



Big Data Aplicado

Curso de especialización en Inteligencia Artificial y Big Data.



2. Gestión de Soluciones

En una **empresa** u organización, **los datos generados** a diario son, principalmente, aquellos derivados de las **operaciones rutinarias** de la empresa. Datos que **tradicionalmente**:

- **se almacenaban en bases de datos relacionales y**
- **su manipulación se correspondía con transacciones realizadas sobre la base de datos.**

El objetivo de cualquier organización es **seleccionar esos datos para**:

- **realizar estudios y análisis que permitan generar informes** que, a su vez,
- **permitan a la empresa extraer información para tomar decisiones estratégicas que conduzcan a la organización al éxito.**



2.1 Sistemas de ayuda a la decisión

El **crecimiento exponencial** de los datos manejados por una organización **ha hecho que los computadores sean las únicas herramientas capaces de procesar estos datos para obtener información y ofrecer ayuda en la toma de decisiones**. En este contexto, **aparecen los sistemas de ayuda a la decisión o Decision Support Systems (DSS)** que ayudan a quienes ocupan puestos de gestión a tomar decisiones o elegir entre diferentes alternativas

Sistema de ayuda a la decisión: Conjunto de técnicas y herramientas tecnológicas desarrolladas para procesar y analizar datos para ofrecer soporte en la toma de decisiones a quienes ocupan puestos de gestión o dirección en una organización. Para ello, el sistema combina los recursos de los gestores junto con los recursos computacionales para optimizar el proceso de toma de decisiones.

2.1 Sistemas de ayuda a la decisión

Los DSS se basan en tecnologías avanzadas de procesamiento de datos, incluyendo la inteligencia artificial, el análisis de datos en tiempo real y la visualización de datos. Permiten a las organizaciones tomar decisiones más fundamentadas, identificar patrones y tendencias en los datos, evaluar riesgos y oportunidades, y responder de manera más ágil a cambios en el entorno empresarial.





2.1 Sistemas de ayuda a la decisión

Mientras que las **bases de datos relacionales** han sido tradicionalmente el componente del back-end **en el diseño de sistemas de ayuda a la decisión**, los almacenes de datos se han convertido en una opción mucho más competitiva como elemento back-end al mejorar el rendimiento de éstas.

Los campos de aplicación de los almacenes de datos no se reducen únicamente al ámbito empresarial, sino que cubren multitud de dominios como las **ciencias naturales, demografía, epidemiología o educación, entre otros muchos**. La propiedad común a todos estos campos y que hace de los almacenes de datos una adecuada solución en estos ámbitos es la **necesidad de almacenamiento y herramientas de análisis que permitan obtener en tiempos razonables información y conocimiento útiles para mejorar el proceso de toma de decisiones**.



2.2 Almacenes de datos: Concepto

La aparición de los **almacenes de datos** está ligada, principalmente, a una serie de retos que es necesario abordar para **convertir los datos transaccionales con los que trabaja una base de datos relacional en información para generar conocimiento y dar soporte al proceso de toma de decisiones.**

- **Accesibilidad.**
- **Integración.**
- **Consultas mejoradas.**
- **Representación multidimensional.**



2.2 Almacenes de datos: Concepto

- **Accesibilidad**: Desde cualquier dispositivo, a cualquier tipo de usuario y a **gran cantidad de información que no puede ser almacenada de forma centralizada**. La accesibilidad, en este sentido, debe **hacer frente al problema de la escalabilidad del sistema y de los datos que este maneja**.
- **Integración**: Referente a la gestión de **datos heterogéneos, con distintos formatos, y provenientes de distintos ámbitos** de la organización. **Una correcta integración debe garantizar a su vez la corrección y completitud de los datos integrados**.



2.2 Almacenes de datos: Concepto

- **Consultas mejoradas:** Permitiendo **incluir operadores avanzados y dar soporte a herramientas y procedimientos que posibiliten obtener el máximo partido de los datos existentes**. De este modo, será posible obtener información precisa para realizar un análisis eficiente.
- **Representación multidimensional:** **Proporciona herramientas para analizar de forma multi-dimensional los datos del sistema**, incluyendo datos de diferentes unidades de la organización con el objetivo de **proporcionar herramientas de análisis y visualización multi-dimensional para mejorar el proceso de toma de decisiones**.



2.2 Almacenes de datos:

Almacén de datos (Data Warehouse): Colección de datos orientados a temas, integrados, variante en el tiempo y no volátil que da soporte al proceso de toma de decisiones de la dirección (ver **Inm02**).

