TEMA 1

MÓDULO:
HERRAMIENTAS DE BIG DATA

# HERRAMIENTAS DE GESTIÓN DEL DATO

**IGNACIO PÉREZ**

Ingenio de Telecomunicaciones por la UAH. Master en Business administration and management por IE Business School. Gerente y arquitecto big data.


Institut de Formació Contínua-IL3
UNIVERSITAT DE BARCELONA

# ÍNDICE

## 5. Procesamiento del dato: Hadoop, Spark y Map Reduce



## Ideas clave

# OBJETIVOS ESPECÍFICOS

- Entender los conceptos de Data Warehouse, Data Mart y Data Lake.
- Entender la diferencia entre datos estructurados, semiestructurados y no estructurados.
- Conocer los problemas existentes previos a la aparición de las tecnologías Big Data y como éstas les han dado solución.
- Conocer las características de las bases de datos relacionales, los distintos tipos existentes y profundizar en el conocimiento de una de la más utilizadas en el mercado (Oracle).
- Conocer las características de la base de datos no relacionales o NoSQL y los distintos tipos existentes, profundizando, además, en el conocimiento de una de la más utilizadas en el mercado (MongoDB).
- Conocer los tres paradigmas de computación: Procesamiento Batch, Procesamiento Streaming y Procesamiento Híbrido.
- Utilizar las herramientas de procesamiento masivo de datos y computación distribuida, profundizando en el conocimiento Spark.

# 1. DATA MART Y DATA LAKES. TIPOLOGÍA DE DATOS: ESTRUCTURADOS Y NO ESTRUCTURADOS

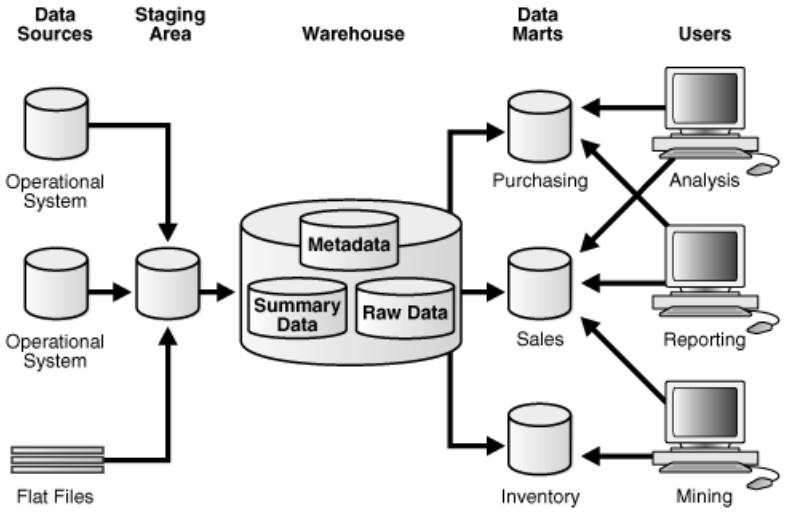
## 1.1. DATA WAREHOUSE Y DATA MART

Un **Data Warehouse** es un almacén de datos único que integra datos de una o más fuentes distintas. Ofrece un repositorio central para almacenar datos actuales e históricos, los cuales se utilizan para generar informes analíticos y de tendencias en una empresa.

Los datos se cargan en el Data Warehouse desde los sistemas que realizan las transacciones diarias de la empresa (por ejemplo, ventas).

Antes de ser almacenados, los datos son procesados y limpiados. Estos procesos son denominados **ETL** (Extracción, Transformación y Carga).

Después de cargarse y procesarse completamente en el Data Warehouse, los conjuntos de datos más grandes pueden tener un subconjunto de información replicado en un **Data Mart** que, generalmente, está orientado a un equipo o línea de negocios específico.




*Fuente: Data Warehousing Concepts*

### PARA SABER MÁS

Fuente: Database Data Warehousing Guide.

## 1.2. DATA LAKE

Un Data Warehouse es un sistema que agrega datos de una o más fuentes y procesa esos datos para que puedan ser utilizados por usuarios específicos. Un **Data Lake** es similar, excepto por algunas diferencias clave:

- Utiliza i**nfraestructura "commodity"** para hacer que el almacenamiento de datos sea menos costoso. Permite el almacenamiento de más datos que un Data Warehouse debido a la reducción de costes.

- Da soporte para más tipos de datos, **semiestructurados y no estructurados.** Permite el almacenamiento de logs de servidores, datos de sensores, actividad de redes sociales y correos electrónicos. Los Data Warehouse suelen almacenar datos estructurados y datos agregados de sistemas transaccionales, lo que reduce la necesidad de almacenar datos no estructurados (correos electrónicos, PDF) o semiestructurados (CSV, XML, JSON).

- Los datos se almacenan sin realizar ningún procesamiento o un mínimo procesado **(Dato Raw)** por lo que se reduce el tiempo necesario para que los datos estén disponibles para su uso en comparación con un Data Warehouse, ya que se requiere poco o ningún trabajo de desarrollo antes de que los datos sin procesar se consuman y estén disponibles.

- Tiene capacidad para ser **utilizado por más personas dentro de la organización** de una empresa. Debido a la capacidad de un Data Lake para almacenar datos sin procesar, podría resultar útil para más personas de lo que se pretendía originalmente. Los usuarios deberán tener más conocimientos para hacer uso de los datos almacenados sin procesar.

- **Adaptabilidad al cambio**. Al no dedicar, inicialmente, tiempo en pensar exactamente cómo se utilizarán los datos, nos permite cambiar el uso de estos. Una vez más, para aprovechar esta flexibilidad, los usuarios deberán tener conocimientos sobre cómo utilizar el Data Lake.

## 1.3. TIPOLOGÍA DE DATOS

Podemos clasificar los datos de acuerdo con tres tipologías:

### 1.3.1. ESTRUCTURADOS

Los datos estructurados son los datos de una empresa que están claramente definidos y cuyo patrón los hace fáciles de buscar.

Normalmente se encuentran en tablas con columnas y filas de datos.

La intersección de la fila y la columna en una celda tiene un valor y se le da una "clave" con la que se puede hacer referencia en las consultas.

Debido a que existe una relación directa entre la columna y la fila, estas bases de datos se conocen comúnmente como bases de datos relacionales.

Un punto de venta que almacena sus datos de ventas (nombre de la persona, producto vendido, cantidad) en una hoja de cálculo de Excel o archivo CSV son ejemplos de datos estructurados.

### 1.3.2. SEMIESTRUCTURADOS

Los datos semiestructurados también tienen una organización, pero la estructura de la tabla se elimina para que los datos se puedan leer y manipular más fácilmente. Los archivos XML o una fuente RSS para una página web son ejemplos de datos semiestructurados.

### 1.3.3. NO ESTRUCTURADOS

Los datos no estructurados son aquellos que, generalmente, no son tan fáciles de buscar, incluidos formatos como audio, video y redes sociales, etc.

Se encuentran en todas partes, por ejemplo, en mensajes de texto, correos electrónicos y redes sociales.

Los datos no estructurados, generalmente, no tienen una estructura organizativa y las tecnologías Big Data utilizan diferentes formas de agregar estructura a dichos datos.

# 2. BIG DATA: VOLUMEN, VELOCIDAD, VARIEDAD, VERACIDAD Y VALOR

## 2.1. INTRODUCCIÓN

La definición oficial de Big Data podemos tomarla del glosario de referencia de conceptos tecnológicos de la revista Gartner.

CITA

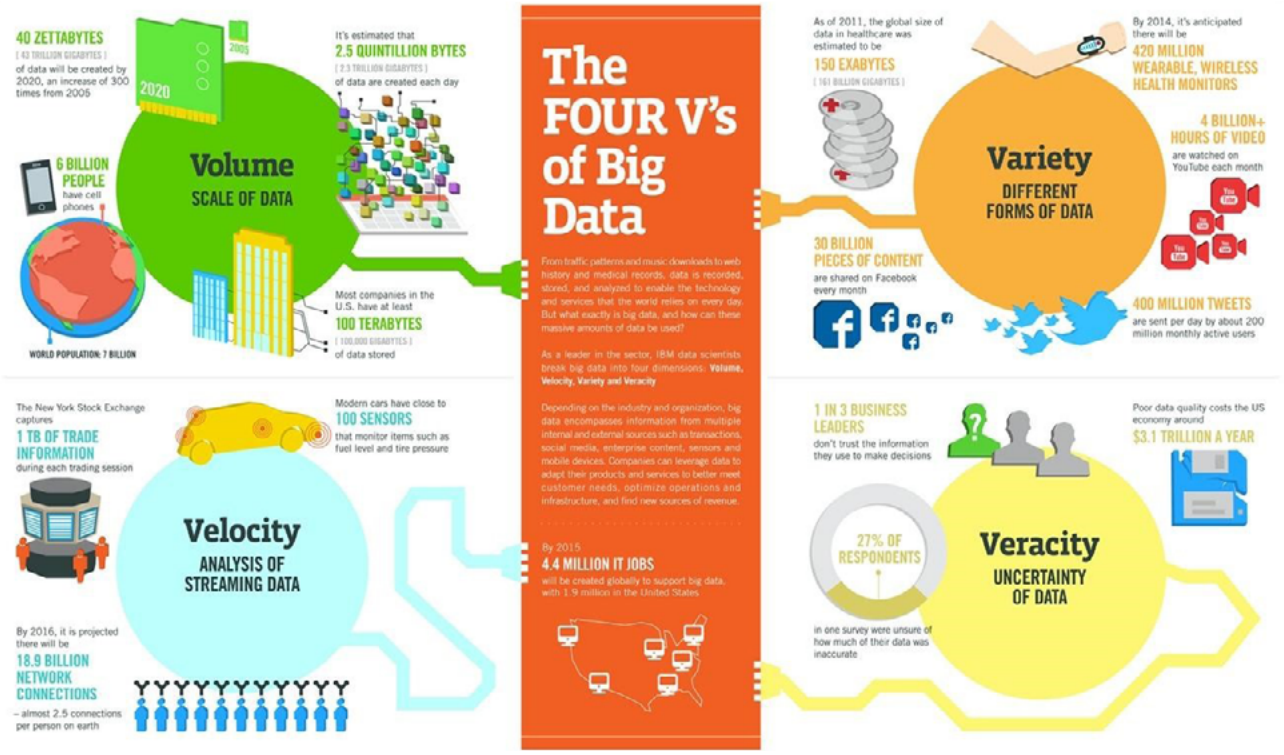
> "**Big Data** is high-**volume,** high-**velocity** and/or high-**variety** information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."
>
> (IT Glossary. Gartner)

Big Data está relacionado con conjuntos de datos o combinaciones de conjuntos de datos cuyo tamaño **(volumen),** complejidad **(variedad)** y velocidad de crecimiento **(velocidad)** dificultan su captura, gestión, procesamiento o análisis mediante tecnologías y herramientas convencionales.

