BASES DE DATOS

SQL vs NoSQL

Fundamentos, Arquitectura y Casos de Uso

Apunte de Cátedra

Ingeniería en Sistemas de Información

Facultad de Ingeniería — Ciclo Lectivo 2025

■ Bases de Datos Relacionales	■ Bases de Datos NoSQL	■ Comparativa Técnica	■■ Casos de Uso

Resumen

El presente apunte tiene como objetivo introducir al estudiante en el universo de las bases de datos, abordando de forma rigurosa y progresiva los conceptos fundamentales del modelo relacional (SQL) y los modelos no relacionales (NoSQL). Se analizan sus arquitecturas, ventajas, limitaciones y criterios de selección, con ejemplos prácticos y casos de uso reales de la industria.

Índice de Contenidos

1. Introducción a las Bases de Datos
   - 1.1 ¿Qué es una Base de Datos?
   - 1.2 SGBD — Sistema de Gestión de Bases de Datos
   - 1.3 Historia y Evolución
2. Modelo Relacional — SQL
   - 2.1 Conceptos Fundamentales
   - 2.2 Estructura: Tablas, Filas y Columnas
   - 2.3 Clave Primaria y Clave Foránea
   - 2.4 Relaciones entre Tablas
   - 2.5 El Lenguaje SQL
   - 2.6 Propiedades ACID
   - 2.7 Normalización
3. Bases de Datos NoSQL
   - 3.1 ¿Por qué surgió NoSQL?
   - 3.2 Tipos de Bases de Datos NoSQL
   - 3.3 Documentos — MongoDB
   - 3.4 Clave-Valor — Redis
   - 3.5 Grafos — Neo4j
   - 3.6 Columnar — Cassandra
   - 3.7 Teorema CAP
4. Tabla Comparativa: SQL vs NoSQL
5. Cuándo Usar SQL y Cuándo NoSQL
6. Ejercicios y Preguntas de Examen

UNIDAD 1

Introducción a las Bases de Datos

1.1 ¿Qué es una Base de Datos?

Una **base de datos** es un conjunto organizado y estructurado de información, almacenada de forma persistente en un soporte informático, de manera que pueda ser accedida, gestionada y actualizada eficientemente. A diferencia de simples archivos de texto o planillas de cálculo, una base de datos permite manejar grandes volúmenes de información con integridad, consistencia y acceso concurrente.

Para entenderlo con una analogía: imaginemos una biblioteca. Los libros son los **datos**, las estanterías organizadas por temática son la **estructura**, y el sistema de catalogación que permite encontrar cualquier libro en segundos es el **SGBD**.

■ Definición Formal

Una base de datos es una colección de datos interrelacionados, almacenados con redundancia controlada, que sirven a múltiples aplicaciones; los datos se almacenan independientemente de los programas que los usan. (Codd, 1970)

1.2 SGBD — Sistema de Gestión de Bases de Datos

Un **Sistema de Gestión de Bases de Datos (SGBD)** — o DBMS por sus siglas en inglés — es el software que actúa como intermediario entre los usuarios/aplicaciones y los datos almacenados. Sus responsabilidades principales son:

- ■■ **Definición de datos:** Permite describir la estructura de los datos mediante lenguajes de definición (DDL).
- ■ **Manipulación de datos:** Provee mecanismos para consultar, insertar, modificar y eliminar datos (DML).
- ■ **Seguridad y Control de Acceso:** Gestiona permisos y roles para proteger la información sensible.
- ■ **Control de Concurrencia:** Coordina el acceso simultáneo de múltiples usuarios sin conflictos.
- ■ **Recuperación ante Fallos:** Garantiza que los datos puedan restaurarse ante caídas del sistema.

- ■ **Optimización de Consultas:** Elige el plan de ejecución más eficiente para cada consulta.

