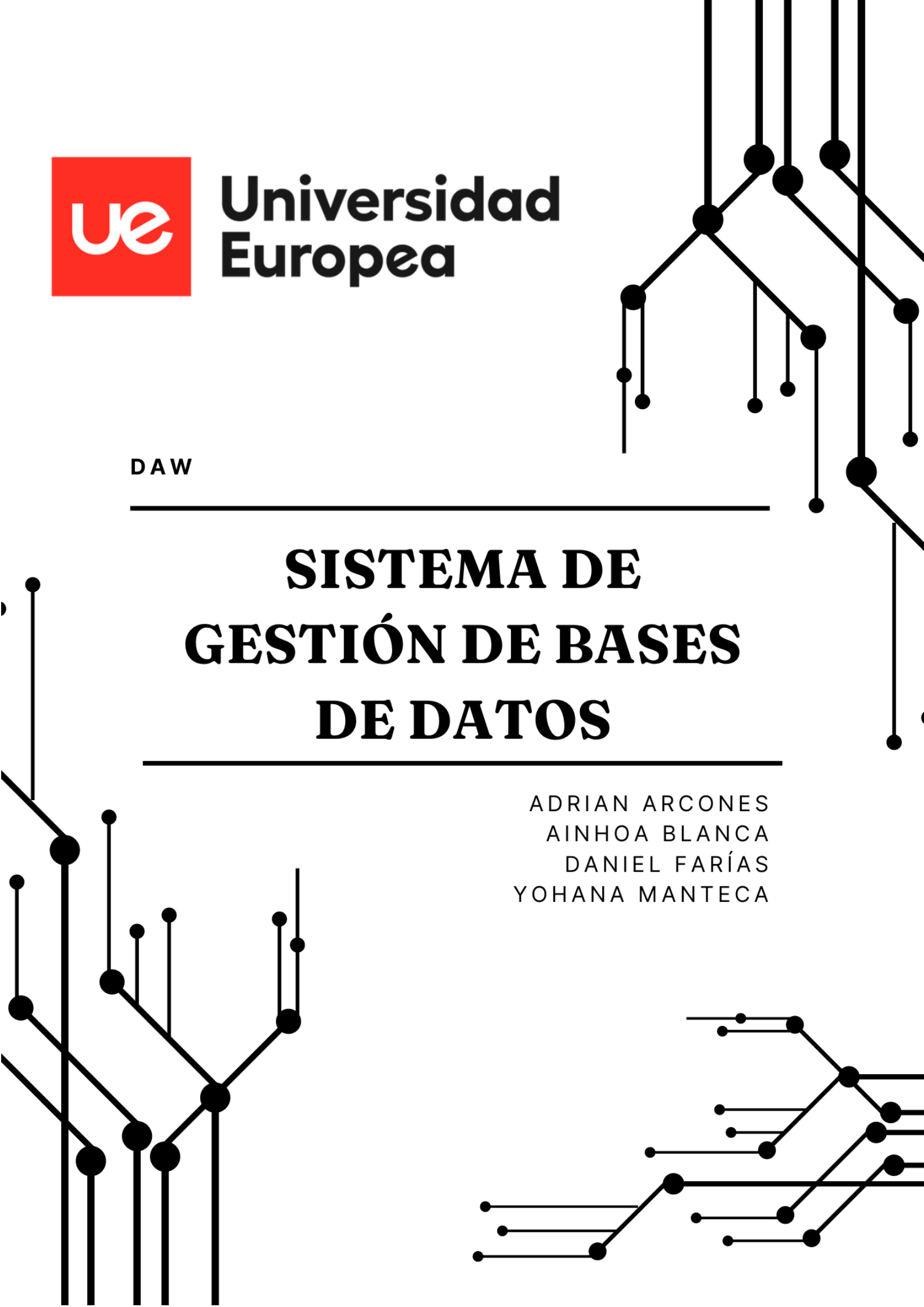

Universidad Europea

DAW

SISTEMA DE GESTIÓN DE BASES DE DATOS

ADRIAN ARCONES
AINHOA BLANCA
DANIEL FARÍAS
YOHANA MANTECA

Índice.