Big Data maneja grandes cantidades de datos **estructurados, semiestructurados y no estructurados** que no se pueden procesar utilizando tecnologías tradicionales.

## 2.2. LAS 4 V'S DEL BIG DATA

Como hemos mencionado en el anterior apartado y cómo podemos ver en la siguiente infografía, **Big Data** da solución a los problemas de **Volumen, Velocidad, Variedad y Veracidad.**




*Fuente: IBM - Four-vs-big-data*

### IMPORTANTE

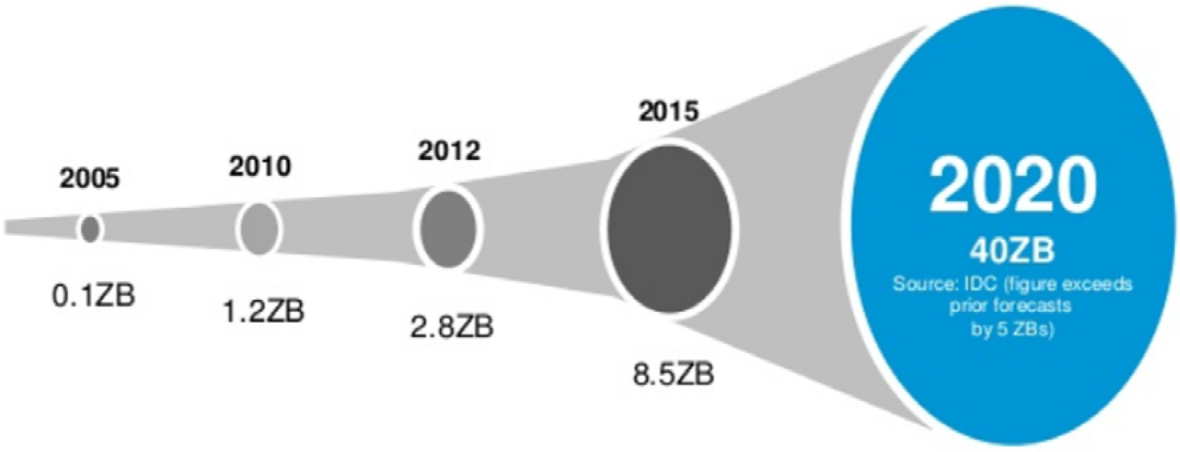
El Big Data trata de extraer información de **Valor** añadido de un gran volumen de datos. El objetivo final de las empresas es tener más información para una mejor **toma de decisione**s que deriven en mejores resultados económicos.

### 2.2.1. VOLUMEN

El **volumen** se refiere al tamaño del conjunto de datos que hay que manejar.

Dicho tamaño adquiere un papel muy relevante en la determinación del valor de los datos y, también, es un factor clave que define si podemos juzgar el fragmento como grande. Por lo tanto, el volumen justifica uno de los atributos más importantes del Big Data.

Se trata, probablemente, de la principal característica con la que todo el mundo asocia al Big Data.




Las empresas recopilan datos de variedad de fuentes, incluidas transacciones comerciales, redes sociales e información de sensores o datos de máquina a máquina.

El volumen se refiere a la gran cantidad de datos generados que deben procesarse y analizarse para tomar decisiones basadas en datos. La toma de estas decisiones se ha ido haciendo más costosas debido al aumento de fuentes de datos y dispositivos de mayor resolución.

En el pasado, almacenarlo habría sido un problema, pero las nuevas tecnologías (como Hadoop) han aliviado la carga.

### 2.2.2. VELOCIDAD

La **velocidad** se refiere, no solo a la alta frecuencia con la que se generan nuevos datos, sino también a la necesidad de responder a la información en tiempo real.

Los datos de alta velocidad se generan con tal ritmo, que requieren técnicas de procesamiento distintas (distribuidas).

Un ejemplo de datos que se generan con alta velocidad serían los mensajes de Twitter o las publicaciones de Facebook.

### 2.2.3. VARIEDAD

La **variedad** se refiere a la diversa naturaleza de la información que se tiene que manejar.

Rara vez los datos se presentan en una forma perfectamente ordenada y lista para su procesamiento. Así pues, un aspecto común en los sistemas Big Data es que los datos sean de orígenes diversos y no cumplan con una estructura ordenada, conocida y preparada para la integración en una aplicación.

Los datos digitales pueden ser generados por el ser humano o por máquinas:

- Los **datos generados por los seres humanos** son el resultado de la interacción humana con sistemas, como los servicios en línea o los dispositivos digitales.

  Ejemplos de estos datos incluyen: redes sociales, publicaciones de blogs, correos electrónicos, intercambio de fotos y mensajería.

- Los **datos generados por máquinas** provienen de programas de software y dispositivos de hardware en respuesta a eventos del mundo real.

  Ejemplos de este tipo de datos incluyen: registros web, datos de sensores, datos de telemetría y trazas de uso de dispositivos (o logs).

La variedad está relacionada con las 3 tipologías de datos que comentamos en el apartado anterior.

### 2.2.4. VERACIDAD

La **veracidad** se refiere a la calidad de los datos que se analizan.

Los datos de alta veracidad tienen muchos registros que son valiosos de analizar y que contribuyen de manera significativa a los resultados generales.

Por su parte, los datos de baja veracidad contienen un alto porcentaje de datos que no tienen sentido. Lo que no es valioso en estos conjuntos de datos se denomina ruido.

Un ejemplo de un conjunto de datos de alta veracidad serían los datos de un experimento o ensayo médico.

#### PARA SABER MÁS

Para ampliar información sobre este apartado (2. Big Data: Volumen, Velocidad, Variedad, Veracidad y Valor) te recomendamos leer el siguiente material en pdf propiedad delL3-UB:

- Módulo: Arquitectura e Infraestructura Big Data. Tema 1: Comprender el Big Data como Data Engineer (tec_mbde_aibd_t1.pdf).
- Módulo: Arquitectura e Infraestructura Big Data. Tema 2: Arquitectura Big Data (tec_mbde_aibd_t2.pdf).

# 3. BBDD RELACIONALES CON SQL (ORACLE, PL/SQL, HEIDISQL,...)

## 3.1. INTRODUCCIÓN

Las **bases de datos relacionales** son el almacén de datos más común para el almacenamiento de **datos estructurados.** Es un tipo de base de datos que cumple con el **modelo relacional.**

Un software RDBMS (Relational Database Management System) es el encargado de mantener las bases de datos relacionales.

Los conceptos más importantes de este tipo de bases de datos son:

- **Base de datos:** una base de datos se compone de varias tablas, denominadas relaciones.
- **Tablas:** conjunto de tuplas compartiendo los mismos atributos; un conjunto de filas y columnas.
- **Filas:** conjunto de datos, que representa un ítem simple. También llamado tupla o registro.
- **Columnas:** elemento etiquetado de una tupla. También llamado atributo o campo.
- **Índices:** permiten tener un acceso más rápido a los datos.
- **Constraints:** conjunto de reglas que deben cumplir los datos almacenados.
- **Triggers:** código que se ejecuta en función de alguna operación sobre la base de datos.
- **Procedimientos Almacenados:** código ejecutable que se asocia y se almacena con la base de datos.

### IMPORTANTE

Prácticamente todos los RDBMS utilizan **SQL (Structured Query Language)** para consultar y mantener la base de datos.

Además, muchas de las herramientas Big Data y de Análisis de Datos dan soporte a SQL como lenguaje para consulta de datos, por lo que es fundamental el conocimiento de este lenguaje para un Data Engineer.

Estas bases de datos se caracterizan por cumplir con un conjunto de cuatro propiedades denominado **ACID** (Iniciales de Atomicity, Consistency, Isolation, Durability). Estas propiedades se consideran esenciales para garantizar la validez de los datos durante una transacción y ayudar a garantizar la calidad de los datos.

- **Atomicity:** Varias operaciones diferentes contra la base de datos deben poder revertirse (roll back) o confirmarse (commit) al mismo tiempo, con el objetivo de garantizar la consistencia de los datos en el ámbito de una transacción.
- **Consistency:** Es la capacidad de una base de datos de garantizar la integridad de los datos mediante la validación de reglas vigentes dentro de la misma base de datos (por ejemplo, restricciones como una clave externa).
- **Isolation:** Es la capacidad de garantizar que cada transacción se produce de forma aislada entre ellas (por ejemplo, la capacidad de leer un valor de un conjunto de datos hasta que sea actualizado y confirmado por una transacción diferente).
- **Durability:** Una vez que se ha finalizado una transacción no se perderán los cambios realizados, aunque haya un fallo o un corte de energía en el sistema.

Las ventajas de los sistemas de base de datos relacionales son las siguientes:

- Capacidad para administrar la consistencia e integridad de los datos con múltiples actualizaciones simultáneas (cumplimiento ACID).
- Agrupación de datos similares en una sola tabla y la capacidad de vincular esa tabla a otras tablas para garantizar un acceso simple y fácil.
- Capacidad para escribir consultas complicadas para un análisis de datos detallado.
- Soporte para el lenguaje SQL (Structured Query Language), que proporciona un lenguaje común para acceder a los datos en todas las tecnologías.

Los inconvenientes de los sistemas de base de datos relacionales son los siguientes:

- Modelo de datos inflexible: es necesario diseñar un modelo de datos antes de usar la base de datos. Esto requiere que se comprenda exactamente cómo se deben usar los datos antes del diseño.
- Escalar una base de datos relacional es mucho más difícil y complejo que una base de datos no relacional. Llegará un momento en que la máquina no tendrá suficiente capacidad para aguantar nuestro volumen.
- La necesidad de mantener relaciones y otras limitaciones dentro de la base de datos penalizan el rendimiento.

## 3.2. TIPOS DE BASES DE DATOS RELACIONALES

Podemos clasificar los sistemas de bases de datos relacionales en dos tipos según el uso que les vayamos a dar:

- **Online Transaction Processing (OLTP):** se utilizará en aquellos casos de uso en los que se van a realizar frecuentes modificaciones en los datos, es decir, altas, modificaciones y borrados.
- **Online Analytical Processing (OLAP):** se utilizará en aquellos casos de uso donde se necesite, principalmente, rendimiento a la hora de realizar consultas pesadas, realizando agrupaciones y agregaciones.

