacionales. Estas relaciones son llamadas relaciones derivadas y su implementación es llamada "vista" o "consulta". Las relaciones derivadas son convenientes, ya que expresan información de varias relaciones actuando como si fuera una sola tabla**.**

## Claves

- **Clave primaria**: una clave primaria es una clave única (puede estar conformada por uno o más campos de la tabla) elegida entre todas las candidatas que define unívocamente a todos los demás atributos de la tabla para especificar los datos que serán relacionados con las demás tablas. La forma de hacer esto (relación entre tablas) es por medio de claves foráneas.

- **Clave externa o foránea:** una clave foránea es una referencia a una clave en otra tabla, determina la relación existente en dos tablas. Las claves foráneas no necesitan ser claves únicas en la tabla donde están y sí a donde están referenciadas.

  Por ejemplo, el código de departamento puede ser una clave foránea en la tabla de empleados. Se permite que haya varios empleados en un mismo departamento, pero habrá uno y solo un departamento por cada clave distinta de departamento en la tabla de departamentos.

- **Clave índice:** Las claves índice surgen con la necesidad de tener un acceso más rápido a los datos. Los índices pueden ser creados con cualquier combinación de campos de una tabla. Las consultas que filtran registros por medio de estos campos, pueden encontrar los registros de forma no secuencial usando la clave índice.

  Las bases de datos relacionales incluyen múltiples técnicas de ordenamiento, cada una de ellas es óptima para cierta distribución de datos y tamaño de la relación.

  Los índices generalmente no se consideran parte de la base de datos, pues son un detalle agregado. Sin embargo, las claves índices son desarrolladas por el mismo grupo de programadores que las otras partes de la base de datos.

## Metodología de base de datos

La metodología de base de datos posee una característica fundamental: la **abstracción de los datos**. Es decir, existe cierto nivel de abstracción de datos, por el cual se suprimen detalles de la organización o el almacenamiento de los mismos, y se da relevancia a características fundamentales de los datos para poder entenderlos o conocerlos mejor (por dar un ejemplo, no te hace falta saber en qué ruta o cuánto espacio de disco te ocupa un dato, se le da más relevancia al dato en sí).

La abstracción sirve para que los diferentes tipos de usuarios puedan observar dichos datos con el nivel de detalle que prefieran.

Un modelo de datos es una colección de conceptos que se pueden utilizar para describir la estructura de la base de datos, y es el encargado de proporcionar los medios necesarios para conseguir la abstracción. Con estructura de base de datos hacemos referencia a los tipos de datos, relaciones, y restricciones que deben mantenerse para los datos (es decir, cómo está compuesta la BD en sí, qué tablas hay, qué datos hay en esas tablas, de qué tipo son, qué relaciones hay entre tablas…).

## Categoría de modelos de datos

- **Modelos de datos de alto nivel (o conceptuales):** este tipo de modelo de datos oculta al usuario cómo están almacenados los datos, y se centran más en cómo los usuarios perciben o ven los datos.

- **Modelos de datos de bajo nivel (o físicos):** describen los detalles de cómo se almacenan los datos en la computadora, en forma de archivos, representando la información como formatos de registro, ordenación de registros y rutas de acceso.

- **Modelos de datos representativos (o de implementación):** ofrecen conceptos que los usuarios finales pueden entender, pero no están demasiado lejos de cómo se organizan los datos dentro de la computadora. Estos son los más utilizados por los DBMS comerciales tradicionales.

Los modelos de datos conceptuales utilizan conceptos como **entidades, atributos y relaciones.**

Una **entidad** representa un objeto o concepto del mundo real, como un empleado o un proyecto que se describe en la base de datos.

Un **atributo** representa alguna propiedad de interés que describe a una entidad, como, por ejemplo, el nombre o el salario de un empleado.

Una **relación** entre dos o más entidades representa una asociación entre dos o más entidades; por ejemplo, una relación de trabajo entre un empleado y un proyecto.

## Esquemas, instancias, y estado de la BD

El esquema de una BD es la descripción de la misma base datos, la cual se específica durante la fase de diseño y no se espera que cambie con frecuencia. La visualización de un esquema recibe el nombre de diagrama del esquema. A cada objeto del esquema lo denominamos estructura del esquema.

En un diagrama del esquema solo se muestran algunos aspectos, como los nombres de los tipos de registros y los elementos de datos. Otros aspectos no se especifican, como los tipos de dato de cada elemento de datos.

**ESTUDIANTE**

| NOMBRE | NumEstudiante | Clase | Especialidad |
|---|---|---|---|

**CURSO**

| NombreCurso | NumCurso | Horas | Departamento |
|---|---|---|---|

**PRERREQUISITO**

| NumCurso | NumPrerrequisito |
|---|---|

**SECCIÓN**

| IDSeccion | NumCurso | Semestre | Profesor |
|---|---|---|---|

**INFORME_CALIF**

| NumEstudiante | IDSeccion | Nota |
|---|---|---|

Cabe resaltar que hay una diferencia entre esquema de la base de datos y estado de la base de datos. **El estado de una BD es una snapshot de los datos de la misma en un momento concreto.** Cuando definimos una BD nueva, solo especificamos su esquema al DBMS. A estas alturas, el estado de la BD es un estado vacío, sin datos. Cuando empezamos a cagar datos iniciales el estado cambia.

## Arquitectura de tres esquemas

El objetivo de la arquitectura de tres esquemas es separar las aplicaciones de usuario y las bases de datos físicas. Hay tres niveles:

- **Nivel interno:** posee un esquema interno que describe la estructura de almacenamiento físico de la BD. El esquema interno utiliza un modelo de datos físico y describe todos los detalles del almacenamiento de datos y las rutas de acceso a la base de datos.

- **Nivel conceptual:** tiene un esquema conceptual que describe la estructura de toda la base de datos para los usuarios. Este esquema oculta detalles de las estructuras de almacenamiento físico y se concentra en describir las entidades, tipos de datos, relaciones, operaciones de los usuarios, y las restricciones.

- **Nivel de vista o externo:** incluye una cierta cantidad de esquemas externos o vistas de usuario. Un esquema externo describe la parte de la BD en la que un grupo de usuarios en particular está interesado y le oculta el resto de la base de datos.

**Independencia de los datos**

- **Independencia lógica de datos:** se refiere a la habilidad de cambiar el esquema conceptual sin tener que cambiar las vistas externas o las aplicaciones que usan la BD. En sí, significa que si añadimos una columna o removemos otra, las vistas de los usuarios y los programas no deberían ser modificadas. Por esta misma razón, la independencia lógica de datos es mucho más difícil de lograr que la independencia física.

- **Independencia física de datos:** es la capacidad de cambiar el esquema físico sin tener que cambiar el esquema conceptual, y por lo tanto, tampoco los externos. En general son modificaciones a cómo se almacenan los datos o en qué tipo de lugar. Pueden realizarse por razones de performance.