Para entender correctamente esta definición, es necesario ahondar en las características que incluye la misma:

- **Orientados a temas:** Es decir, no orientado a procesos (transacciones), sino a entidades de mayor nivel de abstracción como “artículo” o “pedido”.
- **Integrados:** Almacenados en un formato uniforme y consistente, lo que implica depurar o limpiar los datos para poder integrarlos.
- **Variante en el tiempo:** Asociados a un instante de tiempo (mes, trimestre año...)
- **No volátiles:** Se trata de datos persistentes que no cambian una vez se incluyen en el almacén de datos.



2.2 Almacenes de datos: Concepto

El diseño y funcionamiento de los almacenes de datos se basa en el sistema de procesamiento analítico en-línea, OLAP. Este sistema se encarga del **análisis, interpretación y toma de decisiones acerca del negocio**, en contraposición a los sistemas de procesamiento de transacciones en línea, OLTP. Este sistema es encarnado por las bases de datos operacionales, las cuales ejecutan las transacciones de procesos operacionales del día a día.

- los sistemas OLTP están dirigidos por la tecnología y orientados a automatizar las operaciones del día a día de la organización.
- los sistemas OLAP están dirigidos por el negocio y proporcionan herramientas para tomar decisiones a largo plazo, mejorando la estrategia y la competitividad de la organización.



2.2 Almacenes de datos: Concepto

Actividad: Busque las diferencias entre BBDD Operacionales y Almacenes de datos

Característica	BBDD Operacionales	Almacén de Datos
<i>Objetivo</i>		
<i>Usuarios</i>		
<i>Trabaja con...</i>		
<i>Acceso</i>		
<i>Datos</i>		
<i>Integración</i>		
<i>Calidad</i>		
<i>Temporalidad Datos</i>		
<i>Actualizaciones</i>		
<i>Modelo</i>		
<i>Optimización</i>		

2.2 Almacenes de datos: Concepto

Tabla 2.1: Diferencias entre BBDD Operacionales y Almacenes de Datos (ver [GRO9](#)).

Característica	BBDD Operacionales	Almacén Datos
<i>Objetivo</i>	Depende de la aplicación	Toma de decisiones
<i>Usuarios</i>	Miles	Cientos
<i>Trabaja con...</i>	Transacciones predefinidas	Consultas y análisis específicos
<i>Acceso</i>	Lectura y escritura a cientos de registros	Principalmente lectura. Miles de registros
<i>Datos</i>	Detallados, numéricos y alfanuméricos	Agregados, principalmente numéricos
<i>Integración</i>	En función de la aplicación	Basados en temas, con mayor nivel de abstracción
<i>Calidad</i>	Medida en términos de integridad	Medida en términos de consistencia
<i>Temporalidad</i>	Solo datos actuales	Datos actuales e históricos
<i>Datos</i>		
<i>Actualizaciones</i>	Continuas	Periódicas
<i>Modelo</i>	Normalizado	Desnormalizado, multidimensional
<i>Optimización</i>	Para acceso OLTP a parte de la BBDD	Para acceso OLAP a gran parte de la BBDD