A su vez, podemos clasificar las bases de datos relacionales en dos grupos, según la tecnología usada en su implementación. Por cada una de ellas se indican las herramientas más conocidas y utilizadas en el mercado.

### 3.2.1. BASES DE DATOS RELACIONALES TRADICIONALES

Las bases de datos relacionales estructuran su información en forma de tablas. Estas tablas están almacenadas en disco y hay índices que permiten recorrer la información de manera rápida. A partir de ahí, se puede consultar la información a través de consultas con lenguaje SQL y gracias a los optimizadores y motores de ejecución se nos facilita el dato que queremos.

Son buenas tanto para procesamiento OLAP como OLTP, sin embargo, en un entorno con grandes volúmenes de datos, muy probablemente nos encontraremos con las siguientes problemáticas:

- Queremos consultar datos que no están en el índice y, por consiguiente, hay que leer la tabla entera. Además de cargar la máquina, tardaremos mucho tiempo en obtener el dato.
- Los índices serán tan grandes que, aun siendo utilizados al máximo, ralentizarán las consultas que se hagan sobre el dato.
- Las bases de datos relacionales tradicionales no son sistemas distribuidos y, por eso, llegará un punto en que la máquina no tendrá suficiente capacidad para aguantar nuestro volumen.

### EJEMPLO

Bases de datos relacionales:

- Oracle.
- Microsoft SQL Server.
- MySQL.
- PostgreSQL.
- MariaDB.
- Amazon Aurora.

## 3.2.2. MASSIVE PARALLEL PROCESSING (MPP)

Derivan del esfuerzo de los fabricantes en hacer que algo muy similar a una base de datos relacional pueda ser escalable.

### IMPORTANTE

Las bases de datos MPP siguen teniendo tablas, pero cada una de ellas puede estar particionada en diferentes segmentos, los cuales estarán distribuidos en distintas máquinas.

El hecho de que haya estas particiones, hace que algunas funcionalidades clave, como las claves primarias o claves foráneas, no se puedan respetar. Esto impide cierto tipo de acciones que otras bases de datos relacionales permiten, como el upsert (el registro se inserta si no existe o se actualiza si existe).

### IMPORTANTE

Las bases de datos MPP no son muy buenas para el procesamiento OLTP, pero brillan para resolver los casos de uso donde exista mucha analítica o procesamiento OLAP.

El diseño en este tipo de base de datos es muy importante., por eso mismo vale la pena prestar atención a los siguientes puntos:

- Elegir correctamente qué columnas se utilizarán para particionar las tablas: si se elige una

columna con sus valores en cantidad irregular, puede producir data skew y una de las particiones ralentizará todas las acciones sobre esa tabla.

Un ejemplo podría ser una tabla con un censo electoral y particionamos por ciudad: las particiones de las capitales tendrían la mayor parte de la información y los pueblos, muy poca.

- La mayoría de las bases de datos MPP permiten elegir el formato con el que se almacenará el dato. Una buena compresión y un formato columnar siempre es una buena idea.

EJEMPLO

Bases de datos MPP:

- Greenplum.
- Amazon Redshift.
- Teradata.
- Vertica.
- Google BigQuery.
- Snowflake.

## 3.3. ORACLE

### 3.3.1. INTRODUCCIÓN

La base de datos de Oracle es una de las bases de datos relacionales más utilizadas en el ámbito empresarial.

Sus principales características son:

- **Modelo relacional:** los usuarios visualizan los datos en tablas con formato basado en filas y columnas.
- **Herramienta de administración gráfica intuitiva** y cómoda de utilizar.
- **Control de acceso:** se utilizan tecnologías avanzadas para vigilar la entrada a los datos.
- **Protección de datos:** seguridad completa en el entorno de producción y de pruebas y gestión de copias de seguridad.
- **Lenguaje PL/SQL:** permite implementar diseños "activos" que se pueden adaptar a las necesidades cambiantes de negocio.
- **Alta disponibilidad:** escalabilidad, protección y alto rendimiento para la actividad empresarial.

### 3.3.2. OPERACIONES CRUD Y LENGUAJE SQL

Oracle utiliza **SQL (Structured Query Language)** para consultar y mantener la base de datos. Podemos distinguir dos conjuntos de operaciones:

- **Data Definition Language (DDL):** definen las estructuras que almacenarán los datos, así como los procedimientos o funciones que permitan consultarlos (CREATE, ALTER, DROP).
- **Data Manipulation Language (DML):** permite a los usuarios introducir datos para posteriormente realizar tareas de consulta o modificación de los datos que contiene la base de datos (SELECT, INSERT, UPDATE y DELETE).

Al igual que otras bases de datos Oracle, soporta operaciones CRUD (Create, Read, Update y Delete).

#### 3.3.2.1. CREATE

Para realizar esta operación, en primer lugar, debemos crear la tabla dentro de la base de datos. Un ejemplo de creación de una tabla sería el siguiente:

```
CREATE TABLE people
    (name VARCHAR2(30),
     age NUMBER,
     city_of_birth VARCHAR2(30)
)
```

Y, en segundo lugar, insertamos un registro dentro de la tabla:

```
INSERT INTO people (name, age, city_of_birth)
VALUES ("Pedro", 47, "Barcelona")
```

#### 3.3.2.2. READ

Con esta operación, podemos hacer consultas sobre los datos de una tabla.

Un ejemplo de consulta de datos sobre una tabla sería el siguiente:

```
SELECT name, age, city_of_birth
  FROM people
```

#### 3.3.2.3. UPDATE

Con esta operación, podemos hacer actualizaciones sobre los registros de una tabla.

Un ejemplo de actualización de datos sobre una tabla sería el siguiente:

```
UPDATE people
   SET age = age + 3
 WHERE city_of_birth = "Barcelona"
```

### 3.3.2.4. DELETE

Con esta operación podemos eliminar documentos de una tabla.

Un ejemplo de borrado de datos sobre una tabla sería el siguiente:

```
DELETE FROM people
 WHERE city_of_birth = "Barcelona"
```

#### PARA SABER MÁS

Para tener conocimiento de SQL, existen multitud de fuentes de información. Te recomendamos las siguientes:

- En este enlace Conceptos Básicos de Oracle, encontrarás:
  - Cómo hacer consultas (SELECT), ordenaciones (ORDER BY), filtros (WHERE, AND, OR), joins (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN) y agrupaciones (GROUP BY, HAVING).
  - Claves primarias y foráneas.
- Lee las entradas del siguiente blog de Oracle. SQL-101.
- Para ver todos los detalles del lenguaje, consulta el manual de referencia de SQL de Oracle. Oracle Database SQL Language Reference.

### 3.3.3. PL/SQL

PL/SQL es una combinación de SQL y un lenguaje de programación.

Fue desarrollado por Oracle para mejorar las capacidades de SQL.

PL/SQL es uno de los tres lenguajes de programación clave integrados en la base de datos de Oracle, junto con el propio SQL y Java.

PL/SQL tiene las siguientes características:

- Está estrechamente integrado con SQL.
- Ofrece una amplia comprobación de errores.

- Ofrece numerosos tipos de datos.
- Ofrece una variedad de estructuras de programación.
- Soporta programación estructurada a través de funciones y procedimientos.
- Es compatible con la programación orientada a objetos.

Además, ofrece las siguientes ventajas:

- Admite SQL estático y SQL dinámico:
  - El SQL estático admite operaciones DML y control de transacciones dentro de los llamados bloques PL/SQL.
  - El SQL dinámico, permite incrustar declaraciones DDL en bloques PL/SQL.
- Permite enviar un bloque completo de operaciones a la base de datos a la vez, lo que reduce el tráfico de red y proporciona un alto rendimiento para las aplicaciones.
- Brinda una alta productividad a los programadores, ya que pueden consultar, transformar y actualizar datos dentro del motor de base de datos.
- Ahorra tiempo en el diseño y la depuración gracias a características tales como el manejo de excepciones, la encapsulación, la ocultación de datos y los tipos de datos orientados a objetos.
- Las aplicaciones escritas en PL/SQL son totalmente portátiles.
- Proporciona un alto nivel de seguridad.
- Proporciona acceso a paquetes SQL predefinidos.

### PARA SABER MÁS

Para tener conocimiento de PL/SQL existen multitud de fuentes de información, pero te recomendamos las siguientes:

- Leer las entradas del siguiente blog de Oracle: PL/SQL-101.
- Para ver todos los detalles del lenguaje, consulta el manual de referencia de PL/SQL de Oracle. Oracle Database PL/SQL Language Reference.

## 3.3.4. HERRAMIENTAS DE DESARROLLO

Existen varios entornos de desarrollo para escribir sentencias SQL y PL/SQL, siendo Oracle SQL Developer y Oracle SQL*Plus las herramientas más utilizadas.

- **Oracle SQL Developer:** se trata de una interfaz gráfica para SQL y PL/SQL. Proporciona funciones para ver los componentes de la base de datos y actualizar los valores sin escribir ninguna consulta SQL.
- **Oracle SQL*Plus:** se trata de una interfaz por la línea de comandos.

# 4. BBDD NO RELACIONALES (NOSQL CON MONGODB)

## 4.1. INTRODUCCIÓN

En primer lugar, veamos una definición de los que entendemos por bases de datos no relacionales o NoSQL:

*"Sistemas de gestión de datos que difieren del modelo relacional (no usan SQL en sus consultas). Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad). Los sistemas NoSQL se denominan a veces "no sólo SQL" para subrayar el hecho de que también pueden soportar lenguajes de consulta de tipo SQL."*

Fuente: Wikipedia

Las nuevas características que nos van a proporcionar las bases de datos NoSQL respecto a las bases de datos relacionales son:

- **Modelo de datos:** una base de datos NoSQL permite crear una aplicación sin tener que definir el esquema por adelantado. No tener un esquema predefinido hace que las bases de datos NOSQL sean mucho más fáciles de actualizar a medida que los datos y requisitos funcionales vayan cambiando.
- **Estructura de datos:** Las bases de datos NoSQL están diseñadas para manejar datos semiestructurados y no estructurados (por ejemplo: textos, publicaciones en redes sociales, videos, correos electrónicos) que constituyen gran parte de los datos que existen en la actualidad.
- **Escalabilidad:** Resulta barato escalar una base de datos NoSQL porque podemos agregar capacidad escalando horizontalmente, añadiendo nuevos nodos con servidores básicos. Esta capacidad tiene el inconveniente de no poder tener un cumplimiento completo de ACID.
- **Modelo de desarrollo:** Las bases de datos NoSQL, generalmente, se pueden descargar e instalar de forma gratuita (se paga por soporte). Con NoSQL, podemos comenzar un proyecto sin grandes inversiones en licencias de software.

