

## BLOQUE II

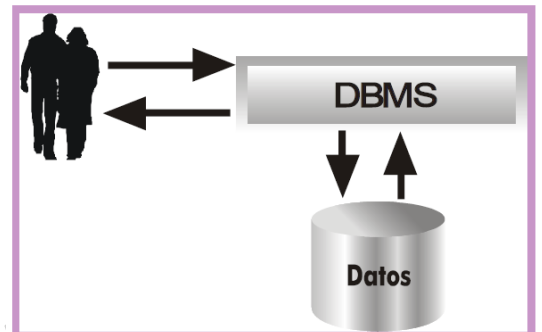
## TEMA 05

### Sistemas de gestión de bases de datos relacionales, orientados a objetos y NoSQL: características y componentes

#### 1. SISTEMAS DE GESTION DE BASES DE DATOS (SGBD)

Un sistema gestor de bases de datos o SGBD (aunque se suele utilizar más a menudo las siglas DBMS procedentes del inglés, Data Base Management System) es el software que permite a los usuarios procesar, describir, administrar y recuperar los datos almacenados en una base de datos.

En estos Sistemas se proporciona un conjunto coordinado de programas, procedimientos y lenguajes que permiten a los distintos usuarios realizar sus tareas habituales con los datos, garantizando además la seguridad de los mismos.



El éxito del SGBD reside en mantener la seguridad e integridad de los datos. Lógicamente tiene que proporcionar herramientas a los distintos usuarios. Entre las herramientas que proporciona están:

- ♦ **Herramientas para la creación y especificación de los datos.** Así como la estructura de la base de datos.
- ♦ **Herramientas para administrar y crear la estructura** física requerida en las unidades de almacenamiento.
- ♦ **Herramientas para la manipulación de los datos** de las bases de datos, para añadir, modificar, suprimir o consultar datos.
- ♦ **Herramientas de recuperación** en caso de desastre
- ♦ **Herramientas para la creación de copias de seguridad**
- ♦ **Herramientas para la gestión de la comunicación** de la base de datos
- ♦ **Herramientas para la creación de aplicaciones** que utilicen esquemas externos de los datos
- ♦ **Herramientas de instalación** de la base de datos
- ♦ **Herramientas para la exportación e importación de datos**

## BLOQUE II

## TEMA 05

### Características

- Reducen la cantidad de información necesaria
- Evitan inconsistencias en los datos
- Proporcionan seguridad tanto de acceso a los datos como de salvaguarda de los mismos.
- Proporcionan independencia de los datos, respecto de las aplicaciones.

### NIVELES DE ABSTRACCIÓN

En cualquier sistema de información se considera que se pueden observar los datos desde dos puntos de vista:

- Vista lógica o externa. Esta es la visión de los datos que poseen los usuarios del Sistema de Información.
- Vista física o interna. Esta es la forma en la que realmente están almacenados los datos.

En un sistema orientado a procesos, los usuarios ven los datos desde las aplicaciones creadas por los programadores. Esa vista pueden ser formularios, informes visuales o en papel,... Pero la realidad física de esos datos, tal cual se almacenan en los discos queda oculta. Esa visión está reservada a los administradores.

En el caso de los Sistemas de Base de datos, se añade una tercera vista, que es la vista **conceptual**. Esa vista se sitúa entre la física y la externa. Se habla pues en Bases de datos de la utilización de tres esquemas para representar los datos.

#### • Esquema físico o interno

Representa la forma en la que están almacenados los datos. Esta visión sólo la requiere el administrador/a. El administrador la necesita para poder gestionar más eficientemente la base de datos. En este esquema se habla de archivos, directorios o carpetas, unidades de disco, servidores, etc.

#### • Esquema Conceptual

Se trata de un esquema teórico de los datos en el que figuran organizados en estructuras reconocibles del mundo real y en el que también aparece la forma de relacionarse los datos. Este esquema es el paso que permite modelar un problema real a su forma correspondiente en el ordenador.

Este esquema es la base de datos de todos los demás. Como se verá más adelante es el primer paso a realizar al crear una base de datos. El esquema conceptual lo realiza diseñadores/as o analistas.

## BLOQUE II

## TEMA 05

### • Esquema Lógico o Externo

Se trata de la visión de los datos que poseen los usuarios finales. Esa visión es la que obtienen a través de las aplicaciones. Las aplicaciones creadas por los desarrolladores abstraen la realidad conceptual de modo que el usuario no conoce las relaciones entre los datos, como tampoco conoce todos los datos que realmente se almacenan.

Realmente cada aplicación produce un esquema externo diferente (aunque algunos pueden coincidir) o vista de usuario. El conjunto de todas las vistas de usuario es lo que se denomina **esquema externo global**.

### 1.1 FUNCIONES. COMPONENTES DE LOS SGBD

#### FUNCIONES

Los SGBD tienen que realizar tres tipos de funciones para ser considerados válidos.

**Función de descripción o definición**  $\equiv$  Permite al diseñador de la base de datos crear las estructuras apropiadas para integrar adecuadamente los datos. Esta función es la que permite definir las tres estructuras de la base de datos (relacionadas con sus tres esquemas).

1. Estructura interna
2. Estructura conceptual
3. Estructura externa

Esta función se realiza mediante el lenguaje de descripción de datos o **DDL**. Mediante ese lenguaje:

- Se definen las estructuras de datos
- Se definen las relaciones entre los datos
- Se definen las reglas que han de cumplir los datos

**Función de manipulación**  $\equiv$  Permite modificar y utilizar los datos de la base de datos. Se realiza mediante el lenguaje de modificación de datos o **DML**. Mediante ese lenguaje se puede:

- Añadir datos
- Eliminar datos
- Modificar datos
- Buscar datos

Actualmente se suele distinguir aparte la función de buscar datos en la base de datos (función de consulta). Para lo cual se proporciona un lenguaje de consulta de datos o **DQL**.

## BLOQUE II

## TEMA 05

**Función de control**  $\equiv$  Mediante esta función los administradores poseen mecanismos para proteger las visiones de los datos permitidas a cada usuario, además de proporcionar elementos de creación y modificación de esos usuarios.

Se suelen incluir aquí las tareas de copia de seguridad, carga de ficheros, auditoria, protección ante ataques externos, configuración del sistema,...

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

### COMPONENTES

**Motor de la base de datos**  $\equiv$  Acepta peticiones lógicas de los otros subsistemas del SGBD, las convierte en su equivalente físico y accede a la base de datos y diccionario de datos en el dispositivo de almacenamiento.

**Subsistema de definición de datos**  $\equiv$  Ayuda a crear y mantener el diccionario de datos y define la estructura del fichero que soporta la base de datos.

**Subsistema de manipulación de datos**  $\equiv$  Ayuda al usuario a añadir, cambiar y borrar información de la base de datos y la consulta para extraer información. El subsistema de manipulación de datos suele ser la interfaz principal del usuario con la base de datos. Permite al usuario especificar sus requisitos de la información desde un punto de vista lógico.

**Subsistema de generación de aplicaciones**  $\equiv$  Contiene utilidades para ayudar a los usuarios en el desarrollo de aplicaciones. Usualmente proporciona pantallas de entrada de datos, lenguajes de programación e interfaces.

**Subsistema de administración**  $\equiv$  Ayuda a gestionar la base de datos ofreciendo funcionalidades como almacenamiento y recuperación, gestión de la seguridad, optimización de preguntas, control de concurrencia y gestión de cambios.

El proceso que realiza un SGBD está en realidad formado por varias capas que actúan como interfaces entre el usuario y los datos. Desde esta óptica para llegar a los datos hay que pasar una serie de capas que desde la parte más externa poco a poco van entrando más en la realidad física de la base de datos.



**BLOQUE II****TEMA 05**

**Facilidades de usuario**  $\equiv$  Son las herramientas que proporciona el SGBD a los usuarios para permitir un acceso más sencillo a los datos. Actúan de interfaz entre el usuario y la base de datos, y son el único elemento que maneja el usuario.