**Lenguajes DBMS**

Una vez completado el diseño de una BD y elegido un DBMS para implementarla, el primer paso es especificar los esquemas conceptual e interno.

- **Lenguaje de definición de datos (DDL):** es utilizado para definir los dos esquemas en un DBMS donde no se mantiene una separación estricta de niveles. El DBMS tendrá un compilador DDL cuya función es procesar las sentencias DDL.

- **Lengua de definición de almacenamiento (SDL):** en los DBMS donde hay una clara separación entre los niveles conceptual e interno, se utiliza DDL sólo para especificar el esquema conceptual, y para el interno se utiliza SDL.

- **Lenguaje de definición de vistas (VDL):** para conseguir una arquitectura de tres esquemas real se necesita este tercer lenguaje, a fin de especificar las vistas de usuario y sus mapeados al esquema conceptual. Pero en la mayoría de los DBMS actuales se utiliza el DDL para definir tanto el esquema conceptual como el externo. En los DBMS relacionales se utiliza SQL como VDL.

- **Lenguaje de manipulación de datos (DML):** es el lenguaje por el cual los usuarios pueden disponer de los medios para manipular la base de datos. La recuperación, inserción, borrado, o modificación de datos son las manipulaciones típicas.

En los DBMSs actuales no existe un lenguaje distinto por cada tipo de lenguaje que mencionamos anteriormente. Se utiliza un lenguaje comprensivo que incluye construcciones para la definición del esquema conceptual, la definición de vistas, y la manipulación de datos. La definición del almacenamiento es dejada de lado normalmente.

Un ejemplo de un lenguaje comprensivo es SQL en base de datos relacionales. SQL representa una combinación de DDL, VDL y DML.

Existen dos tipos de lenguajes DML:

- **Alto nivel, set at a time:** son lenguajes como SQL, los cuales pueden ser ingresados con ayuda del DBMS mediante una terminal o un monitor, o también pueden estar incrustados en un lenguaje de programación de propósito general. Una particularidad de estos lenguajes es que pueden especificar y recuperar muchos registros con una sola sentencia DML, por lo cual reciben el nombre de set a at time.

- **Bajo nivel, record at a time:** estos sí o sí *deben* incrustarse en un lenguaje de programación de propósito general. Normalmente recuperar registros individuales u objetos de la BD, y los procesa por separado.

Siempre que hay comandos DML, de alto o de bajo nivel, incrustados en un lenguaje de programación de propósito general, ese lenguaje se denomina lenguaje host, y el DML sublenguaje de datos. Por el contrario, un DML de alto nivel utilizado de forma interactiva independiente se conoce como lenguaje de consulta. En general, tanto los comandos de recuperación como los de actualización de un DML de alto nivel se pueden utilizar interactivamente y, por tanto, se consideran como parte del lenguaje de consulta.

**Tipos de usuarios del DBMS**

Un usuario es todo aquel que tenga contacto con el sistema de bases de datos.

Se tienen 3 clases generales de usuarios:

1. **Programador de aplicaciones:** son aquellos profesionales en informática que interactúan con el sistema a través del DML (Lenguaje de Manipulación de Datos), los cuales se encuentran en un lenguaje de programación (Pascal, Cobol, etc.) Es el encargado de escribir programas de aplicación que usen Bases de Datos.

2. **Usuario Final:** accede a la base de datos desde un equipo en el cual puede utilizar lenguaje de consulta generado como parte del sistema o acude a un programa de aplicación suministrado por un programador.

3. **Administrador de Bases de Datos**: es el encargado del control general del sistema.

Todo usuario que ingrese o consulte una base de datos puede clasificarse:

**Programador de Aplicaciones.**

**Usuario sofisticado:** interactúa con el sistema sin escribir programas. Generan consultas en un lenguaje de bases de datos.

**Usuario especializado:** algunos usuarios sofisticados desarrollan aplicaciones de bases de datos especializadas. Entre estas aplicaciones se encuentran los sistemas de diseño asistido por computador.

**Usuarios ingenuos:** es el usuario final que utiliza bases de datos sin saberlo, para él es totalmente transparente como se generan las consultas de la información.

Quienes diseñan y participan en el mantenimiento de un BD se les clasifica como Actores en el escenario y Trabajadores tras bambalinas

**Actores en el escenario:** personas que su trabajo depende del uso constante una base de datos.

**Trabajadores tras bambalinas:** están para mantener el sistema de base datos.

**Database Administrators (DBA):** administran 2 recursos, 1. la base de datos y 2. el SGBD y el software con el relacionado. El Administrador de Base de Datos (DBA) es quien autoriza el acceso a la base de datos, vigilar el uso y adquirir hardware y software necesarios para su uso. También es el responsable de velar por la seguridad y lentitud en el sistema.

**Diseñador de Base de Datos:** es el encargado de estructurar la arquitectura para representar y almacenar los datos. Él debe atender a los usuarios de Bases de Datos para comprender sus necesidades presentando un diseño que dé respuesta a sus necesidades.

**Usuarios Finales:** son quienes requieren acceso a la base de datos para generar consultas e informes. Hay varios usuarios finales como son:

- **Usuarios finales esporádicos:** acceden de vez en cuando pero esto no significa que siempre requieran la misma información.
- **Usuarios finales simples o paramétricos:** su función gira en torno a consultas y actualizaciones de la base de datos. Todos estamos acostumbrados a tratar con estos usuarios, como los cajeros bancarios al revisar los saldos, al generar retiros y depósitos.
- **Usuarios finales avanzados**: estos son ingenieros, analistas de negocios, científicos, son quienes conocen los recursos del SGBD para satisfacer requerimientos complejos.
- **Usuarios Autónomos**: utilizan bases de datos personalizadas basadas en programas comerciales que cuentas con interfaces de fácil uso.

**Analista de sistemas y programadores de aplicaciones**: determinan los requerimientos de los usuarios finales.

**Diseñadores e implementadores del SGBD:** se encarga de diseñar e implementar los módulos e interfaces de SGBD. Un Sistema de Gestión de Base de Datos consta de varios componentes y módulos.

## DBMS distribuidos

Los sistemas distribuidos de bases de datos consisten en sitios débilmente acoplados que no comparten ningún componente físico. Además, puede que los sistemas de bases de datos que se ejecutan en cada sitio tengan un grado sustancial de independencia mutua.

Cada sitio puede participar en la ejecución de transacciones que tienen acceso a los datos de uno o varios de los sitios. La diferencia principal entre los sistemas de bases de datos centralizados y los distribuidos es que, en los primeros, los datos residen en una única ubicación, mientras que en los segundos los datos residen en varias ubicaciones. La distribución de los datos es causa de muchas dificultades en el procesamiento de las transacciones y de las consultas.

Un sistema distribuido de bases de datos se almacena en varias computadoras. Los principales factores que distinguen un SBDD de un sistema centralizado son los siguientes:

- Hay múltiples computadores, llamados sitios o nodos.

- Estos nodos deben de estar comunicados por medio de algún tipo de red de comunicaciones para transmitir datos y órdenes entre los sitios.