Cuando estemos en la fase de diseño de la arquitectura de nuestra solución, debemos tener en cuenta los siguientes puntos a la hora de decidir si elegimos una base de datos NoSQL para el almacenamiento de nuestros datos:

- **Modelado:** cuando el modelado "a priori" no está determinado y hay que cambiarlo con frecuencia.
- **Business Case:** cuando se utilizan los datos para un propósito específico. Hay que tener en cuenta el coste de licencias y el mantenimiento de la base de datos.
- **Time To Market:** cuando necesitemos que nuestra solución tenga un despliegue en producción lo más rápido posible. Al no tener esquema predefinido, no es necesario emplear tiempo en

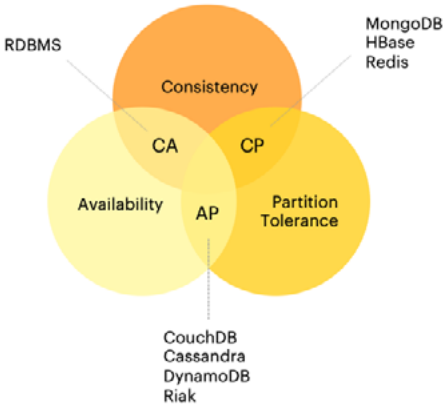
cambiar el esquema.

- **Rendimiento vs consistencia de datos:** cuando podemos aceptar un retraso en la propagación de la información a cambio de un mejor rendimiento.

El teorema CAP (también llamado teorema de Brewer en honor al científico informático Eric Brewer) establece:

"Un sistema informático distribuido no puede garantizar las tres propiedades siguientes al mismo tiempo: Consistency, Availability, Partition Tolerance".

Es el teorema en el que se basa la teoría de las bases de datos.




**Consistency:** cada lectura recibe la escritura más reciente o un error.

**Availability:** cada solicitud recibe una respuesta (sin error), sin garantía de que contenga la escritura más reciente.

**Partition Tolerance:** el sistema continúa funcionando a pesar de que la red descarte (o retrase) una cantidad arbitraria de mensajes entre los nodos.

De esta forma, tenemos las siguientes combinaciones que puede cumplir una base de datos:

- **CA:** sitio único.
- **CP:** es posible que no se pueda acceder a los datos, pero cuando se utilizan son coherentes.
- **AP:** los datos siempre se pueden utilizar, pero algunos pueden no ser consistentes (tiempo de propagación). Es lo que normalmente se llaman **bases de datos eventualmente consistentes.**

### PARA SABER MÁS

Antes de proceder a utilizar una base de datos NoSQL distribuida, debemos entender qué papel juega en el teorema CAP.

## 4.2. TIPOS DE BASES DE DATOS NOSQL

Tenemos cuatro tipos de bases de datos NoSQL. Por cada una de ellas, se indican las más conocidas y utilizadas en el mercado.

- **Almacenamiento Columnar:** almacenan los datos de las tablas como columnas de datos en lugar de filas de datos. Para una misma tabla, puede haber filas con columnas distintas. Esto provoca una ventaja de rendimiento en la búsqueda de los datos porque el sistema no escanea

todas las filas, sino que sólo lee las columnas de las que necesitamos extraer la información, reduciendo drásticamente el tiempo de respuesta de las consultas.

Se utilizan en lugar de la base de datos relacionales orientada a filas, en sistemas donde la velocidad de extracción de las consultas son críticas:

- Apache Cassandra.
- Apache HBase.

- **Clave-Valor:** para cada clave única hay un valor de atributo. Tienen una estructura para guardar información basada, únicamente, en la clave (como si fuera una tabla Hash), lo que hace que el tiempo de respuesta al leer datos sea muy rápido:
  - Redis.
  - Oracle Berkeley DB.

- Orientadas a Documentos: los datos se guardan como documentos (normalmente como JSON). Al ser una base de datos sin esquema, diferentes documentos dentro de la misma tabla podrían tener diferentes esquemas:
  - MongoDB.
  - Couchbase.

- Grafos: la información se estructura en grafos, vértices y aristas. Por lo general, proporcionan un lenguaje para poder navegar dentro de sus conexiones:
  - Neo4j.

## 4.3. MONGODB

### 4.3.1. INTRODUCCIÓN

**MongoDB** es una base de datos NoSQL con las siguientes características principales:

- Base de datos orientada a documentos.
- Open Source (AGPL): https://github.com/mongodb/mongo.
- Proporciona alto rendimiento, alta disponibilidad y escalado horizontal automático.
- Los registros se representan como documentos en formato JSON y son almacenados como documentos en formato BSON, variante de JSON.
- Además de los tipos básicos, el valor de los documentos puede incluir documentos o arrays de documentos.

- Es la tecnología NoSQL más utilizada en la actualidad en el mercado.
- Tiene muy buena documentación.

**JSON (JavaScript Object Notation)** es un formato ligero de intercambio de datos. Es fácil de leer y escribir para los humanos y es fácil de analizar y generar para las máquinas. Se basa en un subconjunto del JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999.

Los documentos JSON se crean a partir de un pequeño conjunto de construcciones muy simples:

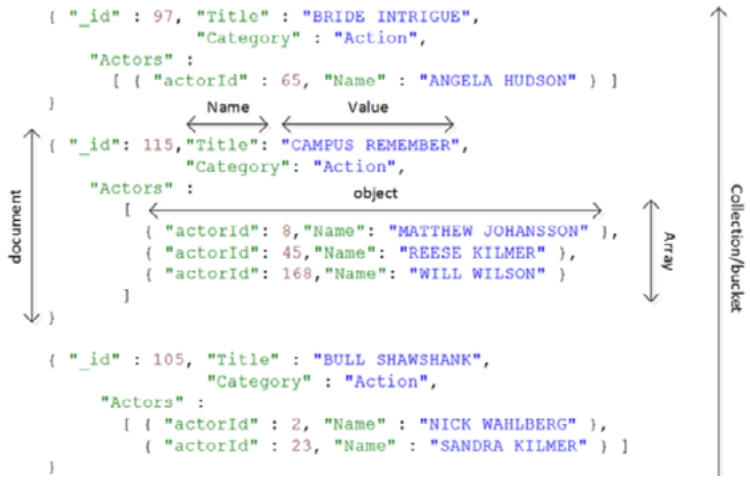
- **Valores:** pueden ser cadenas Unicode, números en formato estándar, booleanos, Arrays u Objetos. Dado que los valores pueden incluir Objetos o Arrays, que a su vez contienen valores, una estructura JSON puede representar un conjunto de información anidado y arbitrariamente complejo. En particular, las Arrays se pueden utilizar para representar grupos repetidos de documentos, que en una base de datos relacional requerirían una tabla separada.
- **Objetos:** consisten en uno o más pares de nombre-valor en el formato "nombre": "valor", entre llaves ("{" y "}") y separados por comas (",").
- **Arrays:** consisten en listas de valores entre corchetes ("["y"]") y separados por comas (",").

MongoDB soporta muchos tipos de datos. Lo más utilizados son:

- **String:** es el tipo de datos más utilizado para almacenar los datos. Deben seguir la codificación UTF-8.
- **Integer:** este tipo se utiliza para almacenar un valor numérico. El número entero puede ser de 32 bits o 64 bits dependiendo del servidor.
- **Boolean:** este tipo se utiliza para almacenar un valor verdadero o falso.
- **Double:** este tipo se utiliza para almacenar números con decimales.
- **Date:** este tipo de datos se utiliza para almacenar fechas.

## PARA SABER MÁS

Tipos de datos en MongoDB.



Son estructuras de datos universales. Prácticamente, todos los lenguajes de programación modernos los admiten de una forma u otra. Tiene sentido que un formato de datos que sea intercambiable con lenguajes de programación también se base en estas estructuras.

El soporte JSON se ha introducido en casi todos los sistemas de bases de datos.

## SABÍAS QUE...

Existen multitud de herramientas para formatear y verificar si un documento JSON está bien construido.

Por ejemplo, hay utilidades on-line tales como JSON Formatter & Validator.

Los componentes son muy similares a sistemas relacionales. Arquitectura basada en cliente y servidor:

- **Mongod:** nombre del proceso de la instancia de la base de datos en el servidor.
- **Mongos:** proceso controlador del sharding (router).
- **Mongo:** cliente utilizado para conectarnos al servidor.

En la siguiente tabla se muestran la correspondencia entre los principales conceptos de una base de datos tradicional y MongoDB.

| Base de datos SQL | MongoDB |
|---|---|
| Base de datos | Base de datos |
| Tabla | Colección |
| Fila | Documento JSON |
| Columna | Campo |
| Índice | Índice |

## PARA SABER MÁS

En el siguiente enlace de la documentación de MongoDB, se hace un mapeo de todos los conceptos:

SQL to MongoDB Mapping Chart

## IMPORTANTE

Las bases de datos NoSQL, y en concreto MongoDB, renuncian a algunas características con el objetivo de mantener la escalabilidad. No disponemos de transacciones sobre varias colecciones ni podemos hacer joins.

### 4.3.2. OPERACIONES CRUD

Al igual que otras bases de datos, MongoDB soporta operaciones CRUD (Create, Read, Update y Delete).

#### 4.3.2.1. CREATE

Con esta operación, creamos un nuevo documento dentro de una colección perteneciente a una base de datos. La sintaxis es la siguiente:

```
use test
db.people.insertOne( {
    name: "Pedro",
    age: 47,
    city_of_birth: "Barcelona"
 } )
```

Con el anterior comando habremos insertado un documento dentro de la colección “people” de la base de datos “test”.

Si queremos insertar varios documentos a la vez, utilizaremos el comando:

```
db.people.insertMany
```

**IMPORTANTE**

No es necesario que existan, previamente, la base de datos ni la colección. Si MongoDB detecta que no existen, las crea. Todo documento en MongoDB tiene asociada una Primary Key, identificada con un campo cuyo nombre es “_id” que, si no se especifica, se genera automáticamente.

#### 4.3.2.2. READ

Con esta operación podemos hacer consultas sobre los documentos de una colección. Su sintaxis es la siguiente:

```
use test
db.people.find(
    { city_of_birth: "Barcelona" },
    { name: 1, age: 1, _id: 0 }
)
```

A este comando, se le pasan como parámetros dos objetos:

- Con el primero, indicamos el **filtro que queremos aplicar a la búsqueda** (sentencia where de una base de datos tradicional).