1.3 Historia y Evolución

Período	Hito / Tecnología	Descripción
1960s	Archivos Secuenciales	Datos en cintas magnéticas. Acceso lineal, sin estructura relacional.
1970	Modelo Relacional	Edgar F. Codd propone el modelo relacional basado en álgebra y cálculo relacional.
1979	Oracle 2.0	Primer SGBD comercial basado en SQL. IBM presenta DB/2.
1986	SQL estándar ANSI	SQL se convierte en estándar internacional de facto para RDBMS.
1990s	MySQL / PostgreSQL	Proliferación de SGBD relacionales open source.
2000s	Web 2.0 & Big Data	Escala masiva de datos impone limitaciones al modelo relacional tradicional.
2007-2009	Surgimiento NoSQL	Google (Bigtable), Amazon (DynamoDB), MongoDB, Cassandra. "NoSQL" como movimiento.
2010s+	NewSQL & Multi-model	Bases de datos que combinan escala horizontal con garantías ACID.

UNIDAD 2

Modelo Relacional — SQL

2.1 Conceptos Fundamentales

El **modelo relacional** fue propuesto por Edgar F. Codd en 1970 en su artículo seminal "A Relational Model of Data for Large Shared Data Banks". Su idea central fue representar la realidad a través de **relaciones matemáticas** (tablas), fundamentadas en la teoría de conjuntos y el álgebra relacional.

El modelo se basa en la premisa de que cualquier entidad del mundo real y las relaciones entre entidades pueden expresarse como tablas bidimensionales (filas y columnas), con una estructura definida previamente mediante un **esquema rígido**.

2.2 Estructura: Tablas, Filas y Columnas

La unidad fundamental del modelo relacional es la **tabla** (también llamada relación). Cada tabla está compuesta por:

- **Columnas (Atributos):** Definen el tipo de dato que se puede almacenar en esa posición. Son el "molde" de la tabla. Cada columna tiene un nombre único y un tipo de dato (INTEGER, VARCHAR, DATE, BOOLEAN, etc.)
- **Filas (Tuplas / Registros):** Cada fila representa una instancia concreta de la entidad. Por ejemplo, un alumno específico en la tabla ALUMNOS.
- **Esquema:** Es la definición formal de la estructura de la tabla. Debe declararse antes de insertar datos.

Ejemplo — Tabla: ESTUDIANTES

id_estudiante	nombre	apellido	fecha_nac	carrera	promedio
1	María	González	2001-03-15	Sistemas	8.75
2	Juan	Pérez	2000-08-22	Civil	7.40
3	Ana	López	2002-11-05	Sistemas	9.10
4	Carlos	Ruiz	1999-06-18	Electrónica	6.85

Figura 2.1: Ejemplo de tabla relacional. Cada columna tiene un tipo definido; cada fila es un registro único.

2.3 Clave Primaria y Clave Foránea

2.3.1 Clave Primaria (PRIMARY KEY)

La **clave primaria** es un atributo (o conjunto de atributos) que identifica de forma única a cada fila de una tabla. Sus propiedades fundamentales son:

- **Unicidad:** No pueden existir dos filas con el mismo valor de clave primaria.
- **No nulidad:** La clave primaria nunca puede ser NULL (valor vacío).
- **Inmutabilidad:** Idealmente no debería cambiar a lo largo del tiempo.

> **■ Ejemplo Práctico**
>
> En la tabla ESTUDIANTES, el campo **id_estudiante** es la clave primaria. Aunque dos estudiantes puedan llamarse igual (Juan Pérez), su id_estudiante siempre será diferente, garantizando identificación unívoca.

2.3.2 Clave Foránea (FOREIGN KEY)

La **clave foránea** es un atributo en una tabla que hace referencia a la clave primaria de otra tabla. Es el mecanismo fundamental para establecer **relaciones** entre tablas y garantizar la **integridad referencial**: no puede existir un registro que apunte a otro que no existe.