**Bases de datos heterogéneas y homogéneas**

En las **bases de datos distribuidas homogéneas** todos los sitios tienen idéntico software de sistemas gestores de bases de datos, son conscientes de la existencia de los demás sitios y acuerdan cooperar en el procesamiento de las solicitudes de los usuarios. En estos sistemas los sitios locales renuncian a una parte de su autonomía en cuanto a su derecho a modificar los esquemas o el software del sistema gestor de bases de datos. Ese software también debe cooperar con los demás sitios en el intercambio de la información sobre las transacciones para hacer posible el procesamiento de las transacciones entre varios sitios.

A diferencia de lo anterior, en las **bases de datos distribuidas heterogéneas,** sitios diferentes puede que utilicen esquemas diferentes y diferente software de gestión de sistemas de bases de datos. Puede que unos sitios no sean conscientes de la existencia de los demás y puede que sólo proporcionen facilidades limitadas para la cooperación en el procesamiento de las transacciones.

Las diferencias en los esquemas suelen constituir un problema importante para el procesamiento de las consultas, mientras que la divergencia del software supone un inconveniente para el procesamiento de transacciones que tengan acceso a varios sitios.

**Ventajas**

- **Naturaleza distribuida:** muchas de estas aplicaciones están distribuidas naturalmente en diferentes lugares. Una compañía puede tener oficinas en varias ciudades, o un banco puede tener múltiples sucursales. Los datos en cada sitio local describen un “minimundo” en ese sitio.

- **Mayor disponibilidad y fiabilidad:** al encontrarse distribuido, se reducen las chances de que el sistema quede totalmente fuera de servicio, o que los datos se pierdan

totalmente.

- **Posibilidad de compartir datos:** es posible controlar los datos y el software localmente en cada sitio. Los usuarios de otros sitios remotos pueden tener acceso a ciertos datos a través del software del SGBDD. Compartimiento controlado de los datos en todo el sistema distribuido.

- **Mejor rendimiento:** las bases de datos de cada uno de los sitios son más pequeñas. Las consultas locales y las transacciones que tienen acceso a un solo sitio son más rápidas. Las transacciones que acceden a más de un sitio se ejecutan en paralelo.

**Desventajas**

- **Complejidad:** Aumento en la complejidad del diseño y en la implementación del sistema. Algunos aspectos de esa complejidad pueden afectar a los usuarios si no se toman las medidas oportunas.

- **Problemas de concurrencia:** se vuelve más difícil establecer controles de concurrencia. Existen técnicas y formas de asegurar que varias transacciones se realicen de forma simultánea, pero aun así la implementación de las mismas es más costosa que un sistema no distribuido.

- **Overhead de procesamiento:** incluso una operación simple puede requerir un largo número de comunicaciones y cálculos adicionales para proveer uniformidad de los datos a través de los sitios.

- **Integridad de datos:** la necesidad de actualizar los datos en múltiples sitios posan problemas de integridad de datos.

- **Overheads por mal manejo de la distribución de datos:** la velocidad de respuesta de las consultas depende mucho de una distribución adecuada de los datos. Una mala distribución conlleva una respuesta lenta para las consultas de usuarios.

**Características**

Desde el punto de vista del usuario, un sistema distribuido deberá ser idéntico a un sistema no distribuido. En términos de SQL, la lógica de las operaciones SELECT, INSERT, UPDATE y DELETE no deberá sufrir cambios.

1. **Autonomía Local:** los sitios de un sistema distribuido deben ser autónomos. Ningún sitio X deberá depender de un sitio Y para su buen funcionamiento. Existencia de un propietario y administración local de los datos.

2. **No dependencia de un sitio central:** no debe haber dependencia de un sitio central “maestro” para obtener un servicio. El sitio central podría ser un cuello de botella. Si el sitio central sufriera un desperfecto, todo el sistema dejaría de funcionar.

3. **Operación continua:** idealmente nunca debería haber necesidad de apagar a propósito el sistema, por ejemplo, para añadir un nuevo sitio o instalar una versión mejorada del DBMS en un sitio ya existente.

4. **Independencia con respecto a la localización**: no debe ser necesario que los usuarios sepan dónde están almacenados físicamente los datos Simplifica los programas de los usuarios. Permite modificar la distribución de los datos dentro de la red.

5. **Independencia respecto a la fragmentación:** 2 clases de fragmentación: Horizontal y Vertical. Los usuarios deberán poder comportarse como si los datos no estuvieran fragmentados en realidad.

6. **Independencia de Réplica:** un sistema maneja réplica de datos si una relación dada se puede representar físicamente mediante varias copias almacenadas en muchos sitios distintos.

7. **Procesamiento distribuido de consultas:** En una consulta distribuida, habrá muchas maneras de trasladar los datos en la red para satisfacer la solicitud. Importancia crucial de la optimización.

8. **Manejo distribuido de transacciones:** Control de Recuperación: el sistema debe asegurar que cada transacción sea atómica (todo o nada).

   Control de Concurrencia: basada en el bloqueo.

9. **Independencia**

   Respecto al Equipo: máquinas diferentes.
   Respecto al Sistema Operativo.
   Respecto a la Red.
   Respecto al DBMS: comunicación mediante SQL.

## Unidad 2: Modelos de Datos

Un modelo de datos es una colección de herramientas conceptuales para la descripción de datos, relaciones entre datos, semántica de los datos y restricciones de consistencia.

El modelo entidad-relación (E-R) es un modelo de datos de alto nivel. Está basado en una percepción de un mundo real que consiste en una colección de objetos básicos, denominados entidades, y de relaciones entre estos objetos.

El modelo relacional es un modelo de menor nivel. Usa una colección de tablas para representar tanto los datos como las relaciones entre los datos. Su simplicidad conceptual ha conducido a su adopción general; actualmente, una vasta mayoría de productos de bases de datos se basan en el modelo relacional. Los diseñadores formulan generalmente el diseño del esquema de la base de datos modelando primero los datos en alto nivel, usando el modelo E-R, y después traduciéndolo al modelo relacional.

**Proceso de diseño de una base de datos**

El primer paso es la **recopilación de requisitos y el análisis**. Durante este paso, los diseñadores de bases de datos entrevistan a los potenciales usuarios de la base de datos para comprender y documentar sus **requisitos en cuanto a datos**. El resultado de este paso es un conjunto por escrito de los requisitos del usuario. Estos requisitos se deben plasmar en un formulario lo más detallado y completo posible. En paralelo al estudio de estos requisitos, resulta útil especificar los requisitos funcionales de la aplicación, que consisten en las operaciones (o transacciones) definidas por el usuario que se aplicarán a la base de datos, incluyendo las recuperaciones y las actualizaciones. En el diseño de software, es frecuente utilizar los diagramas de flujo de datos, diagramas de secuencia, escenarios y otras técnicas para especificar los requisitos funcionales.