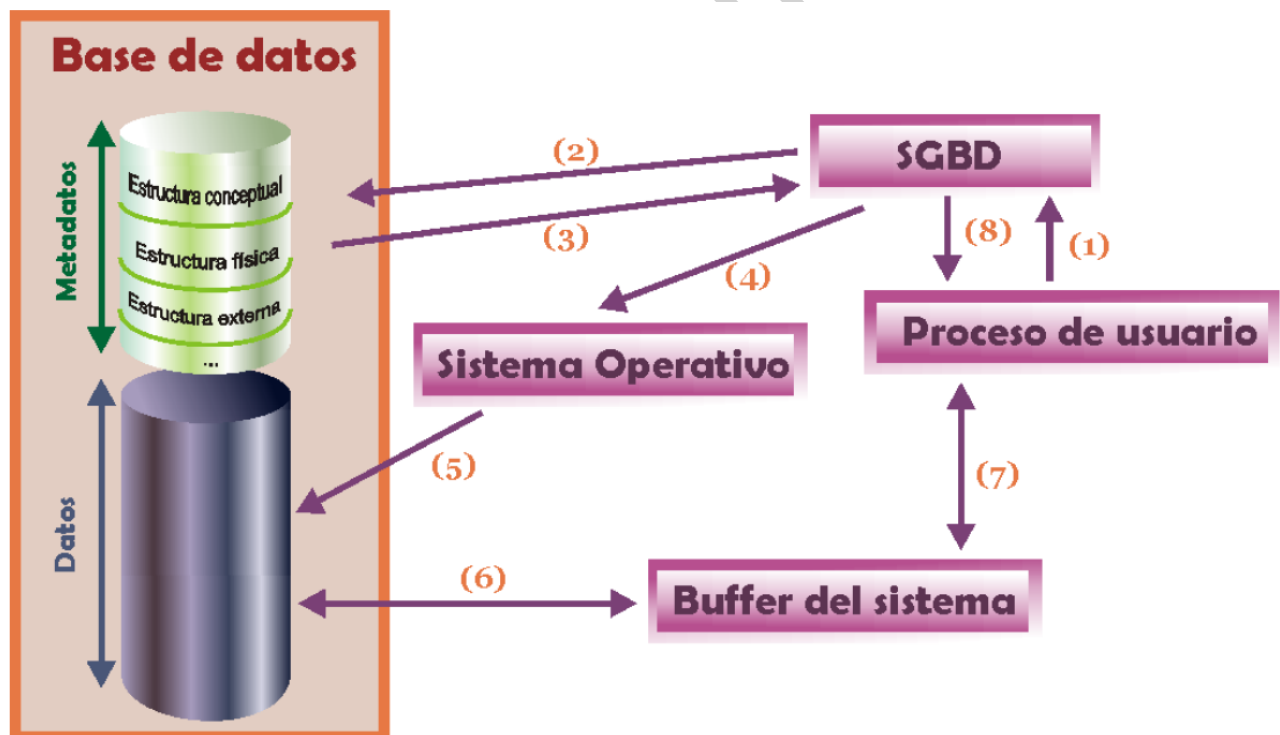
**Capa de acceso a datos**  $\equiv$  La capa de acceso a datos es la que permite comunicar a las aplicaciones de usuario con el diccionario de datos a través de las herramientas de gestión de datos que incorpore el SGBD.

**Diccionario de datos**  $\equiv$  Se trata del elemento que posee todos los metadatos. Gracias a esta capa las solicitudes de los clientes se traducen en instrucciones que hacen referencia al esquema interno de la base de datos.

**Núcleo**  $\equiv$  El núcleo de la base de datos es la encargada de traducir todas las instrucciones requeridas y prepararlas para su correcta interpretación por parte del sistema. Realiza la traducción física de las peticiones.

**Sistema operativo**  $\equiv$  Es una capa externa al software SGBD pero es la única capa que realmente accede a los datos en sí.

El esquema siguiente presenta el funcionamiento típico de un SGBD, y muestra la comunicación entre un proceso de usuario que desea acceder a los datos y el SGBD



## BLOQUE II

## TEMA 05

- 1) El proceso lanzado por el usuario llama al SGBD indicando la porción de la base de datos que se desea tratar
- 2) El SGBD traduce la llamada a términos del esquema lógico de la base de datos. Accede al esquema lógico comprobando derechos de acceso y la traducción física (normalmente los metadatos se guardan en una zona de memoria global y no en el disco)
- 3) El SGBD obtiene el esquema físico
- 4) El SGBD traduce la llamada a los métodos de acceso del Sistema Operativo que permiten acceder realmente a los datos requeridos
- 5) El Sistema Operativo accede a los datos tras traducir las órdenes dadas por el SGBD
- 6) Los datos pasan del disco a una memoria intermedia o buffer. En ese buffer se almacenarán los datos según se vayan recibiendo
- 7) Los datos pasan del buffer al área de trabajo del usuario (ATU) del proceso del usuario. Los pasos 6 y 7 se repiten hasta que se envíe toda la información al proceso de usuario
- 8) En el caso de que haya errores en cualquier momento del proceso, el SGBD devuelve indicadores en los que manifiesta si ha habido errores o advertencias a tener en cuenta. Esto se indica al área de comunicaciones del proceso de usuario. Si las indicaciones son satisfactorias, los datos de la ATU serán utilizables por el proceso de usuario.

### 1.2 MODELOS DE DATOS

El organismo ANSI ha marcado la referencia para la construcción de SGBD. El modelo definido se basa en estudios anteriores en los que se definían tres niveles de abstracción necesarios para gestionar una base de datos. ANSI profundiza más en esta idea y define cómo debe ser el proceso de creación y utilización de estos niveles.

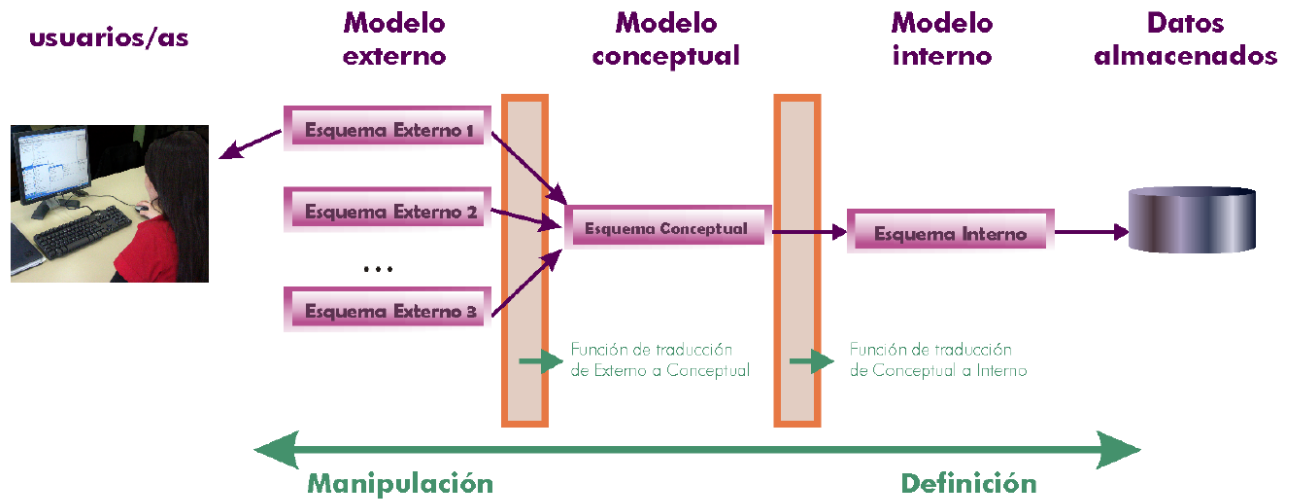
En el modelo ANSI se indica que hay tres modelos: externo, conceptual e interno. Se entiende por modelo, el conjunto de normas que permiten crear esquemas (diseños de la base de datos).

Los esquemas externos reflejan la información preparada para el usuario final, el esquema conceptual refleja los datos y relaciones de la base de datos y el esquema interno la preparación de los datos para ser almacenados.

El esquema conceptual contiene la información lógica de la base de datos. Su estructuración y las relaciones que hay entre los datos.