2.4 Relaciones entre Tablas

Uno de los pilares del modelo relacional es su capacidad de modelar relaciones entre entidades. Existen tres tipos de cardinalidad:

Tipo	Descripción	Ejemplo
1 a 1 (1:1)	Cada registro de A se relaciona con exactamente uno de B.	Empleado — Legajo Personal
1 a Muchos (1:N)	Un registro de A puede relacionarse con varios de B.	Docente → Materias dictadas
Muchos a Muchos (N:M)	Varios de A con varios de B. Requiere tabla intermedia.	Estudiantes ↔ Materias cursadas

2.5 El Lenguaje SQL

SQL (Structured Query Language — Lenguaje de Consulta Estructurado) es el lenguaje estándar para interactuar con bases de datos relacionales. Se divide en sub-lenguajes:

DDL — Data Definition Language (Definición de Datos)

Permite crear, modificar y eliminar estructuras de datos.

```
-- Crear tabla de materias
CREATE TABLE materias (
id_materia INTEGER PRIMARY KEY,
nombre VARCHAR(100) NOT NULL,
creditos INTEGER DEFAULT 4,
id_depto INTEGER REFERENCES departamentos(id_depto)
);

-- Agregar una columna nueva
ALTER TABLE materias ADD COLUMN modalidad VARCHAR(20);

-- Eliminar la tabla
DROP TABLE materias;
```

DML — Data Manipulation Language (Manipulación de Datos)

Permite consultar e insertar/modificar/eliminar registros.

```
-- Insertar un registro
INSERT INTO estudiantes (id_estudiante, nombre, apellido, carrera)
VALUES (5, 'Laura', 'Martínez', 'Sistemas');

-- Consultar con filtro y orden
SELECT nombre, apellido, promedio
FROM estudiantes
WHERE carrera = 'Sistemas'
AND promedio >= 7.0
ORDER BY promedio DESC;

-- Actualizar un registro
UPDATE estudiantes
SET promedio = 9.30
WHERE id_estudiante = 3;

-- Eliminar un registro
DELETE FROM estudiantes WHERE id_estudiante = 5;
```

JOIN — Uniendo tablas

Una de las operaciones más poderosas de SQL es el **JOIN**, que permite combinar filas de dos o más tablas en base a una condición de igualdad (generalmente clave foránea = clave primaria).

```
-- Mostrar qué materias cursa cada estudiante
SELECT e.nombre, e.apellido, m.nombre AS materia, m.creditos
FROM estudiantes e
INNER JOIN inscripciones i ON e.id_estudiante = i.id_estudiante
INNER JOIN materias m ON i.id_materia = m.id_materia
WHERE e.carrera = 'Sistemas'
ORDER BY e.apellido, m.nombre;
```

Tipo de JOIN	Qué devuelve
INNER JOIN	Solo filas que tienen coincidencia en ambas tablas.
LEFT JOIN	Todas las filas de la tabla izquierda + coincidencias de la derecha (NULL si no hay).
RIGHT JOIN	Todas las filas de la tabla derecha + coincidencias de la izquierda (NULL si no hay).
FULL OUTER JOIN	Todas las filas de ambas tablas, con NULL donde no hay coincidencia.

2.6 Propiedades ACID

Las bases de datos relacionales garantizan que sus **transacciones** (unidades de trabajo) cumplan con las propiedades ACID. Una transacción es una secuencia de operaciones que debe ejecutarse de forma completa o no ejecutarse en absoluto.

Propiedad	Significado	Ejemplo
A — Atomicidad	La transacción es una unidad indivisible. O se ejecuta todo, o nada.	Transferencia bancaria: si falla el débito, se revierte el crédito.
C — Consistencia	La BD pasa de un estado válido a otro estado válido. Las reglas de integridad siempre se respetan.	No puede quedar un saldo negativo si hay restricción de saldo_min >= 0.
I — Aislamiento	Transacciones concurrentes no se interfieren entre sí; el resultado es como si se ejecutaran en serie.	Dos cajeros que venden el último asiento de un vuelo no pueden generar overbooking.
D — Durabilidad	Una vez confirmada (COMMIT), los cambios persisten aunque el sistema falle inmediatamente después.	Si el servidor cae tras un COMMIT, al reiniciar los datos están guardados.

```
-- Ejemplo de transacción bancaria
BEGIN TRANSACTION;

UPDATE cuentas SET saldo = saldo - 5000 WHERE id_cuenta = 101; -- Débito
UPDATE cuentas SET saldo = saldo + 5000 WHERE id_cuenta = 205; -- Crédito

-- Si algo falla aquí, el ROLLBACK deshace todo
COMMIT; -- Solo si ambas operaciones exitosas
```

2.7 Normalización

La **normalización** es el proceso de organizar las tablas de una base de datos para reducir la redundancia de datos y mejorar la integridad. Se aplica progresivamente mediante **Formas Normales (FN)**.