Una vez recopilados y analizados todos los requisitos, el siguiente paso es crear un esquema conceptual para la base de datos, mediante un modelo de datos conceptual de alto nivel. Este paso se denomina **diseño conceptual**. El esquema conceptual es una descripción concisa de los requisitos de datos por parte de los usuarios e incluye descripciones detalladas de los tipos de entidades, relaciones y restricciones; se expresan utilizando los conceptos proporcionados por el modelo de datos de alto nivel.

Como estos conceptos no incluyen detalles de implementación, normalmente son más fáciles de entender y se pueden utilizar para comunicar con usuarios no técnicos. El esquema conceptual de alto nivel también se puede utilizar como referencia para garantizar que se han reunido todos los requisitos de datos de los usuarios y que esos requisitos no entran en conflicto. Esta metodología permite a los diseñadores de bases de datos concentrarse en especificar las propiedades de los datos, sin tener que preocuparse por los detalles del almacenamiento. En consecuencia, es más fácil para ellos crear un buen diseño conceptual de bases de datos.

Durante o después del diseño del esquema conceptual, se pueden utilizar las operaciones básicas del modelo de datos para especificar las operaciones de usuario de alto nivel identificadas durante el análisis funcional. Esto también sirve para confirmar que el esquema conceptual satisface todos los requisitos funcionales identificados. Es posible realizar modificaciones en el esquema conceptual si con el esquema inicial no se pueden especificar algunos de los requisitos funcionales.

El siguiente paso del diseño de una base de datos es la implementación real de la misma mediante un DBMS comercial. La mayoría de los DBMSs comerciales actuales utilizan un modelo de datos de implementación (como el modelo de base de datos relacional u objeto-relación), de modo que el esquema conceptual se transforma de modelo de datos de alto nivel en modelo de datos de implementación. Este paso se conoce como diseño lógico o

asignación de modelo de datos; su resultado es un esquema de base de datos en el modelo de datos de implementación del DBMS.

El último paso es la fase de diseño físico, durante la cual se especifican las estructuras de almacenamiento interno, los índices, las rutas de acceso y la organización de los archivos para la base de datos. En paralelo a estas actividades, se diseñan e implementan los programas de aplicación como transacciones de bases de datos correspondientes a las especificaciones de transacción de alto nivel.

**Modelos lógicos basados en registros**

Los modelos lógicos basados en registros se utilizan para describir los datos en los esquemas conceptual y físico. A diferencia de los modelos lógicos basados en objetos, se usan para especificar la estructura lógica global de la BD y para proporcionar una descripción a nivel más alto de la implementación.

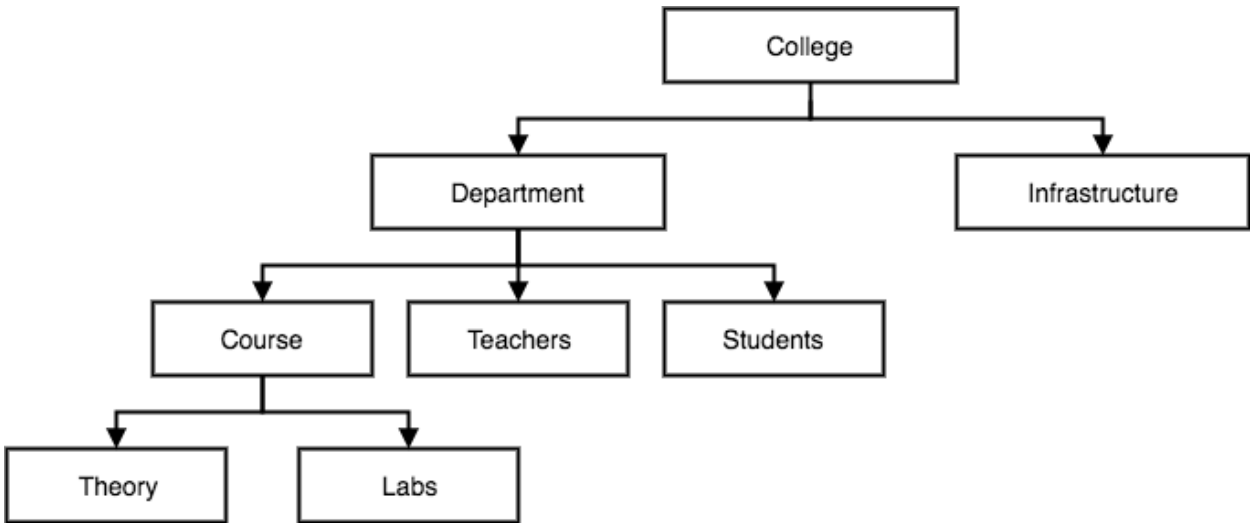
Los modelos basados en registros se llaman así porque la BD está estructurada en registros de formato fijo de varios tipos. Cada tipo de registro define un número fijo de campos, o atributos, y cada campo normalmente es de longitud fija. La estructura más rica de estas BBDD a menudo lleva a registros de longitud variable en el nivel físico.

Los modelos basados en registros no incluyen un mecanismo para la representación directa de código de la BD, en cambio, hay lenguajes separados que se asocian con el modelo para expresar consultas y actualizaciones. Los tres modelos de datos más aceptados son el modelo relacional, de red y jerárquico.

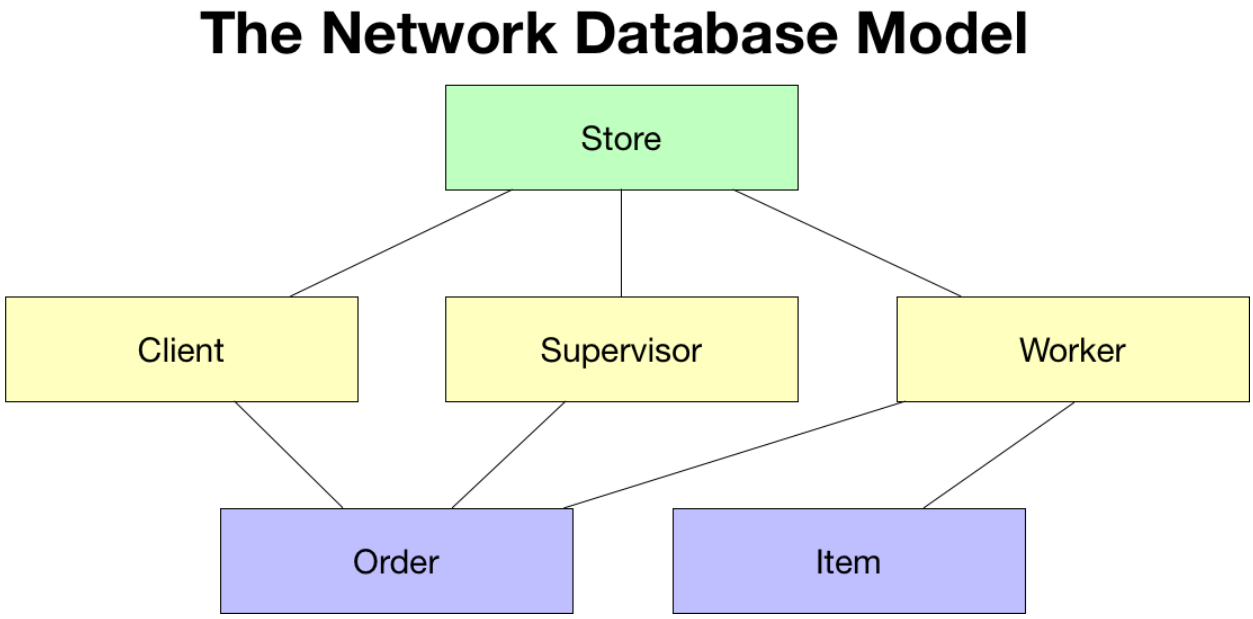
- **Modelo jerárquico:** en un modelo jerárquico, los datos están organizados en una estructura arbórea (dibujada como árbol invertido o raíz), lo que implica que cada registro sólo tiene un padre. Esta estructura permite relaciones 1:N entre los datos.