Introducción.

El presente trabajo tiene como objetivo analizar la gestión de datos en profundidad, abordando aspectos clave como su historia, plataformas compatibles, lenguajes de programación soportados e integración con sistemas de control de versiones. También se explorarán las herramientas y extensiones disponibles, así como los frameworks con los que se puede integrar. Finalmente, se evaluarán las ventajas y desventajas del gestor seleccionado frente a sus competidores, proporcionando una visión completa sobre su funcionalidad y relevancia en el mercado actual.

Sistema de Gestión de Bases de Datos.

El gestor de la base de datos es un software que permite a los usuarios definir, crear, mantener y controlar el acceso a las bases de datos; también es encargado de proporcionar una interfaz a los usuarios entre los datos almacenados y los programas de aplicación que los manejan. Puede verse el gestor de la base de datos como un intérprete entre el usuario y los datos. Toda operación que se quiere realizar "contra" la base de datos debe ser previamente autorizada por el gestor de esta.

Historia y evolución.

Los sistemas de Gestión de Bases de Datos han ido evolucionando a través del tiempo, desarrollando habilidades para adaptarse a cada época y su tecnología. Su historia ha sido la siguiente:

Año 1960: Sistema de navegación.

En los años 60 con la evolución de las computadoras comenzaron a aparecer las primeras bases de datos y estaban basadas en dos modelos, jerárquicos y en red:

- **Modelo jerárquico:** Organiza los datos en una estructura de árbol, donde los registros tienen una relación padre-hijo. Este modelo fue utilizado en sistemas como IBM Information Management System (IMS). No se permitían relaciones múltiples por lo rígido de este modelo.
- **Modelo de red:** Este modelo permitía una estructura más flexible que el modelo jerárquico, con relaciones más complejas entre los datos. Uno de los primeros ejemplos de este enfoque es el sistema IDS (Integrated Data Store), creado por Charles Bachman. Este modelo, sin embargo, requería una navegación explícita entre los datos, lo que lo hacía más complicado de manejar.

Año 1970: Sistema relacional.

En los años 70 llega la gran revolución de las bases de datos. Edgar F. Codd cuando trabajaba en IBM desarrolló el modelo relacional. El cual organiza tablas, lo que facilita el acceso a datos sin necesidad de navegar por estructuras jerárquicas o de red.

- IBM fue el primero en implementar dicho modelo en el proyecto System R e introdujo el uso SQL (Structured Query Language).

- En 1979, Oracle sacó su versión comercial inspirada en el modelo de Codd. Fue uno de los primeros en ofrecer SQL como lenguaje estándar.

Año 1980: Expansión del sistema relacional.

En la década de los 80 sucede que se sigue usando el modelo relacional pero cada empresa lo va evolucionando.

- En 1986, IBM lanza DB2 que es una implementación más completa del modelo relacional.
- En 1989, Microsoft saca su primera versión de SQL Server para competir contra Oracle e IBM.
- En dicha década también se crean bases de datos como PostgreSQL.

Año 1990: Bases de datos orientada a objetos.

En los años 90, con el aumento de la complejidad de las aplicaciones, surgieron las bases de datos orientadas a objetos, como ObjectStore y GemStone, aunque no fueron tan exitosas como las relacionales. Durante este tiempo, MySQL (lanzado en 1995) ganó fama y estatus por ser utilizado junto con PHP en servidores LAMP (Linux, Apache, MySQL, PHP).

Año 2000: Aparición de NoSQL.

Para el nuevo siglo, debido al crecimiento masivo de datos generado por empresas como Google y Amazon, aparecieron las bases de datos NoSQL.