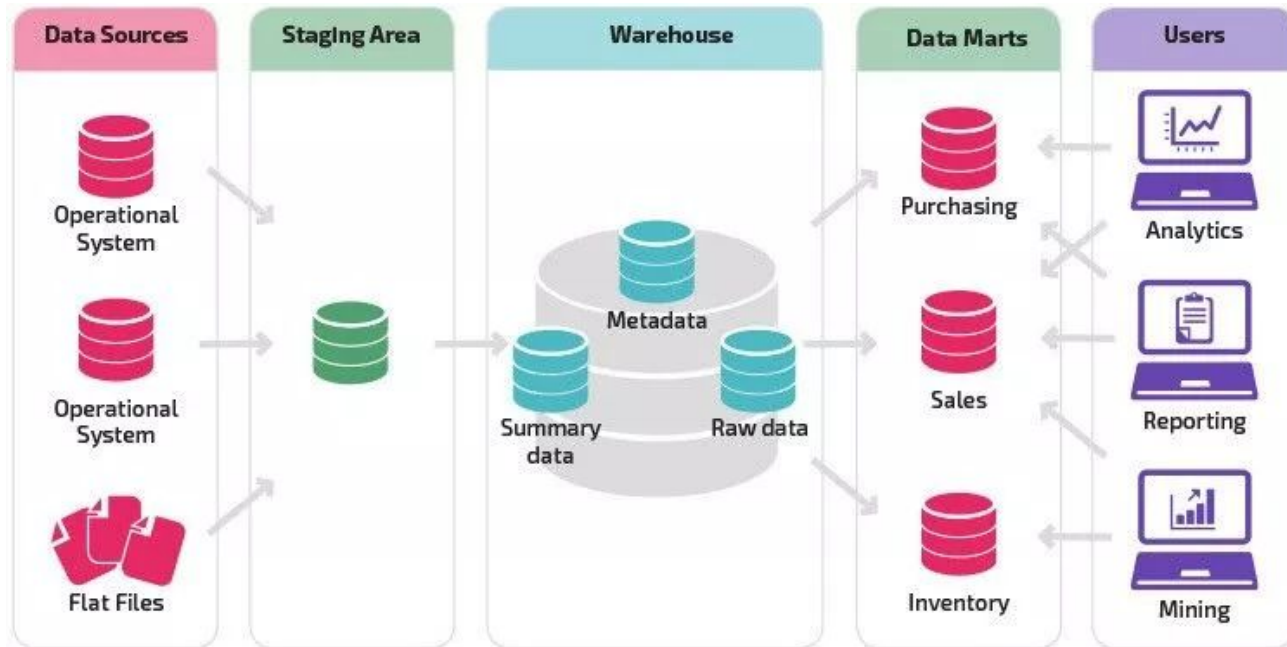


2.3 Almacenes de datos: Arquitectura

Las arquitecturas disponibles para el diseño de almacenes de datos se basan, principalmente, en **garantizar que el sistema cumpla una serie de propiedades** esenciales para su óptimo funcionamiento:

- **Separación:** De los datos transaccionales y los datos estratégicos que sirven como punto de partida a la toma de decisiones.
- **Escalabilidad:** A nivel hardware y software, para actualizarse y garantizar el correcto funcionamiento del sistema a medida que el número de datos y usuarios aumenta.
- **Extensiones:** Permitiendo integrar e incluir nuevas aplicaciones sin necesidad de rediseñar el sistema completo.
- **Seguridad:** Monitorizando el acceso a los datos estratégicos guardados en el almacén de datos.

2.3 Almacenes de datos: Arquitectura



2.3.1 Almacenes orientadas a la estructura

Las arquitecturas orientadas a la estructura reciben su nombre debido a que están diseñadas poniendo especial énfasis en el número de capas y elementos que componen la arquitectura del sistema de almacén de datos.

- **Arquitectura de una capa:** Esta arquitectura busca minimizar el almacenamiento de datos al eliminar redundancias. Utiliza un almacén de datos virtual, con un middleware que interpreta los datos operativos y proporciona una vista multidimensional. Sin embargo, su simplicidad impide cumplir con la propiedad de separación, ya que los análisis se realizan directamente sobre los datos operativos.

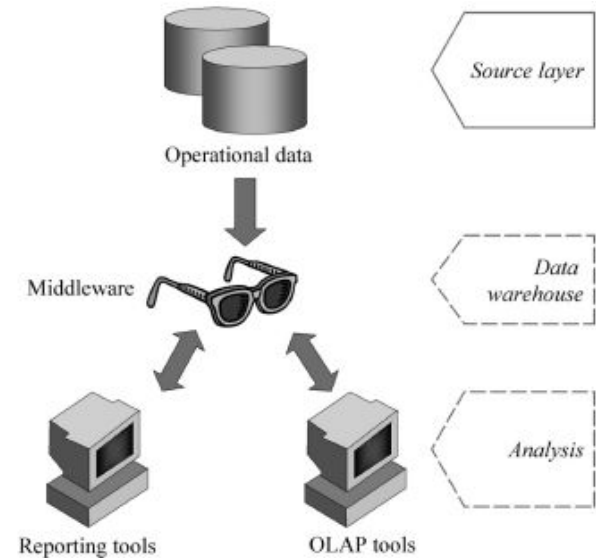


Figura 2.1: Almacén de datos. Arquitectura de una capa.

2.3.1 Almacenes orientadas a la estructura

- **Arquitectura de dos capas:** Diseñada con el objetivo de solucionar el problema de la separación que presentaba la arquitectura de una capa. El esquema consigue subrayar la separación entre los datos disponibles y el almacén de datos a través de los siguientes componentes:
 - **Capa de origen (fuente):** Se corresponde con los orígenes y fuentes de los datos heterogéneos que se pretenden incorporar al almacén de datos.
 - **Puesta a punto:** Proceso por el cual se utilizan **herramientas de Extracción, Transformación y Carga (ETL)** para **extraer, limpiar, filtrar, validar y cargar datos en el almacén de datos.**
 - **Capa de almacén de datos:** Almacenamiento centralizado de la información en el almacén de datos, el cual puede ser utilizado para crear data marts o repositorios de metadatos.
 - **Análisis:** Conjunto de procesos a partir de los cuales los datos son eficientemente y flexiblemente analizados, generando informes y simulando escenarios hipotéticos para dar soporte a la toma de decisiones.