  Sin embargo, la estructura jerárquica es ineficiente para ciertas operaciones de base de datos cuando el camino completo no se incluye en cada registro. Una limitación del modelo jerárquico es su incapacidad para representar de manera eficiente la redundancia en datos.

  En la relación Padre-hijo: el hijo sólo puede tener un padre pero un padre puede tener múltiples hijos. Los padres e hijos están unidos por enlaces. Todo nodo tendrá una lista de enlaces a sus hijos.




- **Modelo de red:** a diferencia del modelo jerárquico, el modelo de red le permite tener a cada registro múltiples padres y registros hijos. El argumento a favor del modelo de red por encima del modelo jerárquico, es que permitía modelar de forma más natural las relaciones entre entidades.




- **Modelo relacional:** el modelo relacional para la gestión de una base de datos es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos.

  En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario casual de la base

de datos.

La información puede ser recuperada o almacenada por medio de «consultas» que ofrecen una amplia flexibilidad y poder para administrar la información. El lenguaje más común para construir las consultas a bases de datos relacionales es SQL, Structured Query Language, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales. Este modelo considera la base de datos como una colección de relaciones.

De manera simple, una relación representa una tabla, en que cada fila representa una colección de valores que describen una entidad del mundo real. Cada fila se denomina tupla o registro y cada columna campo.

**Modelos lógicos basados en objetos**

Se usan para describir datos en los niveles conceptual y de visión, es decir, con este modelo representamos los datos de tal forma como nosotros los captamos en el mundo real, tienen una capacidad de estructuración bastante flexible y permiten especificar restricciones de datos explícitamente. Existen diferentes modelos de este tipo, pero el más utilizado por su sencillez y eficiencia es el modelo Entidad-Relación.

- **Modelo Entidad-Relación:** este modelo representa a la realidad a través de entidades, que son objetos que existen y que se distinguen de otros por sus características, por ejemplo: un alumno se distingue de otro por sus características particulares como lo es el nombre, o el número de control asignado al entrar a una institución educativa, así mismo, un empleado, una materia, etc.

  Las entidades pueden ser de dos tipos:

  **Tangibles:** son todos aquellos objetos físicos que podemos ver, tocar o sentir.

  **Intangibles:** todos aquellos eventos u objetos conceptuales que no podemos ver, aun sabiendo que existen, por ejemplo: la entidad materia, sabemos que existe, sin embargo, no la podemos visualizar o tocar.

  Las características de las entidades en base de datos se llaman atributos, por ejemplo, el nombre, dirección teléfono, grado, grupo, etc. son atributos de la entidad alumno; Clave, número de seguro social, departamento, etc., son atributos de la entidad empleado. A su vez una entidad se puede asociar o relacionar con más entidades a través de relaciones.

- **Modelo orientado a objetos:** una base de datos orientada a objetos almacena datos complejos y relaciones entre datos directamente, sin asignar filas y columnas, y esto hace que sean más adecuadas para aplicaciones que tratan con datos muy complejos. Los objetos tienen relaciones “muchos a muchos” y son accesibles mediante el uso de punteros. Estos punteros se vinculan a los objetos para establecer relaciones. Otro beneficio adicional de una base de datos orientada a objetos es que puede ser programada con pequeñas diferencias de procedimientos sin afectar a todo el sistema.

## Modelo Entidad-Relación

El modelo ER describe los datos como entidades, relaciones, y atributos.

## Entidades y atributos

El objeto básico representado por el modelo ER es una **entidad**, que es una *cosa* del mundo real con una existencia independiente. Cada entidad tiene **atributos** (propiedades particulares que la describen). Una entidad en particular tendrá un valor para cada uno de sus atributos. Los valores de los atributos que describen cada entidad se convierten en la parte principal de los datos almacenados en la base de datos.

## Atributos compuestos VS atributos simples

Los **atributos compuestos** se pueden dividir en subpartes más pequeñas, que representan atributos más básicos con significados independientes. Los atributos que no son divisibles se denominan **atributos simples** o **atómicos**. Los atributos compuestos pueden formar una jerarquía.

Los atributos son útiles para modelar situaciones en las que un usuario se refiere a veces al atributo compuesto como una unidad, pero otras veces se refiere específicamente a sus componentes. Si se hace referencia al atributo compuesto como un todo, no hay necesidad de subdividirlo en atributos componentes.

## Atributos monovalor y multivalor

La mayoría de los atributos tienen un solo valor para una entidad en particular; dichos atributos reciben el nombre de **monovalor** o **de un solo valor**.

En algunos casos, un atributo puede tener un conjunto de valores para la misma entidad (por ejemplo, un atributo Colores para un coche, o un atributo Licenciaturas para una persona). Dichos atributos se denominan **multivalor**. Un atributo multivalor puede tener límites superior e inferior para restringir el número de valores permitidos para cada entidad individual.

## Atributos almacenados y derivados

En algunos casos, dos (o más) valores de atributo están relacionados (por ejemplo, los atributos Edad y FechaNac de una persona). Para una entidad de persona en particular, el valor de Edad puede determinarse a partir de la fecha actual (el día de hoy) y el valor de FechaNac de esa persona. El atributo Edad se denomina entonces **atributo derivado** y se dice que se ha **derivado** del atributo FechaNac, que es el denominado **atributo almacenado**. Algunos valores de atributo se pueden derivar de *entidades relacionadas*; por ejemplo, un atributo NumEmpleados de una entidad DEPARTAMENTO puede derivarse contando el número de empleados relacionados con (o que trabajan para) ese departamento.

## Valores NULL

En algunos casos, es posible que una entidad en particular no tenga un valor aplicable para un atributo. Para estas situaciones se ha creado un valor especial denominado NULL (nulo). NULL también se puede utilizar cuando no se conoce el valor de un atributo para una entidad en particular.

## Atributos complejos

Los atributos compuestos y multivalor se pueden anidar arbitrariamente. Podemos representar el anidamiento arbitrario agrupando componentes de un atributo compuesto entre paréntesis () y separando los componentes con comas, y mostrando los atributos multivalor entre llaves {}.