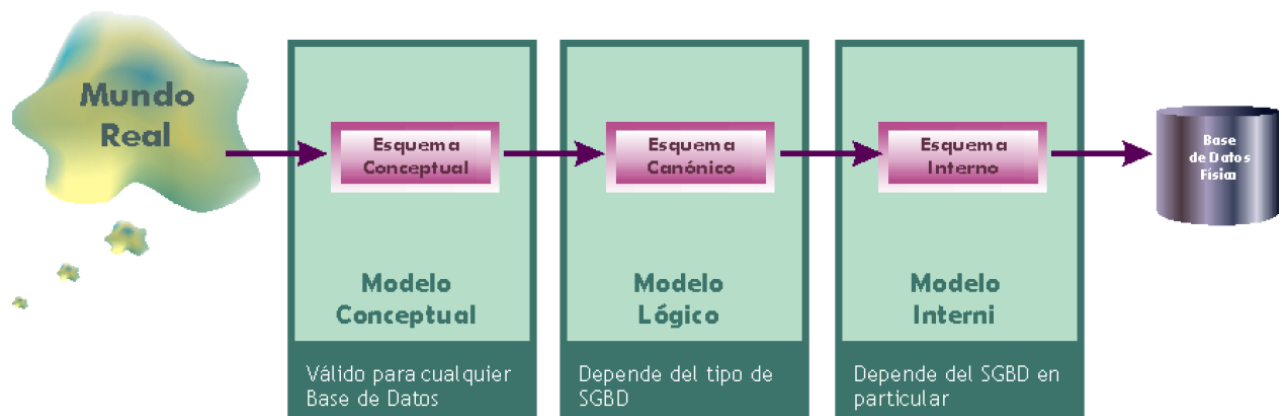
**BLOQUE II****TEMA 05**

El esquema interno contiene información sobre cómo están almacenados los datos en disco. Es el esquema más cercano a la organización real de los datos.



Cada SGBD puede utilizar un modelo diferente para los datos. Por lo que hay modelos conceptuales diferentes según que SGBD utilicemos.

No obstante existen modelos lógicos comunes, ya que hay SGBD de diferentes tipos. En la realidad el modelo ANSI se modifica para que existan dos modelos internos: el modelo lógico (referido a **cualquier** SGBD de ese tipo) y el modelo propiamente interno (aplicable **sólo a un SGBD** en particular). De hecho en la práctica al definir las bases de datos desde el mundo real hasta llegar a los datos físicos se pasa por los siguientes esquemas:



Por lo tanto la diferencia entre los distintos SGBD está en que proporcionan **diferentes modelos lógicos**.

## BLOQUE II

## TEMA 05

### Diferencias entre el modelo lógico y el conceptual

- ♦ El modelo conceptual es independiente del DBMS que se vaya a utilizar. El lógico depende de un tipo de SGBD en particular
- ♦ El modelo lógico está más cerca del modelo físico, el que utiliza internamente el ordenador
- ♦ El modelo conceptual es el más cercano al usuario, el lógico es el encargado de establecer el paso entre el modelo conceptual y el modelo físico del sistema.

Ejemplos de modelos conceptuales:

- Modelo Entidad Relación
- Modelo RM/T
- Modelos semánticos

Ejemplos de modelos lógicos: Comentados posteriormente

- Modelo relacional
- Modelo Red (Codasyl)
- Modelo Jerárquico

**MODELO JERARQUICO**  $\equiv$  Era utilizado por los primeros SGBD. Se le llama también modelo en árbol debido a que utiliza una estructura en árbol para organizar los datos.

La información se organiza con un jerarquía en la que la relación entre las entidades de este modelo siempre es del tipo padre / hijo. De esta forma hay una serie de nodos que contendrán atributos y que se relacionarán con nodos hijos de forma que puede haber más de un hijo para el mismo padre (pero un hijo sólo tiene un padre), es decir, permite **relaciones 1:N** (uno a varios) pero **no permite relaciones N:M** (varios a varios).

Los datos de este modelo se almacenan en estructuras lógicas llamadas segmentos. Los segmentos se relacionan entre sí utilizando arcos.

La forma visual de este modelo es de árbol invertido, en la parte superior están los padres y en la inferior los hijos. Este esquema está en absoluto desuso ya que no es válido para modelar la mayoría de problemas de bases de datos.

Actualmente las bases de datos jerárquicas más utilizadas son IMS de IBM y el **Registro de Windows** de Microsoft.

**MODELO EN RED**  $\equiv$  OBSOLETO. El modelo en red organiza la información en registros (también llamados nodos) y enlaces. En los registros se almacenan los datos, mientras que los enlaces permiten relacionar estos datos. Las bases de datos en red son parecidas a las jerárquicas sólo que en ellas puede haber más de un padre.

En este modelo se pueden representar perfectamente cualquier tipo de relación entre los datos (**permiten relaciones N:M**), pero hace muy complicado su manejo.



## BLOQUE II

## TEMA 05

**MODELO RELACIONAL**  $\equiv$  En este modelo los datos se organizan en tablas cuyos datos se relacionan. Considera la base de datos como una colección de relaciones. De manera simple, una relación representa una tabla que no es más que un conjunto de filas, cada fila es un conjunto de campos y cada campo representa un valor que interpretado describe el mundo real. Cada fila también se puede denominar tupla o registro y a cada columna también se le puede llamar campo o atributo.

**MODELO ORIENTADO A OBJETOS**  $\equiv$  Desde la aparición de la programación orientada a objetos (POO u OOP) se empezó a pensar en bases de datos adaptadas a estos lenguajes. La programación orientada a objetos permite cohesionar datos y procedimientos, haciendo que se diseñen estructuras que poseen datos (atributos) en las que se definen los procedimientos (operaciones) que pueden realizar con los datos. En las bases orientadas a objetos se utiliza esta misma idea.

A través de este concepto se intenta que estas bases de datos consigan arreglar las limitaciones de las relacionales. Por ejemplo el problema de la herencia (el hecho de que no se puedan realizar relaciones de herencia entre las tablas), tipos definidos por el usuario, disparadores (triggers) almacenables en la base de datos, soporte multimedia, etc.

Se supone que son las bases de datos de tercera generación (la primera fue las bases de datos en red y la segunda las relacionales), lo que significa que el futuro parece estar a favor de estas bases de datos. Pero siguen sin reemplazar a las relacionales, aunque son el tipo de base de datos que más está creciendo en los últimos años.

Su modelo conceptual se suele diseñar en UML y el lógico actualmente en ODMG (Object Data Management Group, grupo de administración de objetos de datos, organismo que intenta crear estándares para este modelo).

**MODELO OBJETO-RELACIONAL**  $\equiv$  Tratan de ser un híbrido entre el modelo relacional y el orientado a objetos. El problema de las bases de datos orientadas a objetos es que requieren reinvertir capital y esfuerzos de nuevo para convertir las bases de datos relacionales en bases de datos orientadas a objetos. En las bases de datos objeto-relacionales se intenta conseguir una compatibilidad relacional dando la posibilidad de integrar mejoras de la orientación a objetos.

Estas bases de datos se basan en el estándar SQL 99. En ese estándar se añade a las bases relacionales la posibilidad de almacenar procedimientos de usuario, triggers, tipos definidos por el usuario, consultas recursivas, bases de datos OLAP, tipos LOB,...

Las últimas versiones de la mayoría de las clásicas grandes bases de datos relacionales (Oracle, SQL Server, Informix, ...) son objeto relacionales.

## BLOQUE II

## TEMA 05

### 2. SGBD RELACIONALES

Edgar Frank **Codd** definió las bases del modelo relacional a finales de los 60. Perseguía estos objetivos con su modelo:

**Independencia física.** La forma de almacenar los datos, no debe influir en su manipulación lógica. Si la forma de almacenar los datos cambia, los usuarios no tienen siquiera porque percibirlo y seguirán trabajando de la misma forma con la base de datos. Esto permite que los usuarios y usuarias se concentren en qué quieren consultar en la base de datos y no en cómo está realizada la misma.

**Independencia lógica.** Las aplicaciones que utilizan la base de datos no deben ser modificadas porque se modifiquen elementos de la base de datos. Es decir, añadir, borrar y suprimir datos, no influye en las vistas de los usuarios. De una manera más precisa, gracias a esta independencia el esquema externo de la base de datos es realmente independiente del modelo lógico.

**Flexibilidad.** La base de datos ofrece fácilmente distintas vistas en función de los usuarios y aplicaciones.

**Uniformidad.** Las estructuras lógicas siempre tienen una única forma conceptual (las tablas).

**Sencillez.** Facilidad de manejo (algo cuestionable, pero ciertamente verdadero si comparamos con los sistemas gestores de bases de datos anteriores a este modelo).

