Examen 1

TEMA 1 · BIG DATA – INTRODUCCIÓN (RESUMEN EXPLICATIVO)

1. El origen del Big Data: el problema real

Big Data surge como **respuesta a un problema práctico**, no como una moda tecnológica.

Durante años, las organizaciones han almacenado datos de forma tradicional: primero en papel, luego en hojas de cálculo y finalmente en bases de datos relacionales. Este modelo funcionaba bien **mientras los datos eran pocos, estructurados y generados lentamente**.

El problema aparece cuando los datos:

- Crecen de forma masiva
- Provienen de múltiples fuentes distintas
- Se generan de manera continua
- No tienen una estructura fija

En ese momento, los sistemas tradicionales **dejan de ser suficientes**, tanto por capacidad como por velocidad y flexibilidad. Es aquí donde nace el concepto de **Big Data**.

👉 **Idea clave de examen**:

> Big Data aparece cuando los métodos tradicionales no pueden gestionar la cantidad, velocidad y diversidad de los datos.

---

2. ¿Qué es realmente Big Data?

Big Data no significa únicamente “muchos datos”.

El término hace referencia al **conjunto de datos**, pero también a las **tecnologías, arquitecturas y métodos** necesarios para **almacenarlos, procesarlos y analizarlos** con el objetivo de **extraer valor**.

Por tanto, Big Data puede entenderse desde cuatro perspectivas complementarias:

- Como un **problema de datos masivos**
- Como un **conjunto de tecnologías**
- Como un **modelo de procesamiento distribuido**
- Como una **base para analítica avanzada e inteligencia artificial**

👉 **Frase ancla para memorizar**:

> Big Data es la capacidad de transformar grandes volúmenes de datos complejos en conocimiento útil.

---

3. La naturaleza de los datos

Uno de los grandes cambios que introduce Big Data es que **no todos los datos son iguales**.

Los **datos estructurados** son aquellos que siguen un esquema fijo: filas, columnas, tipos definidos. Son fáciles de almacenar y consultar, pero representan solo una parte del mundo real.

Los **datos no estructurados** no siguen ningún formato predefinido. Incluyen texto libre, imágenes, audio, vídeo o correos electrónicos. Son los más abundantes hoy en día y también los más difíciles de analizar.

Entre ambos encontramos los **datos semi-estructurados**, que no tienen un esquema rígido pero sí cierta organización interna mediante etiquetas o marcas, como ocurre con JSON o XML.

👉 **Idea clave**:

> El verdadero reto del Big Data no son los datos estructurados, sino los no estructurados y semi-estructurados.

---

4. Las V del Big Data: el marco conceptual

Para definir formalmente Big Data se utilizan las llamadas **V**.

La **V de Volumen** hace referencia a la enorme cantidad de datos generados y almacenados.

La **V de Velocidad** indica la rapidez con la que los datos se generan y deben ser procesados, muchas veces en tiempo real.

La **V de Variedad** representa la diversidad de formatos, fuentes y tipos de datos.

Con el tiempo, este modelo se amplía para reflejar mejor la realidad:

- **Veracidad**, relacionada con la calidad de los datos
- **Valor**, que recuerda que los datos solo importan si generan conocimiento útil
- **Variabilidad**, ya que los datos y su significado cambian
- **Visualización**, clave para interpretar la información
- **Vulnerabilidad**, ligada a la seguridad y privacidad

👉 **Regla mnemotécnica**:

> Sin valor, las demás V no sirven de nada.

---

5. De dónde provienen los datos

Los datos en Big Data no proceden de una única fuente. Al contrario, su riqueza está en la **diversidad de orígenes**.

Una parte importante procede de las **personas**: correos electrónicos, redes sociales, formularios o navegación web.

Otra gran fuente son las **máquinas**: sensores, dispositivos IoT, logs de sistemas.

También existen datos derivados de **transacciones**, como compras, pagos o reservas, y de la **analítica digital**, que registra el comportamiento de los usuarios en entornos online.

👉 **Idea clave**:

> Big Data integra datos heterogéneos que antes estaban aislados.

---

6. Big Data, Business Intelligence y Data Science

El temario distingue claramente entre **Business Intelligence** y **Data Science**.

Business Intelligence se centra en **analizar el pasado y el presente**, ayudando a responder qué ha ocurrido y por qué. Su objetivo principal es apoyar la toma de decisiones mediante informes y análisis descriptivos y diagnósticos.

Data Science va un paso más allá. Utiliza estadística avanzada, machine learning e inteligencia artificial para **predecir comportamientos futuros** y **recomendar acciones**, entrando en la analítica predictiva y prescriptiva.

👉 **Comparación mental rápida**:

> BI mira hacia atrás; Data Science mira hacia adelante.

---

7. El enfoque de proyecto en Big Data

Un proyecto Big Data no empieza por la tecnología, sino por el **objetivo**.

Primero se define qué se quiere resolver, después se recopilan los datos necesarios, se limpian y preparan, se exploran para comprenderlos, se analizan mediante modelos y finalmente se presentan y automatizan los resultados.

Este proceso es **iterativo**, no lineal: los resultados pueden obligar a volver atrás y mejorar los pasos anteriores.

👉 **Idea clave**:

> En Big Data, el dato sin contexto no tiene valor.