## Tipo de entidades y conjuntos de entidades

Un **tipo de entidad** define una *colección* (o *conjunto*) de entidades que tienen los mismos atributos. La colección de todas las entidades de un tipo de entidad en particular de la base de datos en cualquier momento del tiempo se denomina conjunto de entidades; al conjunto de entidades normalmente se hace referencia utilizando el mismo nombre que para el tipo de entidad.

Un tipo de entidad se representa en los diagramas ER como un rectángulo con el nombre del tipo de entidad en su interior. Los nombres de los atributos se encierran en óvalos y están unidos a su tipo de entidad mediante líneas rectas. Los atributos compuestos están unidos a sus atributos componente mediante líneas rectas. Los atributos multivalor se muestran en óvalos dobles.

## Atributos clave de un tipo de entidad

Una restricción importante de las entidades de un tipo de entidad es la **clave** o **restricción de unicidad** de los atributos. Un tipo de entidad normalmente tiene un atributo cuyos valores son distintos para cada entidad individual del conjunto de entidades. Dicho atributo se denomina **atributo clave**, y sus valores se pueden utilizar para identificar cada entidad sin lugar a dudas.

En ocasiones, una clave está formada por varios atributos juntos, lo que da a entender que la *combinación* de los valores de atributo debe ser distinta para cada entidad. Si un conjunto de atributos posee esta propiedad, la forma correcta de representar esto en el modelo ER que aquí describimos es definir un *atributo compuesto* y designarlo como atributo clave del tipo de entidad. Una clave compuesta debe ser mínima; es decir, en el atributo compuesto se deben incluir todos los atributos componente para tener una propiedad de unicidad. En una clave no se deben incluir atributos superfluos. En la notación diagramática ER, cada atributo clave tiene su nombre **subrayado** dentro del óvalo.

Algunos tipos de entidad tienen *más de un atributo clave*. Un tipo de entidad también puede *carecer de clave*, en cuyo caso se denomina *tipo de entidad débil.*

## Conjunto de valores (dominios) de atributos

Cada atributo simple de un tipo de entidad está asociado con un **conjunto de valor** (o **dominio** de valores), que especifica el conjunto de los valores que se pueden asignar a ese atributo por cada entidad individual.

Los conjuntos de valores no se muestran en los diagramas ER; normalmente se especifican mediante los **tipos de datos** básicos disponibles en la mayoría de los lenguajes de programación, como entero, cadena, booleano, flotante, tipo enumerado, subrango, etcétera.

## Restricciones en los tipos de relaciones

- **Razones de cardinalidad para las relaciones binarias:** La razón de cardinalidad de una relación binaria especifica el número *máximo* de instancias de relación en las que una entidad puede participar. Las posibles razones de cardinalidad para los tipos de relación binaria son 1:1, 1:N, N:1 y M:N. Las razones de cardinalidad de las relaciones binarias se representan en los diagramas ER mediante 1, M y N en los rombos

- **Restricciones de participación y dependencia de existencia:** La restricción de participación especifica si la existencia de una entidad depende de si está relacionada con otra entidad a través de un tipo de relación. Esta restricción especifica el número *mínimo* de instancias de relación en las que puede participar cada entidad, y en ocasiones recibe el nombre de **restricción de cardinalidad mínima**. Hay dos tipos de restricciones de participación, total y parcial.

  La participación total indica que una entidad solo puede existir si cumple cierta relación. Por dar un ejemplo dentro de una empresa, un EMPLEADO solo puede existir si está asignado a un DEPARTAMENTO. En caso contrario no puede existir.

  La participación parcial indica que no necesariamente la existencia de la entidad dependa de dicha relación.

  En los diagramas ER, la participación total (o dependencia existente) se muestra como una línea doble que conecta el tipo de entidad participante con la relación, mientras que las participaciones parciales se representan mediante una línea sencilla.

## Atributos de los tipos de relación

Los tipos de relación también pueden tener atributos, parecidos a los de los tipos de entidad. Por ejemplo, para registrar el número de horas por semana que un empleado trabaja en un proyecto en particular, podemos incluir un atributo Horas para el tipo de relación TRABAJA_EN.

Los atributos de los tipos de relación 1:1 o 1:N se pueden trasladar a uno de los tipos de entidad participantes. Por ejemplo, el atributo FechaInicio para la relación ADMINISTRA puede ser un atributo de EMPLEADO o DEPARTAMENTO, aunque conceptualmente pertenece a ADMINISTRA. Esto se debe a que ADMINISTRA es una relación 1:1, por lo que cada entidad departamento o empleado participa *a lo sumo en una* instancia de relación. Por tanto, el valor del atributo FechaInicio se puede determinar por separado, bien mediante la entidad departamento participante, bien mediante la entidad empleado (director) participante.

En el caso de un tipo de relación 1:N, un atributo de relación *sólo* se puede migrar al tipo de entidad que se encuentra en el lado N de la relación. En la Figura 3.9, por ejemplo, si la relación TRABAJA_PARA también tiene un atributo FechaInicio que indica la fecha en que un empleado empezó a trabajar para un departamento, este atributo se puede incluir como un atributo de EMPLEADO. Esto se debe a que cada empleado trabaja sólo para un departamento y, por tanto, participa en un máximo de una instancia de relación en TRABAJA_PARA. En los tipos de relación 1:1 y 1:N, la decisión sobre dónde debe colocarse un atributo de relación

(como un atributo de tipo de relación o como un atributo de un tipo de entidad participante) la determina subjetivamente el diseñador del esquema.

Para los tipos de relación M:N, algunos atributos pueden determinarse mediante la *combinación de entidades participantes* en una instancia de relación, no mediante una sola relación. Dichos atributos *deben especificarse como atributos de relación*. Un ejemplo de esto es el atributo Horas de la relación M:N TRABAJA_EN; el número de horas que un empleado trabaja en un proyecto viene determinado por una combinación empleado-proyecto, y no separadamente por cualquiera de estas entidades.

## Tipos de entidades débiles

Los tipos de entidad que no tienen atributos clave propios se denominan **tipos de entidad débiles**. En contraposición, los **tipos de entidad regulares** que tienen un atributo clave se denominan **tipos de entidad fuertes.** Las entidades que pertenecen a un tipo de entidad débil se identifican como relacionadas con entidades específicas de otro tipo de entidad en combinación con uno de sus valores de atributo. Podemos llamar a este otro tipo de entidad **tipo de entidad identificado** o **propietario**, y al tipo de relación que relaciona un tipo de entidad débil con su propietario lo podemos llamar **relación identificativa** del tipo de entidad débil.

Un tipo de entidad débil siempre tiene una ***restricción de participación total*** (dependencia de existencia) respecto a su relación identificativa, porque una entidad débil no puede identificarse sin una entidad propietaria. No obstante, no toda dependencia de existencia produce un tipo de entidad débil.