Codd se percató de que existían bases de datos en el mercado las cuales decían ser relacionales, pero lo único que hacían era guardar la información en tablas, sin estar estas tablas literalmente normalizadas. Entonces publicó **13 reglas** que un verdadero sistema relacional debería cumplir, aunque en la práctica algunas de ellas son difíciles de realizar. Un sistema podrá considerarse "más relacional" cuanto más siga estas reglas.

**Regla 0: Regla de fundación.** Cualquier sistema que se proclame como relacional, debe ser capaz de gestionar sus bases de datos enteramente mediante sus capacidades relacionales.

**Regla 1: Regla de la información.** Toda la información en la base de datos es representada unidireccionalmente por valores en posiciones de las columnas dentro de filas de tablas. Toda la información en una base de datos relacional se representa explícitamente en el nivel Lógico exactamente de una manera: con valores en tablas.

**Regla 2: Regla del acceso garantizado.** Todos los datos deben ser accesibles sin ambigüedad. Esta regla es esencialmente una nueva exposición del requisito fundamental para las llaves primarias. Dice que cada valor escalar individual en la base de datos debe ser lógicamente direccionable especificando el nombre de la tabla, la columna que lo contiene y la llave primaria.

## BLOQUE II

## TEMA 05

**Regla 3: Regla del tratamiento sistemático de valores nulos.** El sistema de gestión de base de datos debe permitir que haya campos nulos. Debe tener una representación de la "información que falta y de la información inaplicable" que sea sistemática y distinta de todos los valores regulares.

**Regla 4: Catálogo dinámico en línea basado en el modelo relacional.** El sistema debe soportar un catálogo en línea, el catálogo relacional, que da acceso a la estructura de la base de datos y que debe ser accesible a los usuarios autorizados.

**Regla 5: Regla comprensiva del sublenguaje de los datos.** El sistema debe soportar por lo menos un lenguaje relacional que:

- Tenga una sintaxis lineal.
- Puede ser utilizado de manera interactiva.
- Tenga soporte de operaciones de definición de datos, operaciones de manipulación de datos (actualización así como la recuperación), de control de la seguridad e integridad y operaciones de administración de transacciones.

**Regla 6: Regla de actualización de vistas.** Todas las vistas que son teóricamente actualizables deben poder ser actualizadas por el sistema.

**Regla 7: Alto nivel de inserción, actualización y borrado.** El sistema debe permitir la manipulación de alto nivel en los datos, es decir, sobre conjuntos de tuplas. Esto significa que los datos no solo se pueden recuperar de una base de datos relacional a partir de filas múltiples y/o de tablas múltiples, sino que también pueden realizarse inserciones, actualización y borrados sobre varias tuplas y/o tablas al mismo tiempo y no solo sobre registros individuales.

**Regla 8: Independencia física de los datos.** Los programas de aplicación y actividades del terminal permanecen inalterados a nivel lógico aunque realicen cambios en las representaciones de almacenamiento o métodos de acceso.

**Regla 9: Independencia lógicas de los datos.** Los programas de aplicación y actividades del terminal permanecen inalterados a nivel lógico aunque se realicen cambios a las tablas base que preserven la información. La independencia de datos lógica es más difícil de lograr que la independencia física de datos.

**Regla 10: Independencia de la integridad.** Las restricciones de integridad se deben especificar por separado de los programas de aplicación y almacenarse en la base de datos. Debe ser posible cambiar esas restricciones sin afectar innecesariamente a las aplicaciones existentes.

**Regla 11: Independencia de la distribución.** La distribución de porciones de base de datos en distintas localizaciones debe ser invisible a los usuarios de la base de datos. Los usos existentes deben continuar funcionando con éxito:

- cuando una versión distribuida del SGBD se carga por primera vez
- cuando los datos existentes se redistribuyen en el sistema.

**BLOQUE II****TEMA 05**

**Regla 12: La regla de la no subversión.** Si el sistema proporciona una interfaz de bajo nivel de registro, aparte de una interfaz relacional, esa interfaz de bajo nivel no debe permitir su utilización para subvertir el sistema. Por ejemplo para sortear las reglas de seguridad relacional o las restricciones de integridad. Esto es debido a que a algunos sistemas no relacionales previamente existentes se les añadió una interfaz relacional pero, al mantener la interfaz nativa, seguía existiendo la posibilidad de trabajar no relacionamente.

**2.1 TRANSACCIONES**

Una **transacción** es una interacción con una estructura de datos compleja, compuesta por varios procesos que se han de aplicar uno después del otro. La transacción debe realizarse de una sola vez y sin que la estructura a medio manipular pueda ser alcanzada por el resto del sistema hasta que se hayan finalizado todos sus procesos.

En bases de datos se denomina **ACID** a las características de los parámetros que permiten clasificar las **transacciones** de los sistemas de gestión de bases de datos. Cuando se dice que una acción es ACID compliant se indica -en diversos grados- que ésta permite realizar transacciones.

**ACID** es un acrónimo en inglés de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad

**ATOMICIDAD** ≡ Las transacciones deben ser completas, es decir, cuando una operación consiste en una serie de pasos, o bien se ejecutan todos o ninguno.

**CONSISTENCIA** ≡ (Integridad). Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de Integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido. "La Integridad de la Base de Datos nos permite asegurar que los datos son exactos y consistentes, es decir que estén siempre intactos, sean siempre los esperados y que de ninguna manera cambian ni se deformen. De esta manera podemos garantizar que la información que se presenta al usuario será siempre la misma."

**AISLAMIENTO** ≡ Esta propiedad asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones, sobre la misma información, sean independientes y no generen ningún tipo de error. Esta propiedad define cómo y cuándo los cambios producidos por una operación se hacen visibles para las demás operaciones concurrentes.

**DURABILIDAD** ≡ (Persistencia). Esta propiedad asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer aunque falle el sistema y que de esta forma los datos sobrevivan de alguna manera.

Cumpliendo estos 4 requisitos un sistema gestor de bases de datos puede ser considerado ACID Compliant y la Bases de datos se dice que es **transaccional**.

## BLOQUE II

## TEMA 05

### 2.1 SGBD RELACIONALES MÁS CONOCIDOS

**Los más usados:** Microsoft SQL Server, MySQL, Oracle Database, PostgreSQL (combina relacional y orientado a objetos).

**Otros:** Apache Derby (JAVADB), Microsoft Access, Db2 de IBM ,IBM Informix, MariaDB ,Sybase ASE ,Firebird , SQLite.

### 3. SGBD ORIENTADOS A OBJETOS

En una base de datos orientada a objetos, la información se representa mediante objetos como los presentes en la programación orientada a objetos. Cuando se integra las características de una base de datos con las de un lenguaje de programación orientado a objetos, el resultado es un sistema gestor de base de datos orientada a objetos (ODBMS, Object Database Management System).

Este modelo trata de adaptar las bases de datos a lenguajes orientados a objetos. Para que un lenguaje orientado a objetos se pueda adaptar a bases de datos, debe cumplir las siguientes premisas:

**Abstracción:** capacidad para conceptualizar los problemas en forma de clases e instancias.

**Encapsulación:** capacidad de aunar dentro de una clase las propiedades y los métodos para tratar sus datos internos.

**Herencia:** capacidad de aunar dentro de una clase las propiedades y los métodos para trabajar con sus datos internos.

**Polimorfismo:** capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente.

Un sistema gestor de bases de datos orientadas a objetos (SGBDOO) puede verse como un SGBD que trata directamente con objetos, permitiendo concurrencia y recuperación, en vez de con registros o tablas.

Un SGBDOO debe satisfacer dos criterios:

Ser un sistema orientado a objetos: abstracción, encapsulación, modularidad, jerarquía, control de tipos, concurrencia, persistencia y genericidad.

Ser un sistema gestor de base de datos: persistencia, concurrencia, recuperación ante fallos del sistema, gestión del almacenamiento secundario y facilidad de consultas.

#### Características

**Persistencia:** Capacidad que tiene el programador para que sus datos se conserven al finalizar la ejecución de un proceso para poder reutilizarlos.

**Concurrencia:** Debe controlar la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos.

## BLOQUE II

## TEMA 05

**Recuperación:** En caso de fallo de hardware o software, el sistema puede retroceder hasta un estado coherente de los datos.