Forma Normal	Regla Principal	Problema que Resuelve
1FN — Primera	Cada celda contiene un valor atómico (no listas ni grupos repetidos). Hay clave primaria.	Grupos repetitivos: "Materias: Álgebra, Análisis" en una sola celda.
2FN — Segunda	Cumple 1FN + todos los atributos no-clave dependen de la clave primaria completa.	Dependencia parcial en claves compuestas.
3FN — Tercera	Cumple 2FN + no hay dependencias transitivas (atributo A → B → C no clave).	Redundancia: guardar ciudad y código postal cuando ciudad depende del CP.
FNBC — Boyce-Codd	Versión más estricta de 3FN para casos con múltiples claves candidatas superpuestas.	Anomalías residuales no capturadas por 3FN.

■ Regla Práctica de los 3FN

El objetivo de las tres primeras formas normales puede resumirse como: "Cada atributo no-clave debe depender de la clave completa (2FN), solo de la clave (3FN), y nada más que la clave (FNBC)."

UNIDAD 3

Bases de Datos NoSQL

3.1 ¿Por qué surgió NoSQL?

A mediados de los 2000, con la explosión de la Web 2.0 y las redes sociales, empresas como Google, Amazon, Facebook y Twitter enfrentaron desafíos que el modelo relacional tradicional no podía resolver de forma eficiente:

- ■ **Volumen masivo de datos:** Petabytes de información generada diariamente que superan la capacidad de un único servidor.
- ■ **Alta velocidad de escritura/lectura:** Millones de operaciones por segundo en tiempo real (tweets, likes, transacciones).
- ■ **Variedad de tipos de datos:** Datos semiestructurados (JSON, XML), grafos de relaciones sociales, series temporales.
- ■ **Esquema flexible:** Los datos cambian de estructura frecuentemente durante el desarrollo ágil.
- ■ **Escalabilidad económica:** Escalar horizontalmente (agregar más servidores) es más barato que escalar verticalmente (servidor más potente).

En este contexto, Google publicó en 2006 su paper sobre **Bigtable** y Amazon presentó **Dynamo** en 2007, sentando las bases de lo que se denominaría el movimiento **NoSQL** (Not Only SQL — No Solo SQL). El término fue popularizado en 2009.

■■ ¿Qué significa NoSQL?

El término "NoSQL" no implica que estas bases de datos sean opuestas a SQL ni que lo reemplacen. En realidad, significa "Not Only SQL" — muchos sistemas NoSQL incluso ofrecen lenguajes de consulta similares a SQL. La diferencia fundamental está en el modelo de datos y las garantías de consistencia.

3.2 Tipos de Bases de Datos NoSQL

No existe "una" base de datos NoSQL; es una familia de soluciones con modelos de datos completamente distintos, cada uno optimizado para un tipo específico de problema:

Tipo	Modelo de Datos	Motor Representativo	Caso de Uso Típico
Clave-Valor	Par llave → valor (como un diccionario/mapa).	Redis, DynamoDB, Riak	Caché, sesiones de usuario, contadores.
Documental	Documentos semiestructurados (JSON/BSON).	MongoDB, CouchDB, Firestore	APIs REST, catálogos de productos, CMS.
Columnar (Wide Column)	Tablas con columnas dinámicas por fila.	Apache Cassandra, HBase	Series temporales, IoT, Big Data analytics.
Grafo	Nodos, aristas y propiedades.	Neo4j, Amazon Neptune	Redes sociales, motores de recomendación, fraude.
Series Temporales	Datos indexados por tiempo.	InfluxDB, TimescaleDB	Métricas de sistemas, sensores, logs.
Motor de Búsqueda	Índice invertido full-text.	Elasticsearch, Solr	Búsqueda de texto, logs, análisis de datos.