---

8. Beneficios y casos de éxito

El uso correcto de Big Data permite:

- Tomar mejores decisiones

- Anticipar tendencias
- Optimizar procesos
- Personalizar productos y servicios
- Obtener ventajas competitivas

Empresas como Netflix, Amazon o Spotify son ejemplos claros de cómo el análisis masivo de datos se traduce en valor económico real.

---

9. Idea final para memorizar el Tema 1

> Big Data no trata de almacenar datos, sino de convertir datos complejos y masivos en conocimiento útil mediante nuevas tecnologías y enfoques.

TEMA 2 · BIG DATA – ARQUITECTURAS

1. Por qué son necesarias las arquitecturas Big Data

Una arquitectura Big Data no surge por capricho tecnológico, sino por una **necesidad estructural**.

Cuando el volumen de datos crece de forma masiva y estos se generan a gran velocidad y desde múltiples fuentes, **una única máquina deja de ser suficiente** tanto para almacenar como para procesar la información.

Las arquitecturas Big Data se diseñan para **manejar datos que son demasiado grandes, rápidos o complejos** para los sistemas tradicionales, garantizando que el sistema siga funcionando incluso cuando parte de la infraestructura falla.

👉 **Idea clave de examen**:

> Una arquitectura Big Data existe para garantizar escalabilidad, tolerancia a fallos y procesamiento distribuido.

---

2. Principios fundamentales de una arquitectura Big Data

Toda arquitectura Big Data debe cumplir una serie de principios básicos.

El primero es la **escalabilidad**, es decir, la capacidad de crecer añadiendo nuevos recursos sin rediseñar el sistema. En Big Data se prioriza el **escalado horizontal**, añadiendo nodos en lugar de hacer una máquina más potente.

El segundo principio es la **tolerancia a fallos**. En un entorno distribuido se asume que los fallos ocurren, por lo que el sistema debe seguir funcionando aunque uno o varios nodos fallen.

Otro principio esencial es el **procesamiento distribuido**, donde las tareas se reparten entre múltiples máquinas para reducir el tiempo de ejecución.

Finalmente aparece la **localidad del dato**, que indica que los cálculos deben ejecutarse lo más cerca posible de donde están los datos, evitando transferencias costosas por red.

👉 **Frase ancla**:

> En Big Data, los fallos no son una excepción, son la norma.

---

3. Almacenamiento de datos: Data Warehouse vs Data Lake

Antes de analizar arquitecturas complejas, es fundamental entender **cómo y dónde se almacenan los datos**.

El **Data Warehouse** es un repositorio centralizado que almacena **datos estructurados y previamente procesados**. Está orientado al análisis histórico y a la inteligencia de negocio. En este modelo, los datos deben adaptarse a un esquema antes de ser almacenados, lo que se conoce como **schema-on-write**. Esto aporta control y consistencia, pero reduce flexibilidad.

El **Data Lake**, en cambio, almacena los datos **en bruto**, tal y como se generan, sin necesidad de definir un esquema previo. Puede contener datos estructurados, semi-estructurados y no estructurados. El esquema se aplica en el momento de la lectura (**schema-on-read**), lo que ofrece una gran flexibilidad para análisis avanzados y ciencia de datos.

👉 **Idea clave**:

> El Data Warehouse organiza los datos antes de guardarlos; el Data Lake los organiza cuando se usan.

4. Arquitecturas de procesamiento: Batch y Streaming

Las arquitecturas Big Data se diferencian también por **cómo procesan los datos**.

El **procesamiento batch** trabaja con grandes volúmenes de datos acumulados. Tiene un inicio y un fin definidos en el tiempo y prioriza la precisión sobre la rapidez. Es ideal para análisis históricos.

El **procesamiento streaming** trabaja con datos que llegan de forma continua. No tiene un final temporal y busca obtener resultados casi en tiempo real, sacrificando en ocasiones precisión a cambio de velocidad.

Muchas arquitecturas actuales combinan ambos enfoques para aprovechar las ventajas de cada uno.

👉 **Comparación mental**:

> Batch analiza el pasado; Streaming reacciona al presente.

---

5. Arquitectura Lambda

La **arquitectura Lambda** surge para combinar procesamiento batch y streaming en un mismo sistema.

Se basa en tres capas:

La **capa batch** almacena los datos en bruto de forma inmutable y recalcula los resultados sobre todo el histórico. Es la capa más precisa, pero también la más lenta.

La **capa de velocidad (speed layer)** procesa los datos recientes en tiempo real para ofrecer resultados rápidos, aunque menos precisos.

La **capa de servicio (serving layer)** permite consultar los resultados generados por las dos capas anteriores.

El objetivo de Lambda es ofrecer **baja latencia sin renunciar a la precisión**, pero a costa de una mayor complejidad.

👉 **Idea clave de examen**:

> Lambda duplica la lógica de procesamiento para combinar precisión y velocidad.

---

6. Arquitectura Kappa

La **arquitectura Kappa** surge como una simplificación de Lambda.

En lugar de mantener dos caminos de procesamiento, Kappa propone **un único flujo basado exclusivamente en streaming**. Todos los datos se tratan como un flujo continuo y, si es necesario recalcular resultados, se vuelve a procesar el stream completo desde el inicio.