Un tipo de entidad débil normalmente tiene una **clave parcial**, que es el conjunto de atributos que pueden identificar sin lugar a dudas las entidades débiles que están *relacionadas con la misma entidad propietaria*. En el peor de los casos, la clave parcial será un atributo compuesto por todos los *atributos de la entidad débil*.

En los diagramas ER, tanto el tipo de la entidad débil como la relación identificativa, se distinguen rodeando sus cuadros y rombos mediante unas líneas. **El atributo de clave parcial aparece subrayado con una línea discontinua o punteada.**

El diseñador de la base de datos toma la decisión del tipo de representación que hay que usar. Uno de los criterios que puede utilizar es elegir la representación del tipo de entidad débil si hay muchos atributos. Si la entidad débil participa independientemente en los tipos de relación de otra forma que su tipo de relación identificativa, entonces no debe modelarse como un atributo complejo.

## Asignación correcta de nombre a las construcciones del esquema

Hay que elegir nombres que transmitan, lo mejor posible, los significados de las distintas estructuras del esquema. Hemos optado por *nombres en singular* para los tipos de entidad, en lugar de nombres plurales, porque el nombre de un tipo de entidad se aplica a cada entidad individual que pertenece a ese tipo de entidad. En nuestros diagramas ER utilizaremos la convención de que los nombres de los tipos de entidades y de los tipos de relación se escriben

en mayúsculas, los nombres de los atributos se escriben con la primera letra en mayúscula, y los nombres de los papeles en minúsculas.

Otra consideración en cuanto a la denominación implica la elección de nombres de relación binaria para que el diagrama ER del esquema se pueda leer de izquierda a derecha y de arriba abajo.

## Relaciones ternarias

Las relaciones ternarias se grafican con un triángulo en lugar de un rombo, y se conecta a las tres entidades que intervienen en la relación. En general las relaciones ternarias nos ofrecen un tipo de información diferente que tres tipos de relaciones binarias. Nos dan información más concreta.

Para saber si tenemos necesidad de una relación ternaria debemos plantearnos lo siguiente: La acción necesita de uno o más objetos de dos entidades para realizar la acción sobre una tercera.

Para definir la cardinalidad correctamente tenemos que siempre tomar dos elementos de dos entidades y evaluar cuántos le corresponden de la tercera. Ahí se puede definir si es cero, uno o muchos en la tercera entidad. Luego debemos pararnos en otras dos de las tres entidades y evaluar la restante. Definir si es cero uno o muchos en la restante y repetir el proceso una tercera y última vez.

## Jerarquías o herencias

Una jerarquía o herencia aparece en nuestro modelo cuando dos o más entidades tienen ciertos atributos en común, y otros tantos específicos a cada una de ellas.

Muchos atributos son comunes a ambas entidades, por lo que podemos idear una entidad padre, también conocida como supraentidad o superentidad, que agrupe a estos atributos. Normalmente, se define un nombre para esta nueva entidad, común a ambas. Luego tendremos las entidades específicas, también conocidas como subentidades, quienes contendrán únicamente los atributos específicos.

Una característica importante de las jerarquías es que las subentidades no poseen atributos identificadores, sino que los “heredan” de la supraentidad. Es por ello, que al momento de jerarquizar, deberán seleccionarse en la supraentidad únicamente aquellos atributos que sean clave.

- **Solapamiento:** Una vez que tenemos definidas la supraentidad y las subentidades, debemos pensar si las subentidades son solapables o no. En otras palabras, si una subentidad puede ser a su vez otra subentidad diferente.
  En el caso de los Medios de transporte, esto es imposible, ya que un vehículo específico, o es Auto o es Omnibus (nunca ambos). Para estos casos, decimos que la jerarquía es **exclusiva** o **sin solapamiento**. Esta restricción de las jerarquías se simboliza con un arco, según puede verse en el ejemplo.

  Un ejemplo de jerarquía **inclusiva** o **con solapamiento** podría ser el caso de tener una supraentidad Persona y las subentidades Alumno y Docente. Dado que la misma persona puede ser Alumno y Docente a la vez, consideramos que en este caso puede

haber solapamiento. Estos casos no requieren adicionar nada al gráfico, ya que son el caso menos restrictivo, por lo que simplemente no dibujamos el arco.

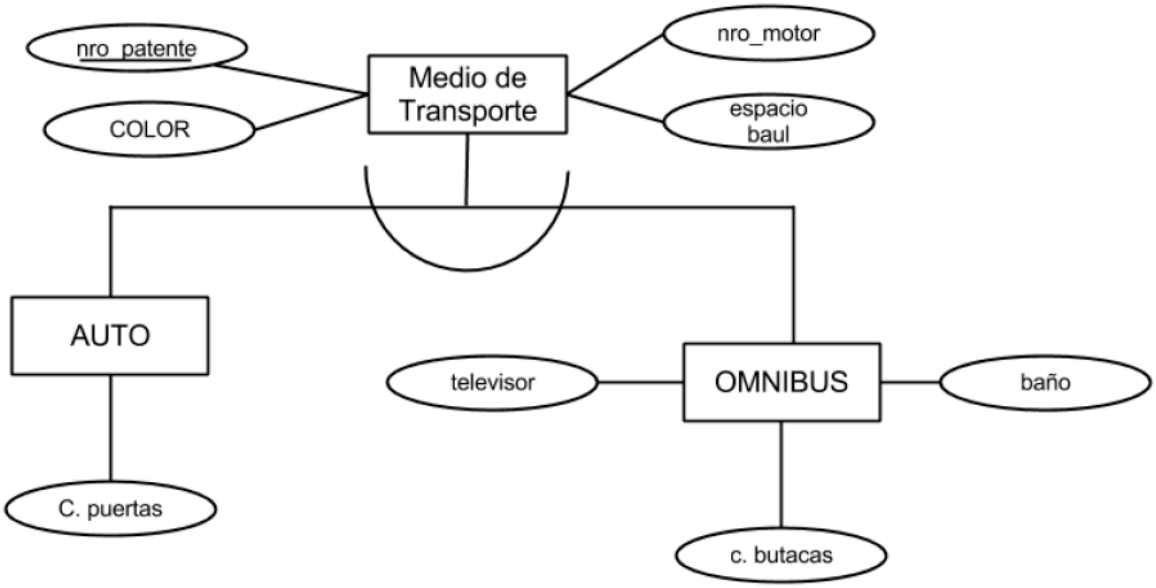



Figura 51. Ejemplo de jerarquía exclusiva o sin solapamiento

- **Partición:** La siguiente restricción que debe plantearse en una jerarquía es la partición. Una jerarquía se la considera de partición total cuando toda supraentidad debe tener al menos una subentidad que la especifica. En cambio, en una partición parcial, una supra entidad podría llegar a existir sin subentidad alguna. Es importante entender que este concepto es independiente del solapamiento.

  Para nuestro ejemplo, un medio de transporte tiene que ser auto u ómnibus. No hay conceptualmente medios de transporte que no sean alguno de estos dos. Por ende, hablamos de una partición total. Simbolizamos esta restricción con una doble línea hacia la supraentidad, según vemos en el ejemplo.