3.3 Bases de Datos Documentales — MongoDB

MongoDB es el motor NoSQL documental más popular del mundo. Almacena datos en documentos con formato **BSON** (Binary JSON), permitiendo estructuras anidadas y arrays. No requiere esquema predefinido: cada documento puede tener campos distintos.

Terminología: SQL vs MongoDB

SQL	MongoDB	Descripción
Database	Database	Contenedor principal de datos.
Table (Tabla)	Collection (Colección)	Agrupa documentos relacionados.
Row (Fila)	Document (Documento)	Unidad básica de dato, formato JSON/BSON.
Column (Columna)	Field (Campo)	Atributo del documento.

Primary Key	_id Field	Identificador único del documento (ObjectId por defecto).
JOIN	Embedding / Linking	Documentos embebidos o referencias entre colecciones.
GROUP BY	Aggregation Pipeline	Framework de transformación y agrupación de datos.
INDEX	Index	Índice para acelerar consultas.

Documentos Embebidos vs Referenciados

Una decisión clave en MongoDB es cómo relacionar datos. A diferencia de SQL (que siempre usa tablas separadas + JOIN), MongoDB permite dos estrategias:

```
// ESTRATEGIA 1: Embedding (Embebido)
// Un documento de estudiante incluye sus materias DENTRO del mismo documento
{
"_id": ObjectId("507f1f77bcf86cd799439011"),
"nombre": "María González",
"carrera": "Sistemas",
"materias": [
{ "nombre": "Bases de Datos", "nota": 9, "año": 2024 },
{ "nombre": "Algoritmos", "nota": 8, "año": 2024 }
]
}
// ESTRATEGIA 2: Referencing (Referenciando)
// El documento guarda solo el ID de otro documento
{
"_id": ObjectId("507f1f77bcf86cd799439011"),
"nombre": "María González",
"carrera": "Sistemas",
"id_inscripciones": [ObjectId("..."), ObjectId("...")]
}
```

Consultas en MongoDB

```
// Insertar un documento
db.estudiantes.insertOne({
nombre: "Carlos", apellido: "Ruiz", carrera: "Electrónica", promedio: 8.5
})
```

```
// Consultar con filtro (equivalente a SELECT ... WHERE)
db.estudiantes.find(
{ carrera: "Sistemas", promedio: { $gte: 7.0 } },
{ nombre: 1, apellido: 1, promedio: 1, _id: 0 }
).sort({ promedio: -1 })

// Aggregation Pipeline (equivalente a GROUP BY + funciones)
db.estudiantes.aggregate([
{ $match: { promedio: { $gte: 6 } } },
{ $group: { _id: "$carrera", promedio_carrera: { $avg: "$promedio" }, total: { $sum: 1 } } },
{ $sort: { promedio_carrera: -1 } }
])
```

3.4 Bases de Datos Clave-Valor — Redis

Redis (Remote Dictionary Server) es una base de datos en memoria (in-memory), extremadamente rápida, que almacena pares clave → valor. Es la solución más común para **caché**, gestión de **sesiones** y **colas de mensajes**.

```
# Almacenar y recuperar un valor simple
SET usuario:1001:nombre "María González"
GET usuario:1001:nombre # → "María González"

# Valor con tiempo de expiración (TTL)
SETEX sesion:tok_abc123 3600 "datos_de_sesion" # Expira en 1 hora

# Estructura Hash (como un mini-objeto)
HSET usuario:1001 nombre "María" carrera "Sistemas" promedio "8.75"
HGETALL usuario:1001

# Lista (cola FIFO)
LPUSH cola_emails "email_bienvenida_user5"
RPOP cola_emails # Consume el primero de la cola
```

3.5 Bases de Datos de Grafo — Neo4j

Las **bases de datos de grafo** modelan los datos como una red de **nodos** (entidades) conectados por **aristas** (relaciones), cada uno con propiedades. Son ideales cuando las relaciones entre datos son tan importantes como los datos mismos.