**Gestión del almacenamiento secundario:** Es soportada por un conjunto de mecanismos que son invisibles al usuario como la gestión de índices, agrupación de datos, selección del camino de acceso, optimización de consultas, etc.

**Facilidad de consultas:** Permitir al usuario hacer cuestiones sencillas a la base de datos. Este tipo de consultas tienen como misión proporcionar la información solicitada por el usuario de forma rápida y correcta.

**El lenguaje (OQL):** Es el lenguaje de consulta propuesto por ODMG-93. Se caracteriza por tener una sintaxis abstracta, su semántica formal puede definirse fácilmente, proporciona un acceso declarativo a los objetos, puede optimizarse fácilmente y tiene una sintaxis concreta al estilo SQL, pero puede cambiarse con facilidad.

### 3.1 MAPEO OBJETO RELACIONAL

El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, **ORM**, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia.

En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Existen herramientas de software o bien librerías para determinados lenguajes, que permiten realizar este mapeo, se les conoce como software ORM o librerías ORM.

#### Software ORM más populares

**Para Java:** Hibernate

**Para PHP:** Eloquent (usando framework laravel), Propel para PHP (usando framework symfony), Doctrine

**Para .NET:** NHibernate, ADO.NET Entity Framework (C#)

**Para Python:** peewee, Object, SQLAlchemy

**Para Node.js:** Sequelize, Prisma

### 3.2 DESFASE OBJETO-RELACIONAL

El desfase objeto-relacional, también conocido como **impedancia objeto-relacional**, consiste en la diferencia de aspectos que existen entre la programación orientada a objetos y la base de datos. Estos aspectos se puede presentar en cuestiones como:

## BLOQUE II

## TEMA 05

- Lenguaje de programación: el programador debe conocer el lenguaje de programación orientado a objetos (POO) y el lenguaje de acceso a datos.
- Tipos de datos: en las bases de datos relacionales siempre hay restricciones en el uso de tipos, mientras que la programación orientada a objetos utiliza tipos de datos más complejos.
- Paradigma de programación: en el proceso de diseño y construcción del software se tiene que hacer una traducción del modelo orientado a objetos de clases al modelo Entidad-Relación (E/R) puesto que el primero maneja objetos y el segundo maneja tablas y tuplas (o filas), lo que implica que se tengan que diseñar dos diagramas diferentes para el diseño de la aplicación.

El modelo relacional trata con relaciones y conjuntos debido a su naturaleza matemática. Sin embargo, el modelo de POO trata con objetos y las asociaciones entre ellos. Por esta razón, el problema entre estos dos modelos surge en el momento de querer persistir los objetos de negocio.

La escritura (y de manera similar la lectura) mediante JDBC implica:

- Abrir una conexión.
- Crear una sentencia en SQL.
- Copiar todos los valores de las propiedades de un objeto en la sentencia, ejecutarla y así almacenar el objeto.

Esto es sencillo para un caso simple, pero complicado si el objeto posee muchas propiedades, o bien se necesita almacenar un objeto que a su vez posee una colección de otros elementos. Se necesita crear mucho más código, además del tedioso trabajo de creación de sentencias SQL.

Este problema es lo que se denomina impedancia Objeto-Relacional, o sea, el conjunto de dificultades técnicas que surgen cuando una base de datos relacional se usa en conjunto con un programa escrito en POO.

### 3.3. BASES DE DATOS ORIENTADAS A OBJETOS CONOCIDAS

**ObjectDB.** es una base de datos orientada a objetos para Java. Se puede utilizar en modo cliente-servidor y en modo incrustado (en proceso). ObjectDB es un software multiplataforma y se puede utilizar en varios sistemas operativos con Java SE 5 o superior.

**Zope Object Database.** ZODB es una base de datos orientada a objetos para almacenar de forma transparente y persistente objetos en el lenguaje de programación **Python**.

**db4o:** Las claves innovadoras de este producto es su alto rendimiento y el modelo de desarrollo que proporciona a las aplicaciones para su capa de acceso a datos, el cual propugna un abandono completo del paradigma relacional de las bases de datos tradicionales. Se evita el problema de la impedancia objeto-relacional.

**Otros:** ZooDB, GemStone S, Objectivity/DB



## BLOQUE II

## TEMA 05

### 4. SGBD NO-SQL

Se puede decir que la aparición del término NoSQL aparece con la llegada de la web 2.0 ya que hasta ese momento sólo subían contenido a la red aquellas empresas que tenían un portal, pero con la llegada de aplicaciones como Facebook, Twitter o Youtube, cualquier usuario podía subir contenido, provocando así un crecimiento exponencial de los datos.

Es en este momento cuando empiezan a aparecer los primeros problemas de la gestión de toda esa información almacenada en bases de datos relacionales. En un principio, para solucionar estos problemas de accesibilidad, las empresas optaron por utilizar un mayor número de máquinas pero pronto se dieron cuenta de que esto no solucionaba el problema, además de ser una solución muy cara. La otra solución era la creación de sistemas pensados para un uso específico que con el paso del tiempo han dado lugar a soluciones robustas, apareciendo así el movimiento NoSQL.

Por lo tanto hablar de bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales generan ciertos problemas debido principalmente a problemas de escalabilidad y rendimiento de las bases de datos relacionales donde se dan cita miles de usuarios concurrentes y con millones de consultas diarias.

Además de lo comentado anteriormente, las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad-relación. Tampoco utilizan una estructura de datos en forma de tabla donde se van almacenando los datos sino que para el almacenamiento hacen uso de otros formatos como clave-valor, mapeo de columnas o grafos (ver epígrafe 'Tipos de bases de datos NoSQL').

### VENTAJAS

Esta forma de almacenar la información ofrece ciertas ventajas sobre los modelos relacionales. Entre las ventajas más significativas podemos destacar:

- Se ejecutan en máquinas con pocos recursos: Estos sistemas, a diferencia de los sistemas basados en SQL, no requieren de apenas computación, por lo que se pueden montar en máquinas de un coste más reducido.
- Escalabilidad horizontal: Para mejorar el rendimiento de estos sistemas simplemente se consigue añadiendo más nodos, con la única operación de indicar al sistema cuáles son los nodos que están disponibles.
- Pueden manejar gran cantidad de datos: Esto es debido a que utiliza una estructura distribuida, en muchos casos mediante tablas Hash.
- No genera cuellos de botella: El principal problema de los sistemas SQL es que necesitan transcribir cada sentencia para poder ser ejecutada, y cada sentencia compleja requiere además de un nivel de ejecución aún más complejo, lo que constituye un punto de entrada en común, que ante muchas peticiones puede ralentizar el sistema.



## BLOQUE II

## TEMA 05

### Principales diferencias con las bases de datos SQL

Algunas de las diferencias más destacables que nos podemos encontrar entre los sistemas NoSQL y los sistemas SQL están:

- No utilizan SQL como lenguaje de consultas. La mayoría de las bases de datos NoSQL evitan utilizar este tipo de lenguaje o lo utilizan como un lenguaje de apoyo. Por poner algunos ejemplos, Cassandra utiliza el lenguaje CQL, MongoDB utiliza JSON o BigTable hace uso de GQL.
- No utilizan estructuras fijas como tablas para el almacenamiento de los datos. Permiten hacer uso de otros tipos de modelos de almacenamiento de información como sistemas de clave-valor, objetos o grafos.
- No suelen permitir operaciones JOIN. Al disponer de un volumen de datos tan extremadamente grande suele resultar deseable evitar los JOIN. Esto se debe a que, cuando la operación no es la búsqueda de una clave, la sobrecarga puede llegar a ser muy costosa. Las soluciones más directas consisten en desnormalizar los datos, o bien realizar el JOIN mediante software, en la capa de aplicación.
- Arquitectura distribuida. Las bases de datos relacionales suelen estar centralizadas en una única máquina o bien en una estructura máster-esclavo, sin embargo en los casos NoSQL la información puede estar compartida en varias máquinas mediante mecanismos de tablas Hash distribuidas.

### ESQUEMAS DINÁMICOS

En las bases de datos relacionales es necesario definir los esquemas antes de añadir los datos. Por ejemplo, si desea almacenar datos de clientes, como el número de teléfono, nombre y apellidos, dirección, ciudad y provincia, la base de datos SQL debe conocer de antemano el tipo de datos que va a almacenar.