Este enfoque reduce la complejidad, elimina la duplicación de código y facilita el mantenimiento del sistema.

👉 **Frase ancla**:

> En Kappa, todo es un stream.

---

7. Arquitectura por capas

Otra forma de diseñar sistemas Big Data es la **arquitectura por capas**, que separa el ciclo de vida del dato en distintas fases bien definidas.

Estas capas suelen incluir:

- Ingesta de datos
- Transporte y colección
- Procesamiento
- Almacenamiento
- Consulta analítica
- Visualización

Este enfoque facilita la organización del sistema, la escalabilidad y el mantenimiento, ya que cada capa tiene una responsabilidad clara.

👉 **Idea clave**:

> Separar responsabilidades reduce la complejidad del sistema.

8. Infraestructura: On-Premise y Cloud

Las arquitecturas Big Data pueden desplegarse en distintos entornos.

Las arquitecturas **on-premise** se instalan en infraestructura propia. Ofrecen mayor control, pero requieren grandes inversiones iniciales y mantenimiento continuo.

Las arquitecturas **cloud** utilizan recursos bajo demanda ofrecidos por terceros. Permiten escalabilidad, elasticidad y pago por uso, reduciendo costes iniciales.

También existen arquitecturas **híbridas**, que combinan ambos modelos según las necesidades del negocio.

👉 **Idea clave de examen**:

> El cloud convierte el coste fijo en coste variable.

9. Modelos de servicio en la nube

En entornos cloud se distinguen tres modelos principales:

- **IaaS**: se alquila la infraestructura; el cliente gestiona el software.
- **PaaS**: se proporciona una plataforma completa para desarrollar y desplegar aplicaciones.
- **SaaS**: el usuario consume directamente el software sin gestionar la infraestructura.

Cada modelo ofrece un nivel distinto de control y responsabilidad.

10. Idea final para memorizar el Tema 2

> Las arquitecturas Big Data existen para procesar datos masivos de forma distribuida, escalable y tolerante a fallos, adaptándose a distintos tipos de procesamiento y despliegue.

TEMA 3 · HADOOP

1. Hadoop como pieza clave del Big Data

Apache Hadoop es la **tecnología que hace posible el Big Data en la práctica**.

Si Big Data define el problema —datos masivos, rápidos y variados—, Hadoop proporciona la **infraestructura necesaria para resolverlo**.

La idea fundamental de Hadoop es sencilla pero poderosa:

en lugar de usar una sola máquina muy potente, se utilizan **muchas máquinas normales trabajando juntas**, formando un **clúster**. De este modo, el sistema puede crecer fácilmente y seguir funcionando aunque algunos nodos fallen.

👉 **Idea clave de examen**:

> Hadoop está diseñado para trabajar con grandes volúmenes de datos en entornos distribuidos, asumiendo que los fallos son normales.

2. Qué es Apache Hadoop

Apache Hadoop es un **framework open source** orientado al **almacenamiento y procesamiento distribuido de datos**.

No es un único programa, sino un **ecosistema de componentes** que cooperan entre sí.

Hadoop permite:

- Escalar horizontalmente (añadiendo nodos)
- Procesar grandes volúmenes de datos en paralelo
- Recuperarse automáticamente de fallos de hardware
- Utilizar hardware estándar, reduciendo costes

👉 **Frase ancla para memorizar**:

> Hadoop no es un producto, es una plataforma distribuida.

---

3. Características fundamentales de Hadoop

Hadoop se apoya en cuatro características esenciales.

La **escalabilidad** permite ampliar el sistema sin rediseñarlo.

La **tolerancia a fallos** garantiza que el sistema siga funcionando incluso si fallan nodos.

La **fiabilidad** se logra mediante la replicación automática de los datos.

La **portabilidad** permite ejecutarlo en distintos sistemas y entornos.

👉 **Idea clave**:

> Hadoop está pensado para crecer y fallar sin detenerse.

---

4. Arquitectura general de Hadoop

Hadoop se estructura en **cuatro módulos principales**, cada uno con una función clara:

- **Hadoop Common**: librerías y utilidades básicas.
- **HDFS**: sistema de archivos distribuido.
- **YARN**: gestor de recursos del clúster.
- **MapReduce**: modelo de procesamiento distribuido.

👉 **Regla mnemotécnica**:

> Common soporta, HDFS almacena, YARN gestiona y MapReduce procesa.

---

5. HDFS: almacenamiento distribuido (teoría + práctica)

5.1 Concepto de HDFS

HDFS (Hadoop Distributed File System) es el sistema de archivos de Hadoop.

Divide los archivos en **bloques grandes (128 MB)** y los distribuye por el clúster, replicándolos para garantizar tolerancia a fallos.

Funciona bajo el principio **WORM (Write Once, Read Many)**:

los datos se escriben una vez y se leen muchas veces.

👉 **Idea clave**:

> HDFS prioriza rendimiento y fiabilidad, no la modificación de datos.

5.2 Nodos de HDFS

- **NameNode**: gestiona los metadatos (estructura y ubicación de bloques).
- **DataNode**: almacena los datos reales.
- **Secondary NameNode**: ayuda a gestionar los metadatos (no es un backup).