Neo4j usa el lenguaje **Cypher** para consultas. Un caso de uso clásico es la red social: encontrar "amigos de mis amigos que viven en Buenos Aires".

```
// Crear nodos y relaciones en Cypher
CREATE (maria:Persona {nombre: "María", ciudad: "Buenos Aires"})
CREATE (juan:Persona {nombre: "Juan", ciudad: "Córdoba"})
CREATE (ana:Persona {nombre: "Ana", ciudad: "Buenos Aires"})
CREATE (maria)-[:AMIGO_DE]->(juan)
CREATE (juan)-[:AMIGO_DE]->(ana)

// ¿Quiénes son amigos de los amigos de María en Buenos Aires?
MATCH (maria:Persona {nombre: "María"})-[:AMIGO_DE*2]->(sugerido:Persona)
WHERE sugerido.ciudad = "Buenos Aires"
RETURN sugerido.nombre
```

3.6 Bases de Datos Columnares — Apache Cassandra

Apache Cassandra es una base de datos distribuida diseñada para manejar grandes volúmenes de datos con alta disponibilidad y sin punto único de falla. Fue desarrollada originalmente por Facebook para la gestión de su bandeja de mensajes.

A diferencia de SQL, en Cassandra el diseño de tablas se basa en los **patrones de consulta** (query-driven design): primero se define qué consultas se harán, luego se diseña la tabla.

3.7 Teorema CAP

El **Teorema CAP** (Brewer, 2000) establece que un sistema distribuido solo puede garantizar **dos** de las siguientes tres propiedades simultáneamente:

Propiedad	Significado	Consecuencia si se pierde
C — Consistency (Consistencia)	Todos los nodos del clúster ven los mismos datos al mismo tiempo.	Lecturas pueden devolver datos desactualizados.
A — Availability (Disponibilidad)	El sistema siempre responde a las solicitudes, aunque algún nodo falle.	El sistema puede volverse inaccesible ante fallos.
P — Partition Tolerance (Tolerancia a Particiones)	El sistema funciona aunque algunos nodos no puedan comunicarse entre sí.	El sistema deja de funcionar si la red se divide.

> **■ Implicación del Teorema CAP**
>
> En la práctica, la partición de red (P) es inevitable en sistemas distribuidos. Por eso la elección real es: CP (consistencia + tolerancia, sacrificando disponibilidad) o AP (disponibilidad + tolerancia, sacrificando consistencia estricta). MongoDB elige CP; Cassandra y DynamoDB eligen AP.

UNIDAD 4

Tabla Comparativa: SQL vs NoSQL

4.1 Comparativa General

Criterio	SQL (Relacional)	NoSQL
Modelo de Datos	Tablas con filas y columnas. Esquema fijo.	Documentos, grafos, clave-valor, columnar. Esquema flexible.
Esquema	Rígido (schema-on-write). Debe definirse antes.	Flexible (schema-on-read). Puede evolucionar sin migraciones.
Lenguaje de Consulta	SQL estándar universal.	Lenguaje propio por motor (MQL, CQL, Cypher, etc.).
Transacciones	ACID garantizado nativo.	BASE en muchos casos. ACID solo en algunos motores.
Relaciones	JOIN entre tablas. Relaciones explícitas.	Embebido o referencias. Join limitado o inexistente.
Escalabilidad	Vertical (más CPU/RAM al servidor).	Horizontal (más servidores/nodos al clúster).
Consistencia	Fuerte (Strong Consistency).	Eventual Consistency en la mayoría.
Madurez	Más de 50 años. Muy probado.	15-20 años. En constante evolución.
Curva de Aprendizaje	SQL estándar, bien documentado.	Varía por motor. Conceptos nuevos.
Casos de Uso	Banca, ERP, e-commerce, RRHH.	Big Data, tiempo real, IoT, redes sociales.
Motores Populares	PostgreSQL, MySQL, Oracle, SQL Server.	MongoDB, Redis, Cassandra, Neo4j, Elasticsearch.