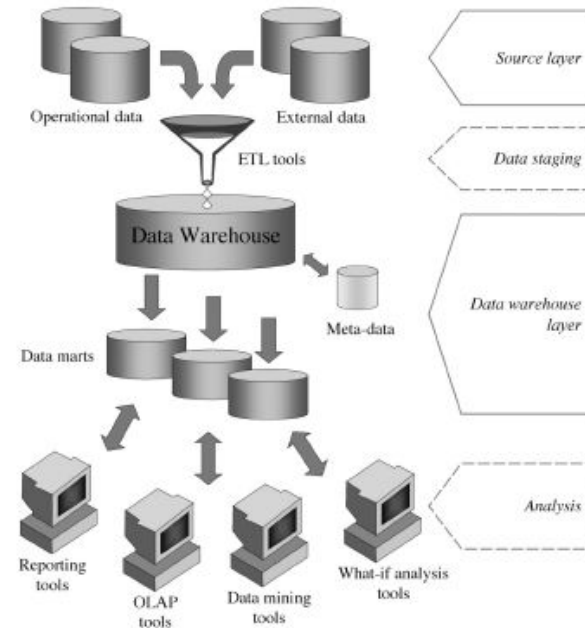


Figura 2.2: Almacén de datos. Arquitectura de dos capas.

2.3.1 Almacenes orientadas a la estructura

- **Arquitectura de tres capas:** Incluye una capa llamada de datos reconciliados o almacén de datos operativos. Con esta capa, **los datos operativos obtenidos tras la limpieza y depuración son integrados y validados, proporcionando un modelo de datos de referencia** para toda la organización. De este modo, el almacén de datos no se nutre de los datos de origen directamente, sino de los datos reconciliados generados, los cuales también son utilizados para realizar de forma más eficiente tareas operativas, como la realización de informes o la alimentación de datos a procesos operativos.

Esta capa de datos reconciliados también puede implementarse de forma virtual en una arquitectura de dos capas, ya que se define como una vista integrada y coherente de los datos de origen.

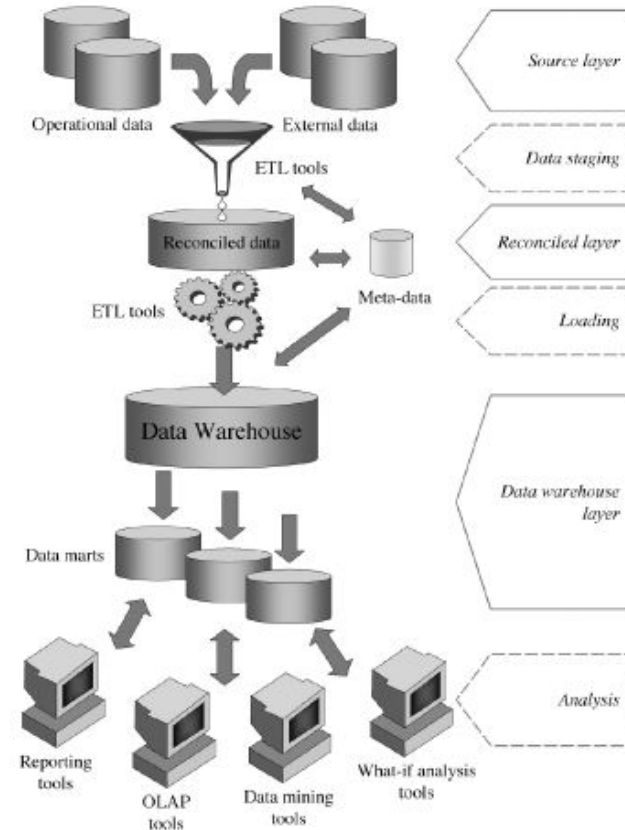


Figura 2.3: Almacén de datos. Arquitectura de tres capas.



2.3.2 Arquitecturas orientadas a la empresa

Arquitectura de data marts independientes

Arquitectura preliminar en la que los distintos data marts son diseñados de forma independiente y contruidos de forma no integrada. Suele utilizarse en los inicios de implementación de proyectos de almacenes de datos y reemplazada a medida que el proyecto va creciendo.