- Con el segundo, indicamos que **campos nos queremos traer.**

### IMPORTANTE

Esta operación siempre va a traer el campo "_id" asociado a cada documento, salvo que sea excluido en los campos a recuperar (_id: 0).

#### 4.3.2.3. UPDATE

Con esta operación, podemos hacer actualizaciones sobre los documentos de una colección. Su sintaxis es la siguiente:

```
use test
db.people.updateMany(
   { city_of_birth: "Barcelona" } ,
   { $inc: { age: 3 } }
)
```

A este comando se le pasan como parámetros dos objetos:

- Con el primero, indicamos el **filtro que queremos aplicar a la actualización** (sentencia where de una base de datos tradicional).
- Con el segundo, indicamos los **campos que queremos actualizar.** En este caso, incrementamos en 3 el campo "age".

#### 4.3.2.4. DELETE

Con esta operación podemos eliminar documentos de una colección. La sintaxis es la siguiente:

```
db.people.deleteMany( { city_of_birth: "Barcelona" } )
```

A este comando se le pasa como parámetro un objeto, el filtro que queremos aplicar al borrado de documentos.

### PARA SABER MÁS

Consulta la siguiente documentación oficial:

MongoDB CRUD Operations.

### 4.3.3. MODELADO DE LOS DATOS

Al igual que en las bases de datos relacionales hacemos un modelado de los datos basado en tablas y sus relaciones, en las bases de datos NoSQL, también, debemos pensar cómo vamos a modelar nuestras entidades.

Por ejemplo, en una base de datos relacional tenemos un modelo de datos con 3 tablas:

**Tabla 1:**

| Id | Name | Level | City |
|---|---|---|---|
| 123 | Pedro | 7 | Barcelona |

**Tabla 2:**

| Level | Description |
|---|---|
| 7 | Gerente |

**Tabla 3:**

| Id | Contact |
|---|---|
| 123 | +34 620 12 34 56 |
| 123 | a@acme.com |

En MongoDB lo podemos modelar como un solo documento:

```
{
  "_id": 123,
  "name": "Pedro",
  "level": 7,
  "level_desc": "Gerente",
  "city": "Barcelona",
  "contact": [
    "+34 620 12 34 56",
    "a@acme.com"
  ]
}
```

#### IMPORTANTE

Aunque estas bases de datos no tengan un esquema predefinido, es importante que exista un Gobierno del Dato y unos Procesos de Calidad.

Los nombres y los tipos de los campos deben estar sujetos a gobernanza y convención de nomenclatura.

Se deben pasar regularmente procesos de calidad para verificar que no se están insertando en las colecciones documentos con una estructura no deseada.

## PIENSA UN MINUTO

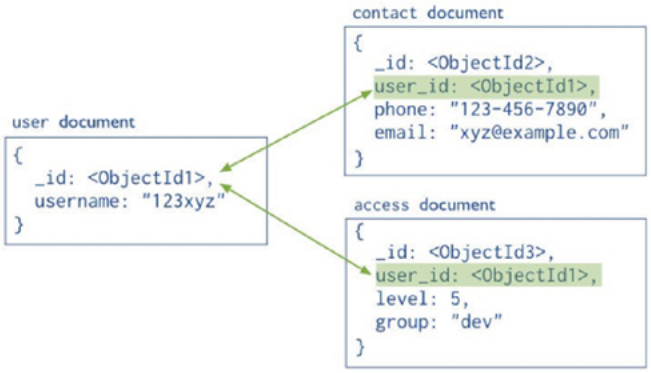
Entonces... ¿Puedo modelar toda mi información en una sola colección?

Una de las decisiones más importantes a la hora de modelar consiste decidir si embebemos documentos (Modelo Embebido) o referenciamos documentos (Modelo Referencial).

En las siguientes imágenes podemos ver un ejemplo de cada uno de ellos:

**Modelo Embebido**


**Modelo Referencial**




## IMPORTANTE

Existe un límite de 16 Mb en los documentos.

### 4.3.3.1. RELACIONES ONE-TO-ONE

Por norma general, en las relaciones one-to-one se opta por el modelo embebido.

¿En qué casos debo utilizar modelo referencial en one-to-one? Únicamente cuando el documento embebido tenga un tamaño excesivamente grande y pueda ocurrir que se supere el tamaño límite del documento (16 Mb).

### 4.3.3.2. RELACIONES ONE-TO-MANY

En relaciones one-to-many, el tipo de modelado depende del contexto de los datos (no existe un estándar como en SQL).

En este tipo de relaciones, se definen los conceptos de “padre” e “hijo”, que también existen en los modelos relacionales.

Por ejemplo, podemos tener una entidad que representa una compra en una tienda on-line y otra entidad que representa los distintos productos que hemos comprado. A la primera entidad la llamaríamos entidad “padre” y a la segunda entidad la llamaríamos “hija”, porque tiene una dependencia (identificados del pedido, clave foránea en terminología SQL) con la primera.

Si los datos embebidos no tienen sentido por si mismos y siempre son accedidos mediante el documento “padre”, entonces debemos optar por el modelo embebido.

Cuando los datos a “embeber” tienen sentido por sí solos, es recomendable modelarlos en una colección diferente.

Cuando usemos referencias en nuestros documentos, también debemos decidir en qué lado debemos almacenar dichas referencias.

- El crecimiento de las relaciones determina en qué lado guardar la referencia:

  1. Si el número de libros por publicador es pequeño con crecimiento limitado, guardar la referencia del libro dentro del documento del publicador podría ser útil.

  2. Si el número de libros por publicador es ilimitado, es aconsejable guardar la referencia del publicador dentro del documento del libro (para evitar arrays con demasiados elementos).

- También, depende del tipo de query que vayamos a realizar:

**Opción 1**

```
{
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA",
    books: [12346789, 234567890, ...]
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf"
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB Develo
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English"
}
```

**Opción 2**

```
{{
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
}
}

{
{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}}

{{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB Develop
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English",
    publisher_id: "oreilly"
}}
```

#### 4.3.3.3. REGLAS DEL MODELADO

A continuación, se detallan las 6 reglas de oro que debemos seguir de cara a realizar un buen modelado de datos:

1. Utilizar siempre documentos embebidos a no ser que exista una razón de peso para no utilizarlos.
2. Si un objeto necesita ser accedido por sí mismo, es una razón de peso para no estar embebido.
3. No usar referencias en arrays cuando la longitud de este puede elevarse demasiado. En general, una alta cardinalidad en arrays es mala idea.
4. No tener miedo de los JOINs a nivel de aplicación. Si los índices de la colección son correctos, los JOINs a nivel de aplicación son solo un poco más caros que los JOINS en una base de datos relacional.
5. Si tenemos documentos embebidos que son constantemente actualizados (podría ser el caso del publisher en el ejemplo anterior), entonces serán candidatos para estar referenciados, ya que se perdería demasiado tiempo buscando todas las referencias de ese objeto actualizando una a una.
6. Estructurar los datos según el tipo de querys y actualizaciones que se vayan a realizar.

PARA SABER MÁS

Consultar la siguiente documentación oficial:

Data Models.

### 4.3.4. RENDIMIENTO

En MongoDB también existe el concepto de índice. Los índices, al igual que en las bases de datos relacionales, mejoran el rendimiento de las consultas:

- Son estructuras de datos que permiten tener una colección pre-ordenada según los parámetros indicados.
- Mejoran la velocidad de las consultas, pero añaden una sobrecarga en las escrituras (para cada escritura, se deben tener en cuenta los índices existentes).
- Los índices pueden crearse ascendente o descendentemente.
- Todos los índices pueden ser recorridos en sentido contrario.
- En general, el uso correcto de índices disminuye notablemente el número de documentos que MongoDB tiene que escanear en una consulta.

A continuación, detallaremos los tipos de índices más utilizados.

### 4.3.4.1. ÍNDICES SIMPLES

Estos índices tienen las siguientes características:

- Especifica un índice en el campo indicado.
- Por defecto, todas las colecciones tienen un índice por el campo "_id".
- Permite crear índices en campos embebidos mediante notación de puntos.

EJEMPLO

```
db.people.createIndex( { "age": 1 } )
```

### 4.3.4.2. ÍNDICES COMPUESTOS

Estos índices tienen las siguientes características:

- Similar al caso anterior, pero con más de un campo.
- Existe un límite de 31 campos para el índice compuesto.
- El orden de los campos en los índices compuestos es muy importante. Al igual que en las bases de datos relacionales, si, por ejemplo, definimos un índice con 3 campos, hay que proporcionarlos en las consultas, empezando por los campos que están a la izquierda.

EJEMPLO

```
db.users.createIndex({"name": 1, "age": -1})
```

### 4.3.4.3. ÍNDICES MULTIKEY

Estos índices tienen las siguientes características:

- Son utilizados para crear índices en aquellos campos que sean arrays.
- Es similar el índice de tipo "campo" pero con arrays.
- No es necesario especificar "multikey" en la creación del índice. MongoDB detecta si alguno de los campos del índice es un array.

- Útil para arrays que contengan tanto valores simples (strings, enteros) como documentos anidados.
- Solo uno de los campos indexados puede ser del tipo array.

EJEMPLO

```
db.users.createIndex({"name": 1, "age": -1})
```

#### 4.3.4.4. ÍNDICES DE TEXTO

Estos índices tienen las siguientes características:

- Los índices textuales permiten realizar consultas y búsquedas de texto en cualquier campo o array de tipo textual.
- Cuando creamos el índice, necesitamos especificar cuál es el campo a indexar.
- Ocupa muchos recursos. Tiene mucha complejidad.
- Existe la posibilidad de indexar todos los índices que contengan strings mediante el operador "$**".