NoSQL agrupa a una serie de tecnologías de bases de datos no relacionales, diseñadas para manejar datos no estructurados o semiestructurados, como documentos JSON, gráficos o datos en tiempo real. Estas tecnologías, como MongoDB, Cassandra, Redis y Neo4j, fueron diseñadas para manejar datos no estructurados y escalar horizontalmente agregando más servidores.

Año 2010 - Actualidad: Bases de datos híbridas y en la nube.

En la década de 2010, se desarrollaron bases de datos híbridas que combinan lo mejor de las relacionales y NoSQL, como Google Cloud Spanner. Además, las bases de datos en la nube, como Amazon RDS y Google BigQuery, se hicieron populares, permitiendo a las empresas gestionar bases de datos sin infraestructura física. También

surgieron sistemas multimodales que integran datos relacionales y no relacionales en una sola plataforma.

Plataformas en las que se puede utilizar.

Con los avances tecnológicos los sistemas de gestión de bases de datos han tenido que evolucionar para poder estar en distintas plataformas. Las más comunes son:

- **Windows**: MySQL, SQL Server, Oracle.
- **Linux**: PostgreSQL, MySQL, MongoDB.
- **MacOS**: PostgreSQL, MySQL.
- **Cloud**: AWS, Google Cloud, Microsoft Azure, IBM Cloud.

Muchos sistemas de gestión de bases de datos son multiplataformas y pueden ejecutarse tanto en servidores locales como en la nube.

Lenguajes de programación que soportan.

La mayoría de los Sistemas de gestión de bases de datos son compatibles con una amplia variedad de lenguajes de programación mediante conectores o bibliotecas. Se pueden dividir en 2 grupos y son:

1. **Lenguajes para Bases de Datos Relacionales (SQL):**

Las bases de datos relacionales usan principalmente SQL, pero se integran con diversos lenguajes mediante APIs o drivers:

- **Python**: Utiliza bibliotecas como SQLite3, PyMySQL, psycopg2 y ORMs como SQLAlchemy.
- **Java**: Se conecta mediante JDBC a bases como MySQL, PostgreSQL y Oracle.
- **C#**: Usa ADO.NET para interactuar con SQL Server y otras bases.
- **PHP, JavaScript (Node.js), Ruby, Go**: Todos estos lenguajes tienen bibliotecas y ORMs para conectarse a bases de datos SQL.

2. **Lenguajes para Bases de Datos NoSQL:**

Las bases NoSQL utilizan APIs o controladores específicos:

- **Python**: Usa PyMongo (MongoDB), cassandra-driver y redis-py.

- **Java, C#, Go, JavaScript**: Estos lenguajes tienen controladores oficiales o de terceros para bases NoSQL como MongoDB, Cassandra y Redis.
- **Ruby y PHP**: También se conectan a NoSQL mediante bibliotecas especializadas.

Integración de control de versiones.

El control de versiones en bases de datos es esencial para gestionar y rastrear cambios en el esquema, procedimientos y datos críticos. Aunque los gestores de bases de datos como MySQL no ofrecen un sistema de control de versiones nativo, existen herramientas especializadas que permiten versionar los cambios de manera eficiente.

Herramientas de Control de Versiones para Bases de Datos.

1. **Liquibase.**

Es una herramienta de código abierto que permite gestionar cambios en el esquema de la base de datos que registran modificaciones. Estos cambios pueden almacenarse en un sistema de control de versiones (como Git) y aplicarse automáticamente en diferentes entornos (desarrollo, pruebas, producción). Su principal ventaja es la capacidad de revertir cambios y controlar versiones de manera automatizada.

2. **Flyway**

Se basa en scripts SQL numerados que registran los cambios en la base de datos. Los scripts se almacenan en un repositorio de control de versiones y se aplican secuencialmente en cada entorno. Su simplicidad lo convierte en una opción popular para proyectos donde se utilizan migraciones frecuentes.

3. **Control de versiones manual con Git.**

En algunos casos, los cambios en la base de datos se gestionan mediante scripts SQL almacenados en repositorios como Git. Aunque este método ofrece flexibilidad y control total, depende del cuidado manual de los desarrolladores para garantizar que los scripts se ejecuten en el orden correcto y en todos los entornos.