👉 **Idea de examen**:

> El NameNode no guarda datos, guarda información sobre los datos.

5.3 HDFS en el examen práctico: comandos clave

En el examen práctico, HDFS se usa **como un sistema de archivos**, pero con comandos propios.

Todos empiezan por:

```
hdfs dfs
```

Navegación

```
hdfs dfs -ls /ruta
```

Crear directorios

```
hdfs dfs -mkdir /practicas
hdfs dfs -mkdir -p /datos/entradas
```

Subir archivos (MUY IMPORTANTE)

```
hdfs dfs -put archivo.txt /practicas
```

Sobrescribir:

```
hdfs dfs -put -f archivo.txt /practicas
```

Ver archivos

```
hdfs dfs -cat /practicas/archivo.txt
hdfs dfs -head /practicas/archivo.txt
```

Borrar

```
hdfs dfs -rm /practicas/archivo.txt
hdfs dfs -rm -r /practicas
```

👉 **Regla de oro**:

> Si no empieza por hdfs dfs, estás en local, no en HDFS.

---

6. Metadatos en HDFS

HDFS mantiene su estado con dos archivos:

- **fsimage**: imagen del sistema de archivos
- **edits**: registro de cambios

Durante el arranque, ambos se combinan para reconstruir el estado actual.

👉 **Frase ancla**:

> fsimage es la foto, edits es el historial.

---

7. YARN: gestión de recursos (teoría + práctica)

YARN separa la **gestión de recursos** del procesamiento.

Gracias a YARN, Hadoop puede ejecutar distintos tipos de aplicaciones, no solo MapReduce.

En la práctica:

- YARN decide **qué nodo ejecuta cada tarea**
- Permite ver ejecuciones desde la interfaz web (normalmente `localhost:8088`)

👉 **Idea clave**:

> YARN no ejecuta el código, gestiona quién lo ejecuta.

8. MapReduce: procesamiento distribuido (teoría + práctica)

MapReduce divide un problema en tareas pequeñas que se ejecutan en paralelo y luego combina los resultados.

Aunque hoy existen alternativas más rápidas, MapReduce sigue siendo **clave en exámenes**.

8.1 Flujo práctico típico

Compilar

```
export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
hadoop com.sun.tools.javac.Main MyJob.java
```

Crear JAR

```
jar cvf MyJob.jar MyJob*
```

Ejecutar

```
hadoop jar MyJob.jar MyJob /entrada /salida
```

El directorio de salida **no debe existir**.

Ver resultados

```
hdfs dfs -ls /salida
hdfs dfs -cat /salida/part-r-00000
```

👉 **Idea clave de examen práctico**:

> Ejecutar, comprobar y entender el resultado es más importante que el código.

---

9. Errores típicos en el examen práctico

- Usar comandos Linux en lugar de HDFS
- Confundir rutas locales con rutas HDFS
- Ejecutar MapReduce con salida existente
- No comprobar los resultados

👉 **Consejo de examen**:

> Siempre verifica cada paso antes de pasar al siguiente.

---

10. Listado general de comandos para repaso rápido

HDFS

```
hdfs dfs -ls
hdfs dfs -mkdir
hdfs dfs -put
hdfs dfs -get
hdfs dfs -cat
hdfs dfs -head
hdfs dfs -rm
hdfs dfs -rm -r
```

MapReduce

```
export HADOOP_CLASSPATH=...
hadoop com.sun.tools.javac.Main
jar cvf
hadoop jar
```

11. Idea final para memorizar el Tema 3

> Hadoop es la plataforma que permite almacenar y procesar datos masivos de forma distribuida, y en el examen práctico se demuestra sabiendo usar HDFS y ejecutar procesos.

TEMA 4 · INTRODUCCIÓN A LAS BASES DE DATOS

1. Por qué existen las bases de datos

Las bases de datos aparecen como solución a un problema básico: **almacenar información de forma organizada, persistente y accesible**.

Antes de ellas, los datos se guardaban en archivos planos, lo que provocaba duplicidades, errores y dificultades para consultar la información.

Una base de datos permite:

- Centralizar la información
- Evitar redundancias
- Acceder a los datos de forma eficiente
- Mantener integridad y consistencia

👉 **Idea clave de examen**:

> Una base de datos existe para almacenar datos de forma estructurada y controlada.

2. Conceptos fundamentales de bases de datos

Una **base de datos** es un conjunto de datos relacionados entre sí, almacenados de forma estructurada.

Un **SGBD (Sistema Gestor de Bases de Datos)** es el software que permite crear, gestionar y consultar esa base de datos.

Ejemplos de SGBD tradicionales:

- MySQL
- Oracle
- PostgreSQL

👉 **Frase ancla**:

> La base de datos es el dato; el SGBD es quien lo gestiona.

3. Modelos de bases de datos

3.1 Bases de datos relacionales

El modelo relacional organiza la información en **tablas** formadas por:

- Filas (registros)
- Columnas (atributos)

Las tablas se relacionan entre sí mediante **claves**.

Características principales:

- Esquema fijo

- Datos estructurados
- Uso del lenguaje SQL
- Alta consistencia

👉 **Idea clave**:

> El modelo relacional prioriza el orden y la consistencia.