PARA SABER MÁS

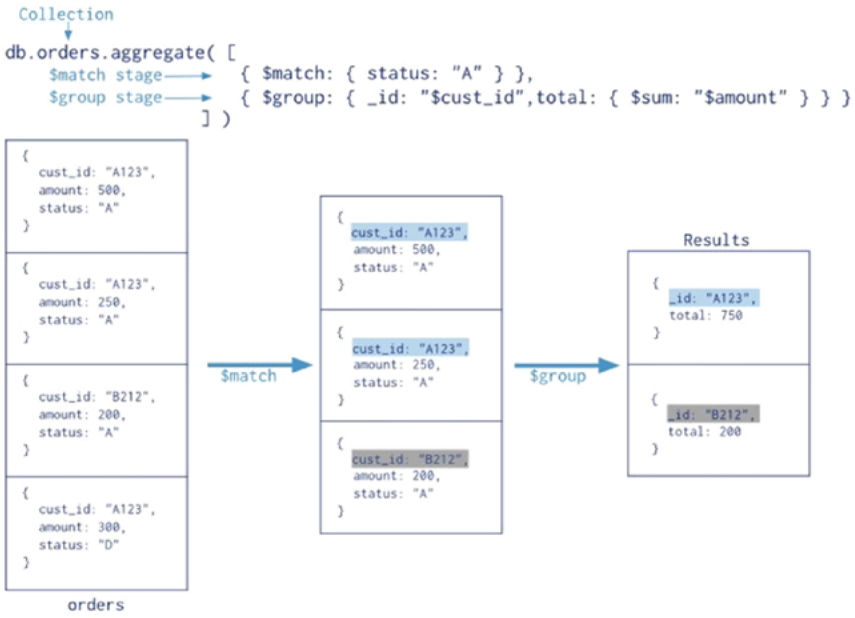
Para tener más detalle sobre los índices en MongoDB, consulta la siguiente documentación oficial:

MongoDB Indexes.

### 4.3.5. FRAMEWORK DE AGREGACIÓN

El proceso de agregación se define como una serie de operaciones a las cuales se somete una colección para obtener un conjunto de resultados calculados, formateados y/o filtrados de manera diferente a como se encuentran en los documentos, cuyo objetivo, en general, es agrupar y/o calcular datos que residen en los documentos de acuerdo con una necesidad particular.

Consiste en hacer pasar a una colección por un conjunto de etapas/estados.




El proceso de agregación tiene operadores para definir las **etapas,** para usar **expresiones** (and/or, comparaciones de campos, manejos de strings, ...) y para, finalmente, realizar las **acumulaciones.**

### 4.3.5.1. ETAPAS

- **$project:** añade o elimina campos al resultado.
- **$match:** filtra el número de documentos en base a una condición.
- **$redact:** reduce el contenido de los documentos en base a los propios valores del documento.
- **$limit:** limita el número de resultados.
- **$skip:** salta los N primeros valores.
- **$unwind:** desdobla un array creando un registro por cada elemento del array.
- **$group:** agrupa documentos de entrada dado un identificador determinado. Puede agruparse por más de un campo.
- **$sort:** ordena los documentos en base a la clave indicada.
- **$geoNear:** retorna un conjunto de documentos ordenados basados en la proximidad de un punto geoespacial. Incorpora la funcionalidad de $match, $sort y $limit.
- **$out:** escribe el conjunto de documentos resultantes en una colección. Debe de ser la última etapa.

### 4.3.5.2. EXPRESIONES

- **Booleanos:** $and, $or, $not.
- **Set:** $setEquals, $setIntersection, $setUnion, $setDifference, $setIsSubset, $anyElementTrue, $allElementsTrue.

- **Comparadores:** $eq, $gt, $gte, $lt, $lte, $ne.
- **Aritméticos:** $add, $subtract, $multiply, $divide, $mod.
- **Strings:** $concat, $substr, $toLower, $toUpper, $strcasecmp.
- **Texto:** $meta (acceso a meta-data).
- **Array:** $size.
- **Variable:** $map, $let.
- **Literal:** $literal.
- **Date:** $dayOfYear, $dayOfMonth, $dayOfWeek, $year, $month, $week, $hour, $minute, $second, millisecond.
- **Condicionales:** $cond, $ifNull.

#### 4.3.5.3. ACUMULADORES

- **$sum:** calcula la suma de los elementos indicados.
- **$avg:** calcula la media de los elementos indicados.
- **$first:** coge el primer elemento para cada dato agregado.
- **$last:** coge el último elemento para cada dato agregado.
- **$max:** devuelve el máximo valor del conjunto de datos agregados.
- **$min:** devuelve el mínimo valor del conjunto de datos agregados.
- **$push:** retorna un array que contiene todas las expresiones indicadas.
- **$addToSet:** retorna un array de valores únicos que contenga las expresiones indicadas.

#### PARA SABER MÁS

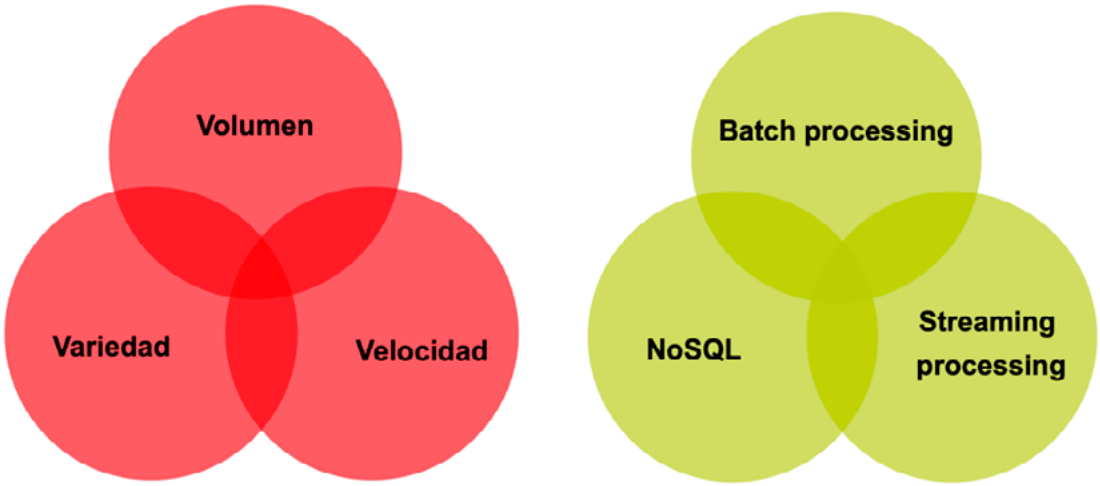
Para obtener más detalle sobre el framework de agregación, consulta la siguiente documentación oficial:

Aggregation.

# 5. PROCESAMIENTO DEL DATO: HADOOP, SPARK Y MAP REDUCE

## 5.1. INTRODUCCIÓN

Con el objetivo de dar solución a los problemas relacionados con el volumen, la velocidad y la variedad, surgió la necesidad de desarrollar nuevas arquitecturas y herramientas.




Para los problemas de **volumen y velocidad,** tenemos, en la actualidad, un amplio conjunto de **herramientas de computación distribuida.**

El otro problema, la **variedad,** se resuelve con las tecnologías de almacenamiento de información **NoSQL** que hemos visto en el anterior apartado.

Estas herramientas de computación distribuida se pueden clasificar de acuerdo con tres paradigmas de computación.

- Procesamiento Batch.
- Procesamiento Streaming.
- Procesamiento Híbrido.

Un factor que diferencia los procesamientos batch y streaming es la **latencia.** La latencia es el tiempo que transcurre entre que hacemos una consulta de información y obtenemos la respuesta.

### IMPORTANTE

Prácticamente, todas las herramientas que se utilizan en Big Data se pueden descargar y usar libremente a efectos de aprendizaje.

Para uso en empresas, existen fabricantes que ofrecen todas estas herramientas empaquetadas en un solo productos y ofrecen un soporte empresarial.

El principal fabricante en la actualidad es:

Cloudera.

## 5.1.1. PROCESAMIENTO BATCH

El procesamiento batch o por lotes da solución al problema del volumen. Sus principales características son:

- Alta latencia.
- Escalable.
- Manejo de grandes cantidades de información estática.
- Distribuido.
- Ejecución en paralelo.
- Tolerante a fallos.

### EJEMPLO

Hoy en día, los sistemas distribuidos de computación batch más populares son:

- Hadoop MapReduce.
- Apache Spark.
- Apache Hive.
- Apache Pig.
- Apache TEZ.
- Cascading.

## 5.1.2. PROCESAMIENTO STREAMING

El procesamiento streaming da solución al problema de la **velocidad.** Sus principales características son:

- Baja latencia.
- Información generada de continuo (Streams).
- Distribuido.
- Ejecución en paralelo.
- Tolerante a fallos.

### EJEMPLO

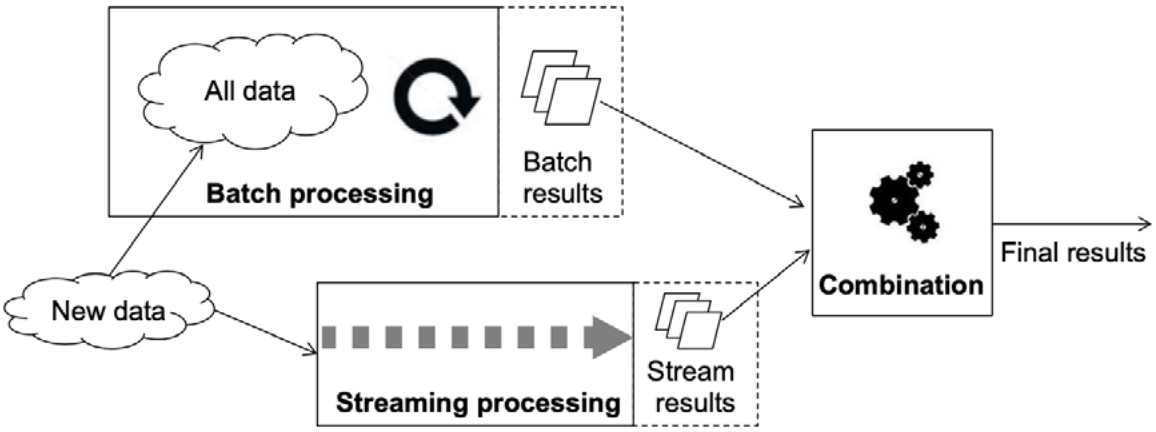
Hoy en día, los sistemas distribuidos de computación en streaming más populares son:

- Apache Spark Streaming.
- Apache Storm.
- Kafka Streams.

## 5.1.3. PROCESAMIENTO HÍBRIDO

El procesamiento híbrido surgió de la necesidad de dar solución a ambos problemas a la vez, **volumen y velocidad.** Sus principales características son:

- Baja latencia.
- Datos en estático + Datos en movimiento.
- Escalable.
- Combina resultados batch y streaming.




La **Arquitectura Lambda** es la solución más utilizada para resolver esta necesidad. Se define en 2012, cuando las tecnologías de procesamiento en streaming (Storm) no eran tan potentes como las de procesamiento en batch (Hadoop).