Esto dificulta la agilidad en el desarrollo, porque cada vez que se añaden características, con frecuencia se debe modificar el esquema de la base de datos. Así, si tras varias iteraciones de desarrollo, decide que desea almacenar los artículos favoritos de los clientes además de su dirección y teléfono, deberá añadir esa columna a la base de datos y luego migrar la base de datos entera al nuevo esquema.

Si la base de datos es grande, el proceso será muy lento y la actividad quedará interrumpida de forma prolongada. Si cambia a menudo los datos que almacena la aplicación —porque realiza iteraciones rápidamente— también se interrumpirá la actividad de forma frecuente. En las bases relacionales tampoco se pueden incluir datos sin ningún tipo de estructura o desconocidos de antemano.

Las bases de datos NoSQL están diseñadas para que se puedan insertar datos sin un esquema predefinido. Por ello resulta fácil realizar modificaciones importantes en las aplicaciones en tiempo real sin interrupciones del servicio, de modo que el desarrollo es

## BLOQUE II

## TEMA 05

más rápido, la integración del código es más fiable y los administradores de las bases de datos tienen menos trabajo. Tradicionalmente, los desarrolladores han tenido que añadir código a la aplicación para realizar controles de calidad de los datos, como, por ejemplo, forzar la presencia de campos, tipos de datos o valores permisibles específicos. Las bases de datos NoSQL, al ser más sofisticadas, permiten la aplicación de las reglas de validación en la base de datos, de modo que los usuarios pueden aplicar la gobernanza a los datos, aprovechando la agilidad que ofrece un esquema dinámico.

### SHARDING AUTOMATICO

Por el modo en el que están estructuradas, **las bases de datos relacionales se suelen escalar de forma vertical**: un solo servidor debe alojar toda la base de datos para garantizar un rendimiento aceptable de las operaciones JOIN y transacciones entre tablas. Esto encarece su coste con rapidez, limita la escalabilidad y crea un número de puntos de fallo relativamente pequeño para la infraestructura de bases de datos. La solución para que las aplicaciones puedan crecer de forma rápida **es permitir la escalabilidad horizontal añadiendo servidores**, en lugar de concentrar más capacidad en un único servidor.

Es posible aplicar el sharding a una base de datos SQL en un gran número de instancias de servidor, pero para ello normalmente se necesitan sistemas SAN y otras configuraciones complejas para que el hardware actúe como un único servidor. Como las bases de datos relacionales no proporcionan esta función de forma nativa, los equipos de desarrollo deben encargarse de implementar varias bases de datos de este tipo en diferentes máquinas. Los datos se almacenan en cada instancia de base de datos de forma autónoma. El código de la aplicación se desarrolla para distribuir los datos, distribuir las consultas y agregar los resultados de los datos en todas las instancias de la base de datos. Se debe desarrollar código adicional para manejar los fallos de los recursos, y realizar operaciones JOIN entre diferentes bases de datos, así como para aplicar otros requisitos, como el reequilibrio de datos y la replicación. Además, muchas de las ventajas de las bases de datos relacionales, como la integridad transaccional, se ven afectadas o eliminadas al utilizar el sharding manual.

Por el contrario, las bases de datos NoSQL suelen admitir el auto-sharding, es decir, pueden distribuir los datos de forma nativa y automática entre un número arbitrario de servidores, sin que la aplicación deba tener constancia de la composición del grupo de servidores. Los datos y la carga de consultas se equilibran automáticamente entre los servidores, y cuando uno falla, se puede sustituir de forma rápida y transparente sin que la aplicación deje de funcionar.

## BLOQUE II

## TEMA 05

### REPLICACION

La mayoría de las bases de datos NoSQL también admiten replicación automática de bases de datos para garantizar la disponibilidad en caso de que se produzcan interrupciones de servicio o paradas de mantenimiento planificadas. Las bases de datos NoSQL más sofisticadas son autorreparables y ofrecen las funciones de conmutación por error y recuperación, así como la posibilidad de distribuir la base de datos entre varias regiones geográficas para soportar fallos regionales y permitir la localización de los datos. A diferencia de las bases de datos relacionales, las bases de datos NoSQL no necesitan utilizar otras aplicaciones o complementos con un precio elevado para implementar la replicación.

### CACHÉ INTEGRADA

Para los sistemas de bases de datos SQL existen varios productos que proporcionan un nivel de caché. Estos sistemas pueden mejorar el rendimiento de las operaciones de lectura de forma importante, pero no mejoran el rendimiento de las operaciones de escritura y añaden complejidad operativa a las implementaciones. Si en su aplicación la mayoría de las operaciones realizadas son de lectura, se puede considerar una caché distribuida, pero si solo tiene un volumen moderado de operaciones de escritura, puede que una caché distribuida no mejore la experiencia para los usuarios finales en términos generales, y añadirá complejidad a la hora de gestionar la invalidación de caché.

Muchas tecnologías de base de datos NoSQL incorporan excelentes funcionalidades de caché, que mantienen durante el máximo tiempo posible los datos usados con frecuencia en la memoria del sistema y hacen que sea innecesario disponer de una capa de caché independiente. Algunas bases de datos NoSQL también ofrecen una capa de gestión de la base de datos integrada en la memoria, pensada para cargas de trabajo que requieren el máximo rendimiento y una latencia mínima.

**BLOQUE II****TEMA 05****Resumen sobre la comparativa entre NoSQL y SQL**

	Bases de datos SQL	Bases de datos NoSQL
Tipos	Un tipo (base de datos SQL) con pequeñas variaciones.	Muchos tipos, entre los que se incluyen almacenes de clave-valor, Bases de datos de documentos: bases de datos orientadas a columnas y bases de datos de grafos.
Historia sobre el desarrollo	Se desarrollan a finales de la década de 1970 para respaldar la primera ola de aplicaciones de almacenamiento de datos.	Se desarrollan a finales de la década de 2000 para superar las limitaciones de las bases de datos SQL, sobre todo en lo relativo a escalabilidad, datos multiestructurados, distribución geográfica y sprints de desarrollo ágiles.
Ejemplos	MySQL, Postgres, Microsoft SQL Server, Oracle Database.	MongoDB, Cassandra, HBase, Neo4j.
Modelo de almacenamiento de datos	Los registros individuales (p. ej., «empleados») se almacenan en filas de tablas, y cada columna almacena un dato específico sobre ese registro (p. ej., «director», «fecha de contratación», etc.) al estilo de las hojas de cálculo. Los datos relacionados se almacenan en tablas separadas, y luego se combinan cuando se ejecutan consultas más complejas. Por ejemplo, «oficinas» se puede almacenar en una tabla y «empleados», en otra. Cuando un usuario quiere encontrar la dirección de trabajo de un empleado, el motor de la base de datos une las tablas «empleado» y «oficina» para obtener toda la información necesaria.	Varía en función del tipo de base de datos. Por ejemplo, los almacenes de clave-valor funcionan de forma parecida a las bases de datos SQL, pero solo tienen dos columnas («clave» y «valor»), con información más compleja, que a veces se almacena como objetos BLOB en las columnas «valor». Las bases de datos de documentos evitan por completo el modelo de tablas y filas, almacenando todos los datos relevantes juntos en un único documento en formato JSON, XML u otro formato, que puede anidar valores de forma jerárquica.