3.2 Bases de datos NoSQL

Las bases de datos NoSQL surgen para responder a las necesidades del Big Data.

Características:

- No usan un esquema rígido
- Escalan horizontalmente
- Manejan grandes volúmenes de datos
- Soportan datos no estructurados

Tipos habituales:

- Clave–valor
- Documentos
- Columnas
- Grafos

👉 **Comparación mental**:

> SQL = estructura y consistencia
> NoSQL = flexibilidad y escalabilidad

---

4. Esquema y estructura de los datos

El **esquema** define cómo se organizan los datos.

En bases de datos relacionales:

- El esquema se define **antes** de insertar los datos
- Cambiarlo es costoso

Este enfoque se conoce como **schema-on-write**.

En entornos Big Data:

- El esquema puede aplicarse al leer los datos
- Mayor flexibilidad

Este enfoque se denomina **schema-on-read**.

👉 **Idea clave de examen**:

> Schema-on-write controla; schema-on-read explora.

---

5. Lenguajes de consulta: SQL

SQL (Structured Query Language) es el lenguaje estándar para trabajar con bases de datos relacionales.

Se divide en varios tipos de comandos:

- **DDL**: definición de estructuras (CREATE, DROP)
- **DML**: manipulación de datos (INSERT, UPDATE, DELETE)
- **DQL**: consultas (SELECT)

👉 **Idea clave**:

> SQL no es solo SELECT, es definición, manipulación y consulta.

---

6. Bases de datos y Big Data

Las bases de datos tradicionales **no desaparecen** con Big Data, pero **no son suficientes por sí solas**.

Limitaciones:

- Escalado vertical
- Dificultad para manejar datos no estructurados
- Coste elevado al crecer

Big Data introduce nuevas formas de almacenamiento y procesamiento, complementando a las BBDD tradicionales.

👉 **Idea clave**:

> Big Data amplía el concepto de base de datos, no lo sustituye.

---

7. Enfoque práctico del Tema 4 en el examen

En el **examen práctico**, este tema no suele pedirte crear una base de datos clásica, pero sí **entender conceptos clave** que luego se aplican en Hive.

Debes saber:

- Qué es una tabla
- Qué es un esquema
- Qué diferencia hay entre SQL y NoSQL
- Qué implica schema-on-write y schema-on-read

Muy típico: preguntas teórico–prácticas que conectan BBDD → Hive.

---

8. Errores típicos de examen

- Pensar que Big Data elimina SQL
- Confundir base de datos con SGBD
- Creer que NoSQL significa “sin estructura”
- No entender el concepto de esquema

👉 **Regla de examen**:

> NoSQL no significa “sin orden”, significa “sin esquema fijo”.

---

9. Conexión directa con el Tema 5 (Hive)

Este tema es **la base directa de Hive**.

- Tablas → tablas Hive
- SQL → HiveQL
- Schema → definición de tablas
- Datos estructurados → HDFS + Hive

👉 **Idea puente**:

> Hive es la adaptación del modelo de base de datos al entorno Big Data.

---

10. Idea final para memorizar el Tema 4

> Las bases de datos organizan los datos; Big Data las amplía para manejar volumen, variedad y escalabilidad.

TEMA 5 · HIVE

1. Por qué aparece Hive

Hive surge para resolver un problema muy concreto:

Hadoop y HDFS son muy potentes para almacenar y procesar datos, pero **no son cómodos para consultar información**.

Trabajar directamente con MapReduce implica:

- Programar en Java
- Mucha complejidad
- Poco enfoque analítico

Hive aparece como una **capa de abstracción** que permite consultar los datos almacenados en HDFS **como si fueran tablas de una base de datos**, utilizando un lenguaje similar a SQL.

👉 **Idea clave de examen**:

> Hive permite consultar datos en HDFS sin programar MapReduce.

---

2. Qué es Hive

Apache Hive es un **sistema de data warehouse** construido sobre Hadoop que:

- Permite definir **tablas**
- Permite ejecutar **consultas tipo SQL**
- Traduce esas consultas a **jobs de procesamiento** (MapReduce, Tez o Spark)

Hive **NO es una base de datos tradicional**:

- No está orientado a transacciones
- Está pensado para análisis por lotes
- Prioriza grandes volúmenes de datos

👉 **Frase ancla**:

> Hive no ejecuta SQL, lo traduce a procesamiento distribuido.

---

3. Arquitectura de Hive

Hive se compone de varios elementos que trabajan juntos.

El **Driver** gestiona la sesión del usuario y coordina la ejecución de las consultas.

El **Metastore** almacena los metadatos: definición de tablas, columnas, tipos y ubicaciones en HDFS.

El **motor de ejecución** se encarga de lanzar los jobs sobre Hadoop.

Las interfaces de acceso pueden ser **Hive CLI**, **Beeline** u otras herramientas externas.

👉 **Idea clave**:

> Hive separa los datos (HDFS) de los metadatos (Metastore).

---

4. Metastore: el corazón de Hive

El Metastore es un repositorio central de metadatos.

Puede funcionar en tres modos:

- **Embebido**: sencillo, para pruebas
- **Local**: metadatos en BBDD local
- **Remoto**: metadatos compartidos (modo profesional)

El Metastore **no guarda datos**, solo información sobre ellos.