Sintaxis, extensiones y componentes para la tecnología.

SQL.

Las bases de datos relacionales utilizan SQL, que es un lenguaje con un estándar definido.

1. **Sintaxis**

Utiliza **SQL (Structured Query Language)** como lenguaje estándar. Aunque sigue un estándar, cada sistema de gestión de bases de datos relacionales (SGBD) puede tener pequeñas variaciones en la sintaxis. Ejemplos:

- **MySQL** tiene AUTO_INCREMENT para claves primarias.
- **PostgreSQL** usa SERIAL para auto-incrementar.
- **SQL Server** utiliza IDENTITY para generar valores automáticos.

2. **Extensiones**:

Los SGBD relacionales permiten la expansión de funcionalidades con extensiones. Ejemplos:

- **PostgreSQL**: Extensiones como **PostGIS** (para datos geoespaciales) y **HStore** (manejo de pares clave-valor).
- **MySQL**: Plugins de motores de almacenamiento como **InnoDB** o **MyISAM**.
- **SQL Server**: Integración de **CLR (Common Language Runtime)** para ejecutar código .NET.

3. **Componentes**:

- **Motores de Almacenamiento**: MySQL permite el uso de distintos motores (InnoDB, MyISAM).
- **Soporte JSON**: Aunque son bases de datos relacionales, MySQL y PostgreSQL soportan almacenamiento y manipulación de datos en **JSON**.
- **Índices Geoespaciales**: Funcionalidades para gestionar y consultar eficientemente datos geográficos.

NoSQL.

En el caso de las bases de datos NoSQL, el reconocimiento de sintaxis y las extensiones funcionan de manera diferente en comparación con las relacionales.

1. **Sintaxis**:

NoSQL no sigue un lenguaje estándar como SQL. Cada sistema tiene su propia forma de interactuar con los datos:

- **MongoDB**: Utiliza consultas basadas en **JSON**.
- **Cassandra**: Emplea **CQL (Cassandra Query Language)**, similar a SQL.
- **Redis**: Usa comandos de clave-valor como GET, SET, EXPIRE.

2. **Extensiones**:

Las bases de datos NoSQL permiten añadir módulos y extensiones para aumentar su funcionalidad:

- **MongoDB**: Extensiones como **GridFS** (para grandes archivos) o índices geoespaciales.
- **Redis**: Módulos como **RedisGraph** (para grafos), **RediSearch** (búsqueda en tiempo real), y **RedisJSON**.
- **Cassandra**: Se puede extender con **Apache Spark** para análisis de grandes datos.

3. **Componentes**:

- **Escalabilidad Horizontal**: Capacidad de distribuir datos en múltiples nodos (sharding), presente en **MongoDB** y **Cassandra**.
- **Replicación**: Alta disponibilidad mediante replicación en clusters distribuidos.
- **Índices Geoespaciales**: En MongoDB, para trabajar con datos de ubicación.
- **Almacenamiento en Memoria**: **Redis** almacena datos en memoria, optimizando la velocidad de lectura y escritura.

Frameworkes qué tienen y cuáles son los más populares.

No es común que los SGBD tengan frameworks específicos, pero muchos lenguajes de programación tienen frameworks que integran el uso de bases de datos, como:

- **Django:** Framework web de Python que utiliza un ORM para bases de datos relacionales.
- **Ruby on Rails:** Framework para Ruby que también incluye un ORM para interactuar con bases de datos.
- **Hibernate:** Framework para Java que facilita el mapeo de objetos relacionales.

¿Existe manual de usuario y ayuda?