  Un caso de partición parcial podría ser dado por la supraentidad Empleado y las subentidades Administrativo y Programador, cada una de estas últimas con sus atributos específicos. Podríamos llegar a tener en nuestra empresa otros empleados que no sean ni administrativos ni programadores (ej: limpieza) por lo que existirían instancias de entidades Empleado sin contener instancias en alguna subentidad.

  Al igual que en el caso de solapamiento, dado que la partición parcial es menos restrictiva que la total, simplemente no se simboliza este caso en el diagrama. Se traza una línea simple en lugar de doble.

**Atributo de jerarquía discriminante**

Si nos encontramos en el caso más restrictivo posible de jerarquía (partición total, sin

solapamiento), podemos optar por la definición de un atributo de jerarquía o discriminante.

Este atributo permitirá, desde el punto de vista de la supraentidad, determinar la subentidad que la especifica. Este atributo puede ser uno ya existente en la supraentidad, o bien puede definirse en este momento.

Dado que nuestro ejemplo posee este tipo de restricciones, definiremos un atributo que permita identificar el tipo de medio de transporte que se especifica. Este atributo se dibuja con un símbolo particular (cápsula) entre la supraentidad y las subentidades, según podemos apreciar en el ejemplo.

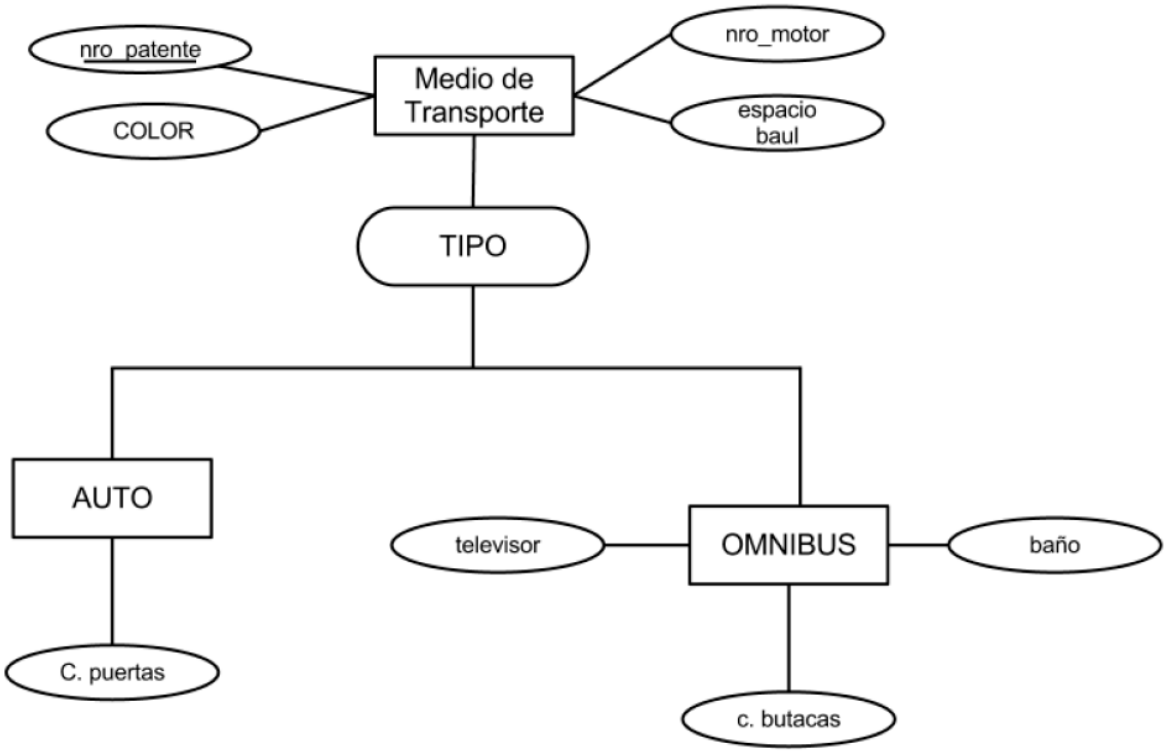
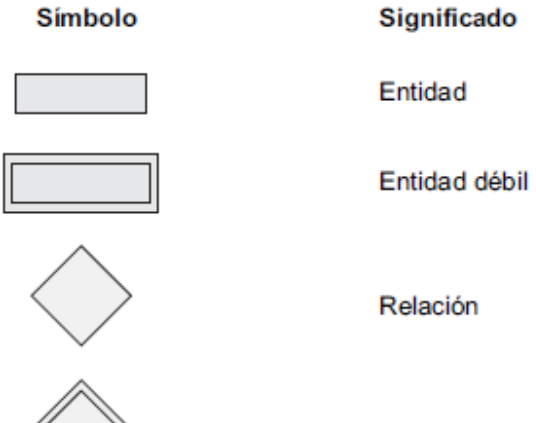



Figura 51. Ejemplo de atributo de jerarquía o discriminante. En este caso, el atributo "Tipo" permite reconocer la subentidad correspondiente al medio de transporte.

Cabe destacar que, como este concepto aplica unicamente para esta forma de restricción (partición total, sin solapamiento), pueden omitirse los símbolos particulares de las restricciones ya que queda determinado por el del atributo.

**Figura 3.14.** Resumen de la notación para los diagramas ER.



## Paso de DER a MR

**Ternarias**

Al igual que las relaciones binarias N:N, las relaciones ternarias generan una relación nueva en el modelo MR, independientemente de la cardinalidad que posean sus conexiones.

Las claves de dicha relación resultante estarán definidas por la cardinalidad. Si es N:N:N, todas las claves que vendrán de afuera serán claves principales (y foráneas).

Si es N:N:1, solo serán claves principales aquellos atributos que vienen de las entidades de cardinalidad N.

Si es N:1:1 se debe colocar como clave principal aquella que viene de la entidad con “N”, y luego se debe optar por elegir una de las claves de los atributos con cardinalidad “1” para finalizar la clave principal.

En el caso de cardinalidad 1:1:1, se debe elegir dos conjuntos de clave proveniente de las entidades con cardinalidad “1”.

Notar que todas las claves son foráneas, incluso aquella que no es parte de la clave principal. Y siempre aparecerán las 3 claves foráneas en una relación ternaria.

**Entidades Débiles**

La clave de la entidad propietaria se trae como clave foránea, y en conjunto con la clave parcial de la entidad débil, conforman la clave primaria.

También se deben incluir los atributos simples pertenecientes a dicha entidad débil.

**Jerarquías / Herencias**

Hay varias formas, pero la más fácil y la que satisface todos los tipos de jerarquías (totales o parciales, disjuntas o solapadas) es en la que la clave principal de la superentidad es heredada por todas las subentidades. **Esta clave es primaria para cada subentidad. Solamente primaria, no es foránea.**