En 2014 las tecnologías de procesamiento streaming (Storm, Kafka, Samza) están más maduras y equiparables en potencia a las de procesamiento batch.

Jay Kreps y Martin Kleppmann (LinkedIn) cuestionan la necesidad de construir una arquitectura tan compleja teniendo en cuenta las tecnologías actuales (2014).

Describen cómo se puede desarrollar un sistema híbrido utilizando simplemente tecnología de procesamiento streaming y lo denominan Arquitectura Kappa.

### PARA SABER MÁS

Visita el siguiente enlace:

https://www.oreilly.com/radar/questioning-the-lambda-architecture/

### EJEMPLO

Hoy en día, los sistemas distribuidos de computación híbrida más populares son los que puedes ver en este enlace:

Apache Flink.

## 5.2. HADOOP

Apache Hadoop es una de las plataformas líderes para el procesamiento y almacenamiento distribuido de datos.

Hadoop no es un tipo de base de datos, sino un ecosistema de software que permite la computación en paralelo. Es un habilitador de ciertos tipos de bases de datos distribuidas NoSQL (como HBase) que permite que los datos se distribuyan en miles de servidores.

Las componentes de software que forman el ecosistema de Apache Hadoop permiten el procesamiento distribuido de grandes conjuntos de datos en grupos de servidores utilizando un modelo de programación simple. Hadoop escala linealmente, multiplicando la cantidad de servidores en el clúster se consigue reducir el tiempo de cálculo proporcionalmente.

Hadoop nos permite abstraernos de la complejidad de los sistemas distribuidos para diseñar soluciones escalables horizontalmente y se basa en dos componentes principales:

- HDFS (Hadoop Distributed File System).
- MapReduce.

### PARA SABER MÁS

Fue en 2003 y 2004 cuando Google publicó dos artículos que sentaron las bases de lo que es Hadoop:

- The Google File System.
- MapReduce: Simplified Data Processing on Large Clusters.

## 5.2.1. HDFS

Es un sistema de archivos distribuido y altamente tolerante a fallos que se utiliza para respaldar los cálculos derivados del procesamiento de los datos. Está diseñado para implementarse en hardware de bajo coste.

Es similar a un sistema de ficheros Unix, salvo que los datos se encuentran distribuidos en varias máquinas. Está diseñado para aplicaciones que no son en tiempo real y demandan un gran rendimiento (ancho de banda) en lugar de acceso de baja latencia.

Es óptimo para ficheros de gran tamaño y para aprovechar la localidad de los datos en combinación con un framework de procesamiento (MapReduce, Spark, etc.).

No se recomienda su uso para aplicaciones con accesos aleatorios y de baja latencia.

### IMPORTANTE

HDFS almacena la información en bloques de 128 MB. Por eso mismo, no es aconsejable utilizar ese sistema de ficheros para almacenar gran cantidad de ficheros pequeños.

### PARA SABER MÁS

Visita el siguiente enlace:

HDFS Architecture.

## 5.2.2. HADOOP MAPREDUCE

Es un paradigma de programación en paralelo, distribuido y tolerante a fallos.

Se trata de un modelo computacional que, básicamente, toma procesos que necesitan cálculos de datos intensivos y distribuye dichos cálculos a través de un número, potencialmente infinito, de servidores (generalmente conocido como un clúster Hadoop).

Ha sido la tecnología que solucionó los problemas de gran volumen de datos y tiene las siguientes características:

- Procesa grandes cantidades de datos de forma distribuida.
- El trabajo se divide en varias tareas y cada tarea se ejecuta de forma independiente en un nodo del clúster.
- Abstrae al programador de todo lo que conlleva la computación distribuida como la comunicación y la coordinación entre máquinas, la recuperación ante los fallos hardware o los informes de estado de cada nodo.
- La entrada se mapea en una colección de pares <clave, valor> y luego se reducen (agregan) los pares con la misma clave.

**Fase Map:**

- Múltiples tareas "Map" son ejecutadas en paralelo.
- Cada tarea "Map" procesa un trozo de la entrada.
- El framework mueve la computación a donde se encuentran los datos, lo que reduce el tráfico de red y el tiempo necesario para transmitir los datos de una máquina a otra.
- Cada llamada a la función "Map" recibe un par <clave, valor> y emite una lista (cero o más) pares <clave, valor>.
- La salida de las tareas "Map" se almacenan localmente en disco y no en el HDFS.

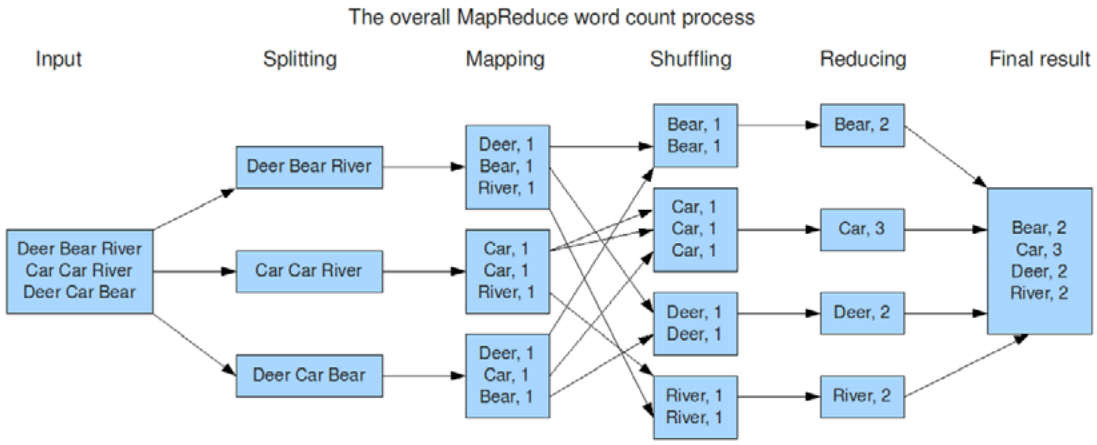
**Fase Sort and Shuffle:**

- El particionado determina a qué tarea "Reduce" debe ir cada par <clave, valor>.
- Los datos son enviados a las tareas "Reduce".
- La ordenación ahorra tiempo a la fase "Reduce", ayudando a distinguir cuándo una nueva llamada a la función "Reduce" es necesaria.
- Los datos son agrupados por clave. Se llamará a la función "Reduce" una vez por cada clave.
- El particionado se realiza en las tareas "Map", mientras que el ordenamiento y la agrupación son realizadas por las tareas "Reduce".

**Fase Reduce:**

- Múltiples tareas "Reduce" se ejecutan en paralelo distribuidas en los nodos del clúster.
- La función "Reduce" recibe como entrada una clave y el conjunto de valores asociados a la misma clave.
- Es llamada exactamente una vez por cada clave.
- Escribe cero o más pares <clave, valor>.
- La salida de la fase "Reduce" se escribe al HDFS.

En la siguiente figura se puede ver un ejemplo donde se hace un MapReduce para realizar un "Word-Count".




## PARA SABER MÁS

Visita el siguiente enlace:

MapReduce Tutorial.

## IMPORTANTE

La codificación de un proceso que utilice MapReduce no es precisamente sencilla. Por este motivo, han surgido otros frameworks que facilitan al desarrollador la tarea de crear dichos procesos, tales como **Spark.**

**Apache Hive** permite ejecutar SQL contra datos almacenados en HDFS. Es una capa de abstracción que traduce las consultas SQL en procesos MapReduce.

## 5.3. SPARK

Como hemos visto en el anterior apartado MapReduce simplificó el análisis de datos en clusters de gran tamaño, pero no satisface mucha de las necesidades que fueron surgiendo:

- Aplicaciones más complejas e iterativas (Ej. Algoritmos de Machine Learning).
- Consultas más interactivas.
- Procesado de datos en tiempo real.

Para dar solución a estas nuevas necesidades, nació, en 2009 en el departamento AMPLab de la Universidad de Berkeley, una nueva herramienta de computación distribuida llamada Spark.

**Spark** realiza los **cálculos en memoria** para aumentar la velocidad del procesamiento de los datos respecto a MapReduce. Se suele integrar dentro de un clúster Hadoop, puede acceder a HDFS y puede procesar datos estructurados almacenados en Hive y datos en streaming de fuentes como HDFS, Apache Flume, Kafka y Twitter.

Las principales características de Spark son:

- **Velocidad:** con Spark las aplicaciones que se ejecutan en clústeres Hadoop se ejecutan hasta 100 veces más rápido.
- **Facilidad de uso:** Spark está implementado en el lenguaje Scala y corre sobre la JVM, pero permite escribir aplicaciones, rápidamente, con diferentes lenguajes, tales como **Scala, Java, SQL, Python y R.** La codificación es mucho más sencilla que con MapReduce.
- **Combina SQL, Streaming y Analítica compleja:** además de las operaciones simples de "Map" y "Reduce", Spark admite consultas SQL, datos en streaming y analítica compleja como Machine Learning y algoritmos de gráficos.
- **Funciona en todas partes:** Spark está diseñado para ejecutarse en diferentes plataformas como Hadoop o Apache Mesos. También, puede ejecutarse en modo "standalone" o en la nube. Spark tiene disponibles un montón de conectores, puede acceder a HDFS, Cassandra, HBase, Amazon S3 y otras diversas fuentes de datos.

Si lo comparamos con MapReduce las mejoras serían las siguientes:

- **Rendimiento mejorado:** Spark maneja trabajos de procesamiento batch hasta cien veces más rápido que MapReduce. Simplemente reduce el número de hilos y escrituras en el disco.
- **Gestión simplificada:** Spark permite a los usuarios realizar procesamiento batch, procesamiento streaming y machine learning en el mismo clúster. Esto simplifica el procesamiento de datos y hace que la infraestructura sea más fácil de administrar.
- **Procesamiento streaming:** Hadoop MapReduce procesa datos almacenados en estático, pero Spark permite el procesamiento streaming a través del componente Spark Streaming.
- **Almacenamiento en caché:** Spark almacena en memoria los resultados de los cálculos parciales, lo que garantiza cálculos de latencia más baja. Los cálculos de MapReduce están completamente orientados a usar el disco.
- **Fácil de usar:** escribir código Spark es más sencillo y necesita muchas menos líneas en comparación con MapReduce.

## PIENSA UN MINUTO

Spark trabaja en memoria de manera distribuida para ser eficiente, y la memoria, como sabemos, es volátil. Pero entonces, ¿Cómo resuelve Spark la tolerancia a fallos?