En los Sistemas de gestión de bases de datos si existen manuales de usuarios y ayuda, pero de cada lenguaje en específico, ya que cada uno tiene sus pequeñas diferencias. Las más usadas en la actualidad son:

- **MySQL:** https://dev.mysql.com/doc/
- **PostgreSQL:** https://www.postgresql.org/docs/
- **SQL Server:** https://docs.microsoft.com/es-es/sql/
- **MongoDB:** https://docs.mongodb.com/

Además, en la actualidad existen diversos foros o comunidades famosas, donde se puede encontrar ayudas, preguntas frecuentes y soluciones a problemas comunes. Esas comunidades pueden ser StackOverflow y grupos en GitHub.

Ventajas y desventajas que presentan frente a otros competidores.

Las bases de datos relacionales (SQL) y las bases de datos no relacionales (NoSQL) están diseñadas para satisfacer diferentes tipos de necesidades y desafíos en el manejo de datos. A continuación, se comparan las ventajas y desventajas de ambas tecnologías para resaltar sus principales diferencias.

SQL (Relacionales)	NoSQL (No Relacionales)
VENTAJAS	
Estructura definida con esquemas claros (tablas).	Flexibilidad en el esquema, ideal para datos no estructurados o semiestructurados (como JSON).
Integridad referencial mediante relaciones y claves foráneas.	Escalabilidad horizontal: se añaden más nodos fácilmente para manejar grandes volúmenes de datos.
Amplio soporte para consultas complejas mediante SQL (joins, subconsultas, etc.).	- Mejor rendimiento en sistemas distribuidos y grandes volúmenes de datos.
Compatibilidad con muchas herramientas de análisis y reportes (BI).	Ideal para aplicaciones en tiempo real y grandes transacciones distribuidas.
Seguridad y consistencia transaccional a través de ACID.	Rendimiento rápido en operaciones de lectura y escritura de datos no estructurados
Extenso soporte para lenguajes de programación y ORMs (Object-Relational Mappers).	Fácil adaptabilidad a cambios en el tipo y estructura de datos.
Comunidad madura y soporte sólido por décadas de uso (PostgreSQL, MySQL, SQL Server, etc.).	Altamente eficiente para aplicaciones como redes sociales, análisis de datos masivos y sistemas de recomendaciones.

SQL (Relacionales)	NoSQL (No Relacionales)
DESVENTAJAS	
Escalabilidad horizontal limitada; más difícil de distribuir geográficamente.	Falta de consistencia estricta en transacciones (usualmente CAP prioriza disponibilidad sobre consistencia).
Estructura rígida; requiere alterar el esquema al cambiar los datos.	Falta de integridad referencial; las relaciones entre datos son más débiles o inexistentes
Requiere recursos de hardware más potentes para manejar grandes volúmenes de datos (escalabilidad vertical).	Carece de soporte avanzado para consultas complejas como joins o subconsultas (MongoDB, por ejemplo).
Puede ser más lento al manejar datos no estructurados o semiestructurados.	Estándares y herramientas menos maduros en comparación con SQL.
Costoso en tiempo y recursos ajustar bases de datos relacionales a aplicaciones web masivamente escalables.	No suele ofrecer consistencia transaccional ACID en todas las operaciones, lo que puede complicar ciertas aplicaciones.
Menor eficiencia en aplicaciones que requieren alta disponibilidad en tiempo real.	Seguridad y control más limitado para ciertos casos en comparación con SQL.

Conclusión.

Los Sistemas de Gestión de Bases de Datos (SGBD) han sido claves para organizar y manejar información de manera eficiente. A lo largo del tiempo, han pasado de modelos antiguos, como el jerárquico, a los modelos relacionales, como MySQL o SQL Server, que son más seguros. Más recientemente, surgieron las bases de datos NoSQL que son más flexibles y se adaptan mejor a grandes volúmenes de datos no estructurados.

Hoy en día, la elección entre usar bases de datos SQL o NoSQL depende de las necesidades de cada aplicación. Mientras las SQL son ideales para datos más estructurados, las NoSQL son perfectas para manejar grandes cantidades de datos de forma rápida y distribuida. Ambos tipos tienen ventajas y desventajas, y su uso depende del proyecto en el que se estén implementando.