👉 **Conexión con Tema 4**:

> El Metastore equivale al esquema de una base de datos.

---

5. Tablas en Hive: internas y externas (MUY IMPORTANTE)

5.1 Tablas internas (managed tables)

En una tabla interna:

- Hive gestiona los datos y los metadatos
- Los datos se almacenan en el warehouse de Hive
- Al borrar la tabla, **se borran los datos**

👉 **Idea clave de examen**:

> Borrar tabla interna = borrar datos.

5.2 Tablas externas (external tables)

En una tabla externa:

- Hive solo gestiona los metadatos
- Los datos están fuera del control de Hive
- Al borrar la tabla, **los datos permanecen**

👉 **Idea clave de examen**:

> Borrar tabla externa ≠ borrar datos.

---

6. Tipos de datos en Hive

Hive soporta **tipos simples** y **tipos complejos**.

Los tipos simples incluyen:

- INT, BIGINT
- FLOAT, DOUBLE
- STRING
- BOOLEAN
- TIMESTAMP

Los tipos complejos permiten modelar datos más ricos:

- **ARRAY**: listas ordenadas
- **MAP**: pares clave–valor
- **STRUCT**: estructuras con campos

👉 **Idea clave**:

> Hive permite trabajar con datos complejos sin desnormalizar.

---

7. HiveQL: el lenguaje de Hive

HiveQL es un lenguaje similar a SQL, pero adaptado a Big Data.

Se divide conceptualmente en:

- **DDL**: definición de estructuras
- **DML**: carga y manipulación de datos
- **SELECT**: consultas

HiveQL no está pensado para operaciones rápidas, sino para **consultas analíticas sobre grandes volúmenes**.

8. Hive en el examen práctico: qué debes saber hacer

En el examen práctico, Hive suele evaluarse a través de tareas como:

- Crear tablas
- Cargar datos
- Ejecutar consultas
- Relacionar tablas

No se busca complejidad, sino **entender el flujo completo**.

9. Comandos y ejemplos prácticos de Hive (CLAVE)

9.1 Bases de datos

```
SHOW DATABASES;
USE ejemplo;
```

9.2 Crear tablas

```
CREATE TABLE empleados (
  nombre STRING,
  edadINT,
  salarioDOUBLE
);
```

Tabla externa:

```
CREATEEXTERNALTABLE empleados_ext (
  nombre STRING,
  edadINT
)
LOCATION'/datos/empleados';
```

9.3 Cargar datos

```
LOAD DATALOCAL INPATH'/home/usuario/empleados.txt'
INTOTABLE empleados;
```

👉 `LOCAL` indica sistema local → HDFS.

9.4 Consultas básicas

```
SELECT*FROM empleados;
SELECT nombre, salarioFROM empleadosWHERE salario>30000;
SELECTCOUNT(*)FROM empleados;
```

9.5 JOINs (muy típico)

```
SELECT e.nombre, d.departamento
FROM empleados e
JOIN departamentos d
ON e.id= d.id;
```

10. Errores típicos en el examen práctico

- Confundir tablas internas con externas
- Pensar que Hive es transaccional
- Olvidar que Hive trabaja sobre HDFS
- No entender qué se borra al hacer DROP TABLE

👉 **Regla de examen**:

> Hive consulta datos, no los modifica como una BBDD clásica.

---

11. Conexión final: Hadoop + BBDD + Hive

- Hadoop aporta el almacenamiento y procesamiento
- Las BBDD aportan el modelo tabular
- Hive une ambos mundos

👉 **Idea puente final**:

> Hive es el SQL del Big Data sobre Hadoop.

---

12. Idea final para memorizar el Tema 5

> Hive permite analizar grandes volúmenes de datos en Hadoop usando un lenguaje similar a SQL, separando datos y metadatos.

GUÍA PRACTICA

HDFS + HIVE

FLUJO GENERAL DEL EXAMEN (MEMORÍZALO)

> 1️⃣ Ver datos → 2️⃣ Preparar HDFS → 3️⃣ Crear tabla → 4️⃣ Cargar datos → 5️⃣ Consultar

Si sigues este orden **nunca te pierdes**.

HDFS – LO JUSTO Y NECESARIO

1.1 Ver qué hay en HDFS

Siempre empieza aquí:

```
hdfs dfs -ls /
```

Si el enunciado dice *"trabaja en /practicas"*:

```
hdfs dfs -ls /practicas
```

1.2 Crear directorio (si lo piden)

```
hdfs dfs -mkdir -p /practicas
```

1.3 Subir datos a HDFS (CASI SIEMPRE)

```
hdfs dfs -put -f fichero.csv /practicas/
```

Verifica SIEMPRE:

```
hdfs dfs -ls /practicas
```

👉 **Regla de oro**:

> Nunca pases a Hive sin comprobar que el fichero está en HDFS.

1.4 Ver datos rápido (para comprobar separador)

```
hdfs dfs -head /practicas/fichero.csv
```

ENTRAR EN HIVE Y PREPARAR EL ENTORNO

En la terminal de Ubuntu:

```
hive
```

Si todo está correcto, verás algo como:

```
hive>
```

Muestra las tablas dentro de hive:

```
SHOW DATABASES;
```

Si el examen pide una base concreta:

```
USE examen
```

Si no dice nada:

👉 trabaja en `default` (no pasa nada).