**BLOQUE II**

**TEMA 05**

Esquemas	La estructura y los tipos de datos se establecen de entrada. Para almacenar información sobre un nuevo dato, se debe modificar toda la base de datos, que deja de funcionar durante este tiempo.	Suelen ser dinámicos, y en ocasiones es necesario aplicar algunas reglas de validación de datos. Las aplicaciones pueden ir añadiendo nuevos campos según convenga y, a diferencia de lo que ocurre con las filas de las tablas SQL, en caso necesario se pueden almacenar juntos datos de naturaleza diferente. Para algunas bases de datos (como las orientadas a columnas), resulta un poco más complicado añadir nuevos campos de forma dinámica.
Escalabilidad	Escalado vertical, es decir, es necesario añadir más potencia a un único servidor en caso de que aumente la demanda. Es posible repartir las bases de datos SQL por diferentes servidores, pero es una tarea que requiere una participación importante del equipo de ingeniería, y suelen perderse funcionalidades relacionales básicas como las operaciones JOIN, la integridad de las referencias y las transacciones.	Escalado horizontal, es decir, para aumentar la capacidad, el administrador de la base de datos solo tiene que añadir servidores básicos o instancias de la nube. La base de los datos distribuye los datos en los servidores según las necesidades.
Modelo de desarrollo	Mix of open technologies (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open technologies
Soporta transacciones ACID de registros múltiples	Sí	Generalmente no. MongoDB 4.0 y posteriores admiten transacciones ACID de múltiples documentos. Más Información
Manipulación de datos	Lenguaje específico que utiliza instrucciones Select, Insert y Update, p. ej., SELECT fields FROM table WHERE...	A través de APIs orientadas a objetos

**BLOQUE II****TEMA 05**

Coherencia Se puede configurar un nivel de consistencia elevado

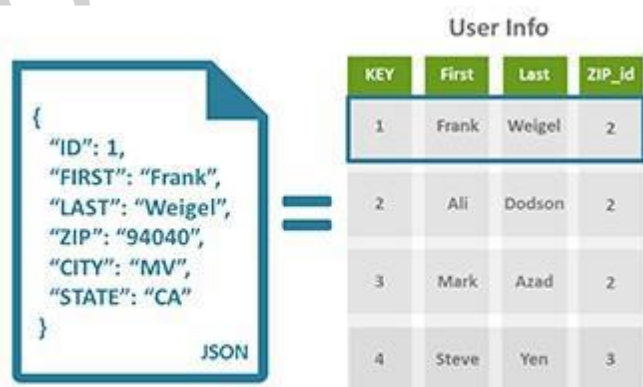
Depende del producto. Algunos ofrecen una consistencia elevada (p. ej., MongoDB, con consistencia ajustable para las lecturas), mientras que otros ofrecen consistencia eventual (p. ej., Cassandra).

**4.1 TIPOS DE BASES DE DATOS NoSQL**

**Bases de datos DOCUMENTALES** ≡ en estas bases de datos se empareja cada clave con una estructura de datos compleja que se denomina 'documento'. Los documentos pueden contener muchos pares de clave-valor distintos, o pares de clave-matriz, o incluso documentos anidados.

Las bases de datos documentales, en inglés, document stores, se utilizan para la administración de datos semiestructurados. Se trata de datos que no siguen una estructura fija, sino que llevan la estructura casi en sí misma. Sin embargo, con ayuda de marcadores dentro de estos datos, la información puede ordenarse. Debido a la falta de una estructura clara, estos datos no son adecuados para bases de datos relacionales porque su información no puede clasificarse en tablas.

Una base de datos documental crea un par simple: un documento específico se asigna a una clave. En este documento, que puede tener formato XML, JSON o YAML, por ejemplo, se puede encontrar la información propiamente dicha. Como la base de datos no requiere un esquema específico, en un almacén de documentos pueden integrarse diferentes clases de documentos. Los cambios en los documentos no afectan a la base de datos.

**¿Cómo funcionan las bases de datos documentales?**

En teoría, una base de datos de documentos puede almacenar datos en diferentes formatos sin un esquema coherente. En la práctica, sin embargo, para facilitar el trabajo con la base de datos, se suele utilizar un formato de archivo para los documentos y una estructura fija para los datos. Esto permite, por ejemplo, procesar mejor las consultas a la base de datos. En general, se pueden realizar las mismas acciones con una base de datos documental que con un sistema relacional: la información se puede insertar, modificar, borrar y consultar.

## BLOQUE II

## TEMA 05

Para poder realizar estas acciones, cada documento recibe un identificador único que puede componerse tanto de una simple cadena de caracteres como de la ruta completa que lleva hasta él. Cuando se buscan datos, se comprueban los propios documentos, es decir, no se recorren todas las columnas en la base de datos en busca de esos datos, sino que se extraen de los mismos documentos.

### ¿Cuáles son las ventajas y desventajas de las bases de datos documentales?

En las bases de datos relacionales clásicas, debe haber un campo para cada dato, y esto en cada entrada. Si la información no está disponible, la celda queda vacía, pero debe haber una. Las bases de datos documentales son mucho más flexibles: la estructura de los documentos no tiene por qué ser coherente, lo que permite almacenar conjuntos de datos no estructurados muy grandes en una sola base de datos.

También es más fácil integrar datos nuevos: mientras que, en una base de datos relacional, el nuevo punto de información se debe insertar en todos los registros de datos, en una base de datos orientada a documentos basta con integrar la información nueva en solo unos pocos registros. El contenido adicional se puede añadir a otros documentos, pero esto no es necesario.

En estas bases de datos, además, la información no se reparte en varias tablas enlazadas. Todo se almacena en el mismo lugar, lo que puede mejorar el rendimiento. Sin embargo, esta ventaja de velocidad solo puede explotarse en las bases de datos documentales, siempre y cuando no se intente insertar elementos relacionales, porque las referencias no encajan en el concepto documental. Si, pese a todo, se intenta conectar a los documentos entre sí, el sistema se vuelve muy complejo y voluminoso. Cuando se trata de datos muy interconectados, se recomienda usar un sistema de base de datos relacional.

### Las bases de datos documentales más conocidas

Las bases de datos documentales son de gran importancia, en particular para el desarrollo de aplicaciones web. Debido a la demanda resultante del desarrollo web, hay ahora numerosos sistemas de gestión de bases de datos (SGBD) en el mercado. La siguiente selección enumera los más conocidos:

**BaseX:** este proyecto open source utiliza Java y XML. BaseX viene con una interfaz de usuario.

**CouchDB:** la Apache Software Foundation lanzó el software de código abierto CouchDB. El sistema de gestión de bases de datos está escrito en Erlang, utiliza JavaScript y se utiliza en aplicaciones de Ubuntu y Facebook, entre otras.

**Elasticsearch:** este motor de búsqueda funciona con una base de datos documental. Para esto, emplea documentos JSON.

**eXist:** el SGBD eXist, de código abierto, se ejecuta en una máquina virtual Java y, por lo tanto, puede utilizarse con independencia del sistema operativo. Utiliza principalmente documentos XML.



**BLOQUE II****TEMA 05**

**MongoDB:** MongoDB es la base de datos NoSQL más extendida del mundo. El software está escrito en C++ y emplea documentos similares a JSON. De esquema libre, es decir, que cada entrada puede tener un esquema de datos diferente que nada tenga que ver con el resto de registros almacenados. Es bastante rápido a la hora de ejecutar sus operaciones ya que está escrito en lenguaje C++.

Para el almacenamiento de la información, utiliza un sistema propio de documento conocido con el nombre BSON, que es una evolución del conocido JSON pero con la peculiaridad de que puede almacenar datos binarios.

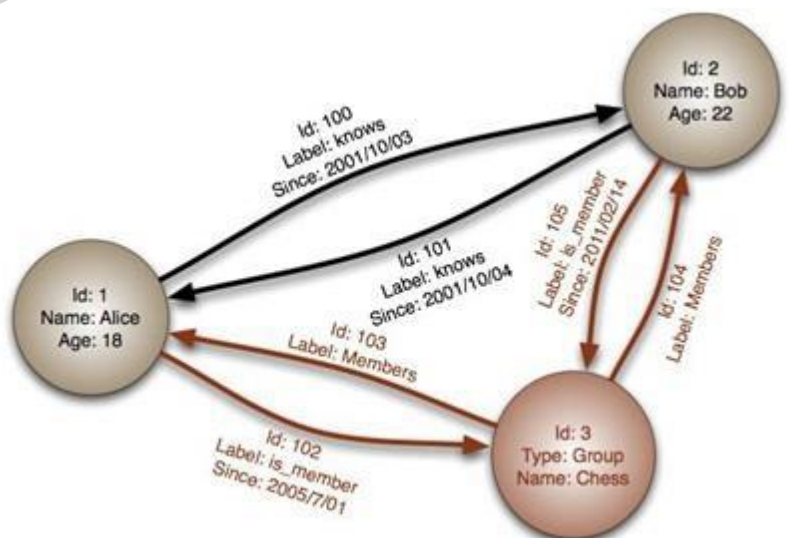
**SimpleDB:** con SimpleDB (escrito en Erlang), Amazon ha desarrollado su propio SGBD para los servicios en la nube de la compañía. El proveedor cobra una tarifa por su uso.