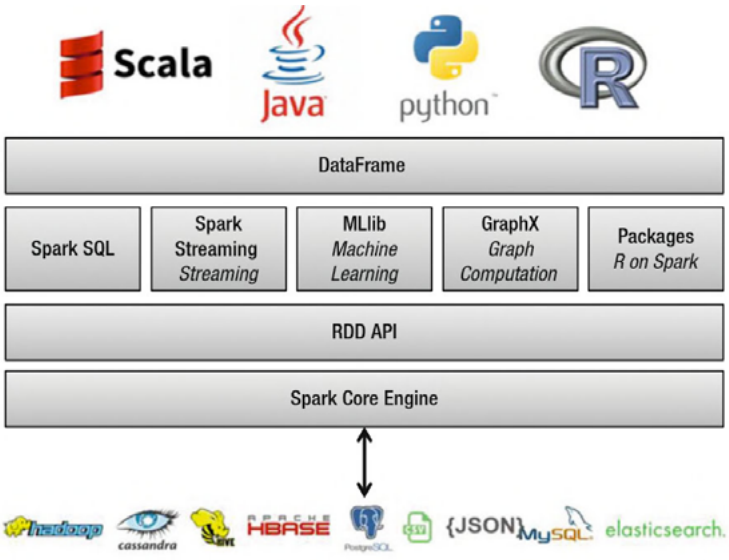
Necesitamos una memoria distribuida tolerante a fallos, para ello, Spark introduce en nueva interfaz llamada **Resilient Distributed Dataset o RDD.**

Los RDD tienen las siguientes características:

- Es la unidad básica de computación en Spark.
- Conceptualmente, es similar a una lista tradicional.
- Contiene cualquier tipo de elementos:
  - Primitivos (int, float, etc.).
  - Secuencias (tuplas, listas, dicts, etc.).
  - Objetos serializables.
- Un RDD se crea a partir de una fuente de datos o a partir de otro RDD.

Los componentes principales que forman parte de Spark son:

- **Spark Core:** tiene los componentes básicos de gestión de recursos, tolerancia a fallos, interacción con sistemas de almacenamiento, pero, sobre todo, la API de RDD.
- **Spark SQL:** es el componente que permite trabajar con datos estructurados. Añade una capa extra de abstracción a los RDD, y consigue exponer los datos para que sean consultados en una forma muy similar a como se hace con SQL tradicional. Introduce dos nuevos conceptos llamados DataFrame y DataSet.
- **Spark Streaming:** sirve para programar aplicaciones en streaming. Extiende los RDD de forma que sirvan para ingestar y procesar flujos continuos de datos, y orquesta el motor de ejecución para que se transformen en ventanas de tiempo, o lo que se denomina microbatches.
- **Machine learning library (MLlib)** o "biblioteca de aprendizaje automático": se trata de una biblioteca que contiene algoritmos escalables de machine learning. Implementa los procedimientos más comunes de clasificación, regresión y clusterización. Además, proporciona las herramientas necesarias para hacer las evaluaciones de los modelos, así como su posterior serialización y guardado en disco.
- **GraphX:** es una biblioteca para grafos. Extiende la interfaz de RDD para que pueda representar grafos dirigidos y expone operaciones para manipularlos. Además, implementa algunos de los algoritmos más comunes, como el PageRank o la detección de componentes conexos.



### 5.3.1. SPARK CORE

Los RDD son la abstracción que utiliza Spark para representar conjuntos de datos distribuidos. Cada RDD está dividido en múltiples particiones, y cada una de ellas puede ser ejecutada en un nodo distinto.

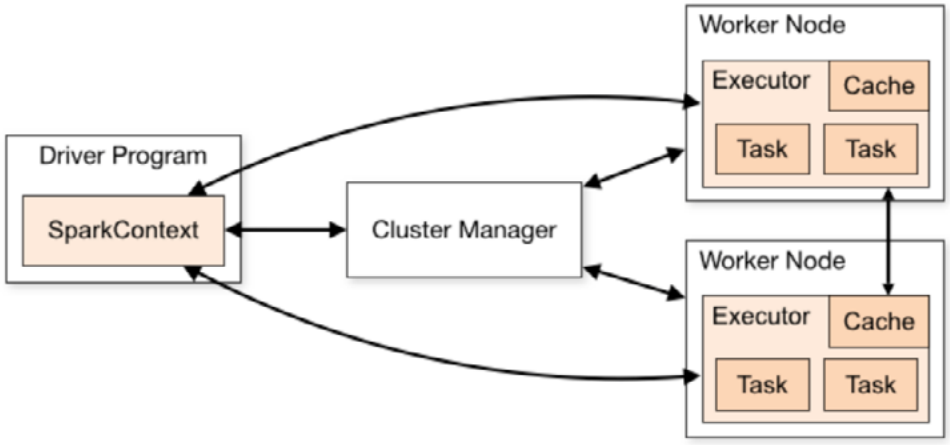
#### IMPORTANTE

Los RDD son **inmutables,** no se puede modificar su contenido. Permiten dos tipos de operaciones sobre ellos: transformaciones y acciones. Una transformación construye un nuevo RDD y una acción computa el resultado del RDD en cuestión.

Se introduce el concepto de Spark Context el cual tiene las siguientes características:

- Es lo primero que se crea en un programa con Spark, es el que nos da el acceso al clúster.
- En la consola de Spark, por defecto, se crea uno que se puede encontrar en la variable "sc".
- Cuando se arranque un programa, debemos crearlo nosotros a partir de una nueva instancia de SparkContext.
- Es un objeto que "sabe cómo interactuar con el clúster" y permite crear RDDs a partir de orígenes de datos.

En la siguiente figura, podemos ver los compontes de un clúster Spark:




#### 5.3.1.1. TRANSFORMACIONES

Las transformaciones crean un nuevo RDD a partir de uno existente. Es importante señalar que las transformaciones se ejecutan en el clúster.

Las transformaciones se acaban concatenando una tras otra y acaban formando un grafo dirigido acíclico (DAG) que no computará nada hasta que no se produzca alguna acción. Esto tiene dos ventajas:

- Los cálculos son directos.
- Te permite recuperar los datos en caso de fallos.

#### 5.3.1.2. ACCIONES

Las acciones devuelven un resultado al Driver. Es muy importante garantizar que el resultado que se devuelva entre en memoria del nodo donde resida el Driver.

PARA SABER MÁS

En la documentación de Spark, está disponible toda la información sobre RDDs, transformaciones y acciones:

- RDD Programming Guide.
- Python RDD API.
- Scala RDD API.

### 5.3.2. SPARK SQL

A pesar de su superior flexibilidad y productividad ante MapReduce, desarrollar las transformaciones y acciones adecuadas sobre RDD no es tarea fácil. Por eso mismo, e inspirándose en Apache Hive, apareció el componente Spark SQL, un componente que permite escribir consultas SQL sobre los datos de Spark.

Este componente permite a los desarrolladores establecer una estructura por encima de un RDD y tratarla como si fuera una tabla: son los llamados DataFrames y Datasets.

PARA SABER MÁS

En el siguiente enlace, se encuentra la documentación de Spark y puedes acceder a toda la información:

Spark SQL, DataFrames and Datasets Guide.

### 5.3.3. SPARK STREAMING

Hemos visto Apache Spark como un motor de procesamiento batch, de la misma forma que dicho procesamiento ocurre en los RDD, Spark Streaming proporciona los llamados DStreams (discretized streams), a los cuales se les aplicarán transformaciones que representarán todas las operaciones que se deben efectuar en un stream infinito de datos.

### IMPORTANTE

Spark Streaming no es una herramienta pura y pensada desde el principio para el procesamiento streaming, ya que utiliza la aproximación de microbatches para la computación de flujos de datos. Spark nació para el procesamiento batch y amplió su funcionalidad para soportar streaming.

### PARA SABER MÁS

Si accedes a la documentación de Spark, podrás ver toda la información:

Spark Streaming Programming Guide.

## 5.3.4. MACHINE LEARNING LIBRARY (MLLIB)

Machine Learning Library (MLlib) fue lo que, inicialmente, movió a sus creadores para concebir el proyecto.

Su objetivo es dar accesibilidad a grandes volúmenes de datos a algoritmos de inteligencia artificial y facilitar todo el proceso de extracción de features (“características”), generación de modelos, persistencia, y su posterior testing (“puesta a prueba”) para comprobar el error en las predicciones.

La principal interfaz para interactuar con MLlib es la de DataFrames, y la librería proporciona una API de más alto nivel para establecer secuencias de transformaciones y estimadores. Su nombre, ML Pipelines, hace referencia a la concatenación de los eventos como si fueran tuberías.

### PARA SABER MÁS

Si accedes a la documentación de Spark, podrás ver toda la información:

Machine Learning Library (MLlib) Guide.

## 5.3.5. GRAPHX

Apache Spark proporciona una implementación por encima del motor de RDD para ejercer computación sobre grafos. Da la posibilidad al desarrollador de representar multígrafos dirigidos en una estructura denominada property graph (“grafo de propiedades”). Ahí, tanto los vértices como las aristas pueden tener atributos, y proporciona una serie de rutinas para ejercer transformaciones sobre los datos pero, también, para que el contenido del grafo pueda ser accesible a consultas interactivas.

Además, proporciona implementaciones de los algoritmos más comunes sobre grafos, como PageRank, el cálculo de componentes conexos y, también, una API para programar Pregel sobre los property graph.

## PARA SABER MÁS

Para profundizar más sobre GraphX, puedes acceder a la documentación de Spark:

GraphX Programming Guide.

## PARA SABER MÁS

Para ampliar información sobre este apartado (5. Procesamiento del dato: Hadoop, Spark y Map Reduce) te recomendamos leer el siguiente material en pdf propiedad IL3-UB:

Módulo: Herramientas Big Data (tec_mbde_hbd.pdf)

# IDEAS CLAVE

- Las tecnologías Big Data surgieron para dar solución a los problemas de volumen, velocidad y variedad.
- El lenguaje SQL sigue siendo ampliamente utilizado en la actualidad, por lo que es necesario su conocimiento.
- Las bases de datos no relacionales nos permiten resolver nuevos problemas derivados de la existencia de datos semiestructurados y no estructurados, así como los problemas derivados de la escalabilidad de las soluciones.
- Los tres paradigmas de computación son: Procesamiento Batch, Procesamiento Streaming y Procesamiento Híbrido.
- El motor de computación distribuido Apache Spark y sus diferentes componentes nos facilitan el desarrollo de aplicaciones sobre grandes volúmenes de datos.