CREAR TABLA (PASO CRÍTICO DEL EXAMEN)

3.1 ¿INTERNA o EXTERNA?

- **EXTERNA** → si los datos ya están en HDFS o te dicen *"no borrar datos"*
- **INTERNA** → si cargas desde LOCAL o no especifican nada

3.2 PLANTILLA MENTAL DE TABLA (MEMORÍZALA)

Tabla INTERNA

```
CREATE TABLE tabla (
  col1 TIPO,
  col2 TIPO
)
ROW FORMAT DELIMITED
FIELDS TERMINATEDBY',';
```

Tabla EXTERNA (MUY TÍPICA)

```
CREATEEXTERNALTABLE tabla (
  col1 TIPO,
```

```
  col2 TIPO
)
ROW FORMAT DELIMITED
FIELDS TERMINATEDBY','
LOCATION'/practicas';
```

`LOCATION` SIEMPRE apunta al directorio, no al fichero.

3.3 Comprobar estructura

```
DESCRIBE tabla;
```

CARGAR DATOS (EL PASO QUE MÁS SE FALLA)

CASO A: Datos YA en HDFS

👉 **NO uses LOCAL**

```
LOAD DATA INPATH'/practicas/fichero.csv'
INTOTABLE tabla;
```

CASO B: Datos en local

👉 **USA LOCAL**

```
LOAD DATALOCAL INPATH'/home/hadoop/fichero.csv'
INTOTABLE tabla;
```

COMPROBACIÓN OBLIGATORIA

```
SELECTCOUNT(*)FROM tabla;
```

Si da `0` → algo está mal (ruta, separador, tipos).

CONSULTAS (SOLO LO IMPRESCINDIBLE)

Siempre prueba en este orden:

```
SELECT*FROM tabla LIMIT5;
```

Luego ya haces lo que pida el enunciado:

- `WHERE`
- `GROUP BY`
- `ORDER BY`
- `JOIN`

👉 Si una consulta grande falla, vuelve a:

```
SELECTCOUNT(*)FROM tabla;
```

SI HAY DOS TABLAS (JOIN – MUY TÍPICO)

```
SELECT a.col, b.col
FROM tabla1 a
JOIN tabla2 b
ON a.id= b.id;
```

Si no hay resultados:

- revisa tipos (`INT` vs `STRING`)
- revisa valores reales (`SELECT DISTINCT id FROM tabla1 LIMIT 10;`)

BORRAR / REHACER (SIN MIEDO)

```
DROPTABLE tabla;
```

👉 Si es **INTERNA**, borra datos

👉 Si es **EXTERNA**, **NO borra** datos

APÉNDICE · BANCO DE RECURSOS BIG DATA (TEMAS 1–5)

A. GLOSARIO DE CONCEPTOS CLAVE (POR TEMAS)

TEMA 1 · BIG DATA

Concepto	Definición clara y directa
Big Data	Conjunto de tecnologías y técnicas para almacenar, procesar y analizar datos masivos, rápidos y variados que no pueden tratarse con sistemas tradicionales.
Datos estructurados	Datos con esquema fijo (tablas, filas y columnas). Ejemplos: bases de datos SQL, Excel.
Datos no estructurados	Datos sin formato definido. Ejemplos: texto libre, correos, imágenes, vídeos.
Datos semi-estructurados	Datos con cierta organización interna sin esquema rígido. Ejemplos: JSON, XML.
Volumen	Cantidad masiva de datos generados y almacenados.
Velocidad	Rapidez con la que los datos se generan, transmiten y procesan.
Variedad	Diversidad de formatos y fuentes de datos.
Veracidad	Calidad y fiabilidad de los datos.
Valor	Capacidad de los datos para generar información útil.
Visualización	Representación gráfica de los datos para facilitar su comprensión.
Business Intelligence (BI)	Análisis descriptivo y diagnóstico del pasado y presente para apoyar decisiones.
Data Science	Uso de estadística y machine learning para análisis predictivo y prescriptivo.

TEMA 2 · ARQUITECTURAS BIG DATA

Concepto	Definición clara y directa
Arquitectura Big Data	Diseño del sistema que permite escalabilidad, tolerancia a fallos y procesamiento distribuido.
Escalado horizontal	Aumentar capacidad añadiendo nodos al sistema (no máquinas más potentes).
Tolerancia a fallos	Capacidad del sistema para seguir funcionando aunque fallen componentes.
Localidad del dato	Ejecutar los cálculos donde están almacenados los datos.

Concepto	Definición clara y directa
Data Warehouse	Repositorio de datos estructurados y procesados, orientado a análisis.
Schema-on-write	El esquema se define antes de almacenar los datos.
Data Lake	Repositorio de datos en bruto, de cualquier tipo y formato.
Schema-on-read	El esquema se aplica en el momento de la lectura de los datos.
Procesamiento batch	Procesamiento por lotes sobre datos históricos.
Procesamiento streaming	Procesamiento continuo de datos en tiempo real.
Arquitectura Lambda	Combina batch, streaming y serving para precisión y baja latencia.
Arquitectura Kappa	Arquitectura basada solo en streaming, más simple que Lambda.