4.2 ACID vs BASE

Mientras SQL garantiza propiedades ACID, muchos sistemas NoSQL distribuidos siguen el modelo **BASE** para lograr alta disponibilidad y escalabilidad:

ACID (SQL)	BASE (NoSQL Distribuido)
Atomicity — Todo o nada	Basically Available — El sistema responde siempre, incluso con datos parcialmente inconsistentes
Consistency — Estado siempre válido	Soft state — El estado puede cambiar con el tiempo, incluso sin nuevas escrituras
Isolation — Transacciones aisladas	Eventually Consistent — El sistema alcanzará consistencia en algún momento futuro
Durability — Cambios permanentes	(La durabilidad se mantiene, pero la consistencia puede diferirse)

UNIDAD 5

Cuándo Usar SQL y Cuándo NoSQL

5.1 Criterios de Selección

La elección entre SQL y NoSQL no es una cuestión de "cuál es mejor" — ambos son herramientas diseñadas para contextos distintos. Un arquitecto de software competente elige en base a los requisitos concretos del sistema. Los factores clave a evaluar son:

Elegir SQL cuando...

- Los datos tienen una estructura bien definida y estable (pocos cambios de esquema).
- La integridad y consistencia de los datos son críticos (sistemas financieros, médicos, legales).
- Se requieren transacciones complejas con múltiples entidades relacionadas.
- Las consultas son ad-hoc y complejas (muchos JOINs, subconsultas, agregaciones).
- El equipo tiene experiencia con SQL y el volumen no justifica la complejidad de NoSQL.
- El volumen de datos es manejable en un servidor (hasta algunos cientos de GB o pocos TB).

Elegir NoSQL cuando...

- El esquema de datos es variable o evoluciona frecuentemente (productos con atributos distintos).
- Se necesita escalar horizontalmente a múltiples servidores de forma económica.
- Los volúmenes son extremadamente grandes (petabytes) o las tasas de escritura son masivas.
- La aplicación requiere baja latencia garantizada para lecturas/escrituras (caché, sesiones).
- Los datos son naturalmente documentales (APIs JSON), grafos (redes) o series temporales.
- La disponibilidad continua (alta disponibilidad) es más importante que la consistencia estricta.

5.2 Casos de Uso Reales de la Industria

Sistema	Empresa / Dominio	SQL o NoSQL	Justificación
Sistema Bancario Central	Banco / Fintech	SQL (PostgreSQL)	ACID crítico. Cada centavo debe cuadrar. Transacciones complejas.

Catálogo de Productos	E-commerce (Amazon, MercadoLibre)	NoSQL (MongoDB)	Productos con atributos muy distintos (ropa vs electrónica vs libros).
Feed de Redes Sociales	Twitter / Instagram	NoSQL (Cassandra / Redis)	Millones de escrituras por segundo. Disponibilidad > consistencia.
Sistema de RRHH	Empresa mediana-grande	SQL (MySQL, Oracle)	Estructura estable. Relaciones complejas entre empleados, roles, nómina.
Motor de Recomendaciones	Netflix / Spotify	NoSQL (Neo4j / DynamoDB)	Relaciones "usuarios-que-vieron-esto-también-vieron" → grafo natural.
Logs y Monitoreo	DevOps / Infraestructura	NoSQL (Elasticsearch)	Terabytes de logs. Búsqueda full-text. Time-series queries.
Sistema de Salud	Hospital / PAMI	SQL (PostgreSQL)	Historia clínica tiene estructura estable. Integridad de datos vital.
Sesiones de Usuarios Web	Cualquier webapp	NoSQL (Redis)	Millones de sesiones. Acceso en microsegundos. TTL automático.

■■ Arquitectura Políglota (Polyglot Persistence)

Las aplicaciones empresariales modernas suelen usar múltiples motores de bases de datos en paralelo, cada uno para lo que mejor hace. Por ejemplo: PostgreSQL para datos transaccionales, Redis para caché, Elasticsearch para búsqueda, y MongoDB para contenido flexible. Esto se denomina "Polyglot Persistence" y es una práctica estándar en arquitecturas de microservicios.