**Otros:** RavenDB, BaseX, djondb, SimpleDB de Amazon, IBM Lotus Domino, Terrastore

**ALMACENES DE GRAFOS** ≡ Se utilizan para almacenar información sobre redes de datos, como las conexiones sociales.

Una base de datos orientada a grafos o graph database se basa, como bien dice su nombre, en grafos, un conjunto de objetos (vértices y aristas) que permite representar datos interconectados, así como las relaciones entre ellas, de forma comprensible y como un único y más amplio conjunto de datos. Los grafos están formados por nodos o vértices, que son propiedades de datos u objetos claramente señalizadas e identificables, y aristas o arcos, que representan las relaciones entre los objetos. Gráficamente, estos dos componentes tienen forma de puntos y líneas, respectivamente. Las aristas tienen un extremo inicial y uno final, mientras que cada nodo siempre contiene un número concreto de relaciones a otros nodos, ya sean de entrada o de salida.

Dos conceptos habituales en cuanto a la estructura de una graph database son el Labeled-Property Graph (grafo de propiedades etiquetadas) y Resource Description Framework (marco de descripción de recursos, RDF). En el primero, se asignan propiedades (properties) concretas tanto a los nodos como a las aristas; en el segundo, en cambio, la estructura del grafo se regula mediante **tripels y quads**: los tripels se componen de tres elementos, siguiendo el esquema nodo-borde-nodo. Los quads complementan a los tripels con información de contexto adicional, facilitando así su separación en grupos.





**BLOQUE II****TEMA 05****Las bases de datos orientadas a grafos más conocidas**

**Neo4j:** Neo4j es la graph database más popular y está concebida como modelo de código abierto. Usa un lenguaje llamado cypher.

**Amazon Neptune:** esta base de datos de gráficos puede usarse a través de la nube pública de Amazon Web Services y se abrió al público en 2018 como base de datos de alto rendimiento.

**SAP Hana Graph:** SAP ha creado con SAP Hana una plataforma basada en un sistema de gestión de bases de datos relacional que se completa con el modelo integrado SAP Hana Graph, orientado a grafos.

**OrientDB:** esta graph database está considerada como uno de los modelos más rápidos disponibles actualmente.

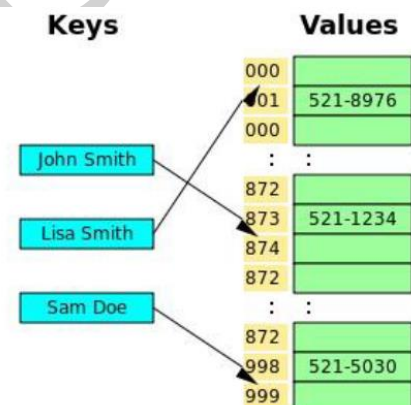
**Otras:** DEX/Sparksee, AllegroGraph, InfiniteGraph, Sones GraphDB, InfoGrid, HyperGraphDB

**ALMACENES DE PARES CLAVE-VALOR**  $\equiv$  Son las bases de datos NoSQL más simples. Cada elemento de la base de datos se almacena como un nombre de atributo (o «clave»), junto con su valor. Ejemplos de almacenes de clave-valor son Riak y Berkeley DB. En algunos almacenes de clave-valor, como Redis, cada valor puede tener un tipo, como «entero», lo que le añade funcionalidad.

Esta modalidad de base de datos, key-value database o store en inglés, se basa en una tabla de tan solo dos columnas. En una de ellas se guarda un valor y en la otra, una clave que representa una característica identificativa única. Un valor puede ser sencillo, como una cadena de caracteres o un número entero, o pueden ser objetos complejos (un documento también puede ocupar el lugar de un valor, aunque, entonces se hablaría de una base de datos de documentos). En las bases de datos se pueden incluir también referencias a archivos, así como a tuplas (conjunto de valores).

El contenido de la base de datos puede ser muy heterogéneo, por lo que es posible incluir objetos distintos en una misma columna. Lo mismo se aplica a los rasgos identificativos. En la mayoría de los casos, las claves seguirán un determinado esquema, pero esto no es indispensable. Tanto las cadenas de caracteres como los enteros se pueden constituir siguiendo criterios libres.

Las bases de datos clave-valor son muy efectivas en la consulta y fáciles de escalar, lo cual se deriva de la sencilla estructura del modelo. Asimismo, este modelo de base de datos ofrece una gran velocidad de búsqueda gracias a la sencilla conexión entre la clave y el valor.



## BLOQUE II

## TEMA 05

### Bases de datos clave-valor más conocidas.

**Amazon DynamoDB:** pertenece a Amazon Web Services (AWS) y puede emplearse también como base de datos de documentos.

**Berkeley DB:** desarrollado por Oracle, ofrece interfaces para diferentes lenguajes de programación.

**Redis:** proyecto de código abierto. Constituye uno de los SGBD más utilizados. Es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes (clave/valor) pero que opcionalmente puede ser usada como una base de datos durable o persistente.

Tiene la ventaja de que sus operaciones son atómicas y persistentes. Por ponerle una pega, Redis no permite realizar consultas, sólo se puede insertar y obtener datos, además de las operaciones comunes sobre conjuntos (diferencia, unión e inserción).

Creado en ANSI C, por lo tanto es compatible y funciona sin problemas en sistemas Unix, Linux y sus derivados, Solaris, OS/X sin embargo no existe soporte oficial para plataformas Windows.

**Riak:** Riak existe en una variante de código libre, como solución empresarial y en forma de almacenamiento en la nube.

**Voldemort:** extendido SGBD, utilizado e impulsado por LinkedIn, entre otros.

**Cassandra:** de Apache The Apache Cassandra. Dispone de un lenguaje propio para realizar consultas CQL (Cassandra Query Language). Cassandra es una aplicación Java por lo que puede correr en cualquier plataforma que cuente con la JVM. Se puede usar también como base de datos columnar.

**Otros:** BigTable, de Google, Oracle NoSQL

### BASES DE DATOS ORIENTADAS A COLUMNAS, COLUMNARES o TABULAR ≡

Estas bases de datos permiten realizar consultas en grandes conjuntos de datos y almacenan los datos en columnas, en lugar de filas.

En general, las bases de datos se orientan a filas. El sistema de gestión de base de datos (SGBD) crea una línea para cada entrada. Los campos con la información se listan uno tras otro. Un ejemplo clásico lo encontramos en las bases de datos relacionales. Las bases de datos columnares hacen lo contrario: por cada entrada, hay una columna, por lo tanto, los datos de cada entrada están dispuestos uno debajo del otro (y no uno al lado del otro, como en la variante orientada a filas).

Para clarificar, mostramos aquí el sistema orientado a filas:

**BLOQUE II****TEMA 05**

Número	Apellido	Nombre	Clave
1.	Skywalker	Luke	3FN-Z768
2	Kenobi	Obi-wan	7TR-K345
3	Organa	Leia	8NN-R266

La base de datos columnar gira la base de datos, quedando de esta forma:

Número	1.	2.	3.
Apellido	Skywalker	Kenobi	Organa
Nombre	Luke	Obi-wan	Leia
Clave	3FN-Z768	7TR-K345	8NN-R266

Los sistemas relacionales basados en líneas se utilizan sobre todo cuando hay que realizar muchas transacciones rápidamente. Escribir, cambiar, borrar entradas: todo esto funciona muy bien con bases de datos relacionales. Las bases de datos columnares se utilizan cuando hay que analizar grandes cantidades de datos.

**Las bases de datos columnares más conocidas**

Aunque los sistemas de gestión de bases de datos columnares se utilizan desde hace ya bastante tiempo, el número de implementaciones disponibles es aún muy escaso, porque hay mayor demanda de bases de datos relacionales. A pesar de esto, se han establecido algunos sistemas.

**Amazon Redshift:** como parte de Amazon Web Services (AWS), Redshift ofrece un almacén de datos columnar para big data.

**MariaDB-ColumnStore:** el DBMS MariaDB (Fork de MySQL) de código abierto ofrece con el ColumnStore también una combinación de base de datos columnar y relacional.

**SAP HANA:** la plataforma de desarrollo de SAP también utiliza una combinación de una base de datos relacional y una columnar.

**Apache Cassandra:** el software libre está basado en Apache Hadoop y está escrito en Java.

**MonetDB:** este software de código abierto se desarrolló prestando especial atención a la minería de datos.

**Otros:** HBase de Apache, BigTable de Google, LevelDB versión abierta de BigTable, Hypertable