TEMA 3 · HADOOP

Concepto	Definición clara y directa
Hadoop	Framework open source para almacenamiento y procesamiento distribuido de datos.
Clúster	Conjunto de máquinas que trabajan como un único sistema.
HDFS	Sistema de archivos distribuido de Hadoop.
Bloque HDFS	Unidad mínima de almacenamiento en HDFS (128 MB por defecto).
Factor de replicación	Número de copias de cada bloque en HDFS (normalmente 3).
WORM	Write Once, Read Many: escribir una vez, leer muchas.
NameNode	Nodo maestro que gestiona metadatos (no almacena datos).
DataNode	Nodo que almacena físicamente los datos.
Secondary NameNode	Ayuda a gestionar los metadatos de HDFS (no es backup).
fsimage	Imagen del estado del sistema de archivos HDFS.
edits	Registro de cambios realizados en HDFS.
YARN	Gestor de recursos y planificación de tareas del clúster.
MapReduce	Modelo de procesamiento distribuido por lotes.

TEMA 4 · BASES DE DATOS

Concepto	Definición clara y directa
Base de datos	Conjunto organizado de datos relacionados.
SGBD	Software que permite crear, gestionar y consultar bases de datos.
Modelo relacional	Organización de datos en tablas relacionadas mediante claves.
NoSQL	Bases de datos sin esquema rígido, escalables horizontalmente.
Schema-on-write	El esquema se define antes de insertar los datos.
Schema-on-read	El esquema se define al leer los datos.
SQL	Lenguaje para definir, manipular y consultar datos.
DDL	Lenguaje de definición de datos (CREATE, DROP).
DML	Lenguaje de manipulación de datos (INSERT, UPDATE, DELETE).
DQL	Lenguaje de consulta de datos (SELECT).

TEMA 5 · HIVE

Concepto	Definición clara y directa
Hive	Sistema de data warehouse sobre Hadoop con lenguaje similar a SQL.
HiveQL	Lenguaje de consultas utilizado por Hive.
Metastore	Repositorio central de metadatos de Hive.

Concepto	Definición clara y directa
Tabla interna (managed)	Hive gestiona datos y metadatos; DROP TABLE borra los datos.
Tabla externa (external)	Hive gestiona solo metadatos; DROP TABLE no borra los datos.
Warehouse de Hive	Directorio donde Hive almacena las tablas internas.
ARRAY	Tipo complejo que almacena listas ordenadas.
MAP	Tipo complejo que almacena pares clave–valor.
STRUCT	Tipo complejo que agrupa varios campos con nombre.

B. BANCO DE COMANDOS – CONSULTA RÁPIDA

HDFS (EXAMEN PRÁCTICO)

```
hdfs dfs -ls /ruta# listar
hdfs dfs -mkdir -p /ruta# crear directorios
hdfs dfs -put fichero /ruta# subir a HDFS
hdfs dfs -get /ruta/fichero# bajar a local
hdfs dfs -head fichero# ver primeras líneas
hdfs dfs -cat fichero# ver contenido
hdfs dfs -rm fichero# borrar archivo
hdfs dfs -rm -r ruta# borrar directorio
```

HIVE – ESTRUCTURA BÁSICA

Bases de datos

```
SHOW DATABASES;
USE nombre_bd;
```

Crear tabla interna

```
CREATE TABLE tabla (
  col1 TIPO,
  col2 TIPO
)
ROW FORMAT DELIMITED
FIELDS TERMINATEDBY',';
```

Crear tabla externa

```
CREATEEXTERNALTABLE tabla (
  col1 TIPO,
  col2 TIPO
)
ROW FORMAT DELIMITED
FIELDS TERMINATEDBY','
LOCATION'/ruta_hdfs';
```

Cargar datos

```
LOAD DATALOCAL INPATH'/ruta_local'
INTOTABLE tabla;
```

```
LOAD DATA INPATH'/ruta_hdfs'
INTOTABLE tabla;
```

Consultas de comprobación

```
SELECTCOUNT(*)FROM tabla;
SELECT*FROM tabla LIMIT5;
DESCRIBE tabla;
```

JOIN

```
SELECT a.col, b.col
FROM t1 a
JOIN t2 b
ON a.id= b.id;
```

Borrar

```
DROPTABLE tabla;
```

C. MAPA MENTAL DE EXAMEN (1 VISTA)

```
BigData
 ├─ Arquitecturas
 │   ├─ Batch / Streaming
 │   ├─ Lambda / Kappa
 │   └─Data Lake / Warehouse
 ├─ Hadoop
 │   ├─ HDFS (datos)
 │   ├─ YARN (recursos)
 │   └─ MapReduce (procesa)
 ├─ BBDD
 │   ├─ SQL
 │   └─ NoSQL
 └─ Hive
    ├─ Tablas
    ├─ Metastore
    └─ Consultas
```

D. FRASES CLAVE PARA DESBLOQUEARTE EN EL EXAMEN

- *"Si no empieza por* `hdfs dfs` *, estoy en local."*
- *"Hive no guarda datos, guarda metadatos."*
- *"Tabla interna borra datos, externa no."*
- *"Si COUNT()* da 0, algo está mal antes del SQL."*
- *"Primero datos, luego tablas, luego consultas."*