UNIDAD 6

Ejercicios y Preguntas de Examen

6.1 Preguntas Conceptuales

1. Explique la diferencia entre una clave primaria y una clave foránea. ¿Por qué la integridad referencial es importante?
2. Describa las propiedades ACID con un ejemplo concreto de una transacción bancaria.
3. ¿Por qué el modelo relacional requiere un esquema rígido? ¿Cuál es la ventaja y desventaja de esto frente al esquema flexible de MongoDB?
4. Explique el Teorema CAP. ¿Por qué en la práctica la elección real es entre CP y AP?
5. ¿Cuándo usaría embedding y cuándo referencing en MongoDB? Justifique con un ejemplo.
6. Desarrolle un escenario de negocio donde elegiría SQL y otro donde elegiría NoSQL. Fundamente cada decisión.
7. ¿Qué es la normalización? Explique la 1FN, 2FN y 3FN con ejemplos.
8. Explique qué significa "Eventual Consistency" y en qué se diferencia de "Strong Consistency".

6.2 Ejercicios Prácticos

Ejercicio 1 — Diseño de Esquema SQL

Diseñe el modelo relacional completo (tablas, columnas, tipos de datos, claves primarias y foráneas) para el siguiente sistema:

■ Enunciado

Una universidad desea gestionar: Facultades, Carreras (pertenecen a una facultad), Materias (pertenecen a una carrera, tienen correlatividades entre sí), Profesores (pueden dictar múltiples materias), Alumnos (están inscriptos en una carrera y pueden cursar múltiples materias), Inscripciones a materias (con nota final y año).

a) Dibuje el Diagrama Entidad-Relación (DER).
b) Escriba las sentencias CREATE TABLE para todas las entidades.
c) Escriba una consulta SQL que liste los alumnos con promedio superior a 8 junto con su carrera.

d) Identifique posibles violaciones de normalización y proponga correcciones.

Ejercicio 2 — MongoDB vs SQL

Dado el siguiente documento MongoDB de un producto de e-commerce, responda:

```
{
"_id": ObjectId("..."),
"nombre": "Zapatillas Running Pro X",
"marca": "Nike",
"precio": 29999.99,
"categorias": ["calzado", "deportivo", "running"],
"variantes": [
{ "talle": 40, "color": "negro", "stock": 15 },
{ "talle": 41, "color": "negro", "stock": 8 },
{ "talle": 40, "color": "blanco", "stock": 3 }
],
"especificaciones": {
"material_suela": "goma", "peso_gramos": 280, "drop_mm": 10
},
"reseñas": [
{ "usuario": "user_123", "puntuacion": 5, "comentario": "Excelentes!" },
{ "usuario": "user_456", "puntuacion": 4, "comentario": "Muy cómodas" }
]
}
```

a) ¿Cómo representaría este mismo producto en un modelo relacional SQL? Dibuje las tablas necesarias.
b) Escriba la consulta MongoDB para encontrar todos los productos Nike con stock disponible en talle 41.
c) Analice las ventajas y desventajas de usar embedding (como en el documento) vs referencing para las reseñas.

6.3 Glosario de Términos Clave

Término	Definición
SGBD / DBMS	Software que gestiona el almacenamiento, recuperación y administración de datos.
DDL	Data Definition Language. Comandos para definir estructura: CREATE, ALTER, DROP.

DML	Data Manipulation Language. Comandos para datos: SELECT, INSERT, UPDATE, DELETE.
ACID	Atomicidad, Consistencia, Aislamiento, Durabilidad. Propiedades de transacciones confiables.
BASE	Basically Available, Soft state, Eventually consistent. Modelo alternativo para sistemas distribuidos.
JOIN	Operación SQL que combina filas de dos tablas en base a una condición.
Índice	Estructura de datos que acelera las consultas a costa de espacio en disco.
Sharding	Técnica de distribución horizontal de datos en múltiples servidores (NoSQL).
Replicación	Copia de datos en múltiples nodos para alta disponibilidad y tolerancia a fallos.
Schema	Definición formal de la estructura de los datos (tablas, tipos, restricciones).
ORM	Object-Relational Mapping. Framework que mapea objetos del código a tablas SQL.
Throughput	Número de operaciones por unidad de tiempo que puede procesar el sistema.
Latencia	Tiempo que tarda el sistema en responder a una solicitud individual.