Arquitectura en bus

Similar a la anterior, asegura la integración lógica de los data marts creados, ofreciendo una visión amplia de los datos de la empresa y permitiendo realizar análisis rigurosos de los procesos que en ella se llevan a cabo.



2.3.2 Arquitecturas orientadas a la empresa

Arquitectura hub-and-spoke (centro y radio)

Esta arquitectura es muy utilizada en almacenes de datos de tamaños medio y grande. Su diseño pone especial énfasis en garantizar la escalabilidad del sistema y permitir añadir extensiones al mismo. Para ello, los datos se almacenan de forma atómica y normalizada en una capa de datos reconciliados que alimenta a los data marts construidos que contienen, a su vez, los datos agregados de forma multidimensional. Los usuarios acceden a los data marts, si bien es cierto que también pueden hacer consultas directamente sobre los datos reconciliados.

Arquitectura centralizada

Se trata de un caso particular de la arquitectura hub-and-spoke. En ella, la capa de datos reconciliados y los data marts se almacenan en un único repositorio físico.



2.3.2 Arquitecturas orientadas a la empresa

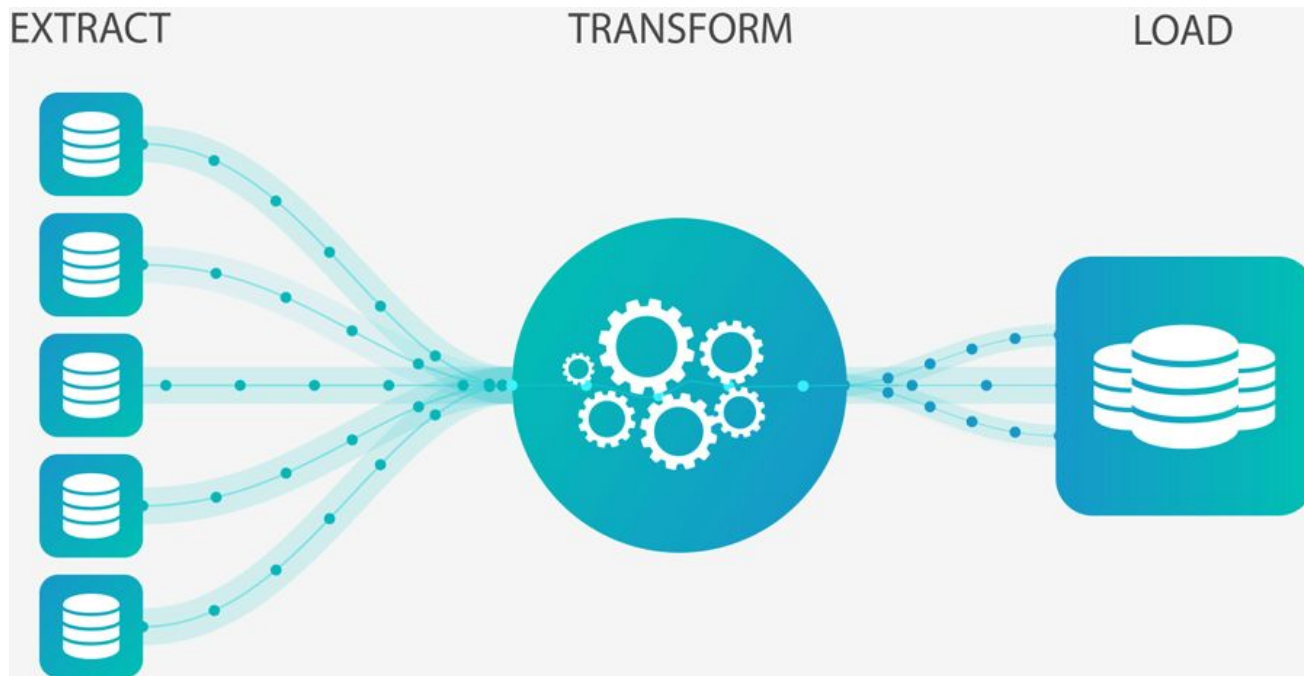
Arquitectura federada

Se trata de un tipo de arquitectura muy utilizada en entornos dinámicos, cuando se pretende integrar almacenes de datos o data marts existentes con otros para ofrecer un entorno único e integrado de soporte a la toma de decisiones. De esta forma, cada almacén de datos y cada data mart es integrado virtual o físicamente con los demás. Para ello, se utilizan una serie de técnicas y herramientas avanzadas como son las ontologías, consultas distribuidas e interoperabilidad de metadatos, entre otras.

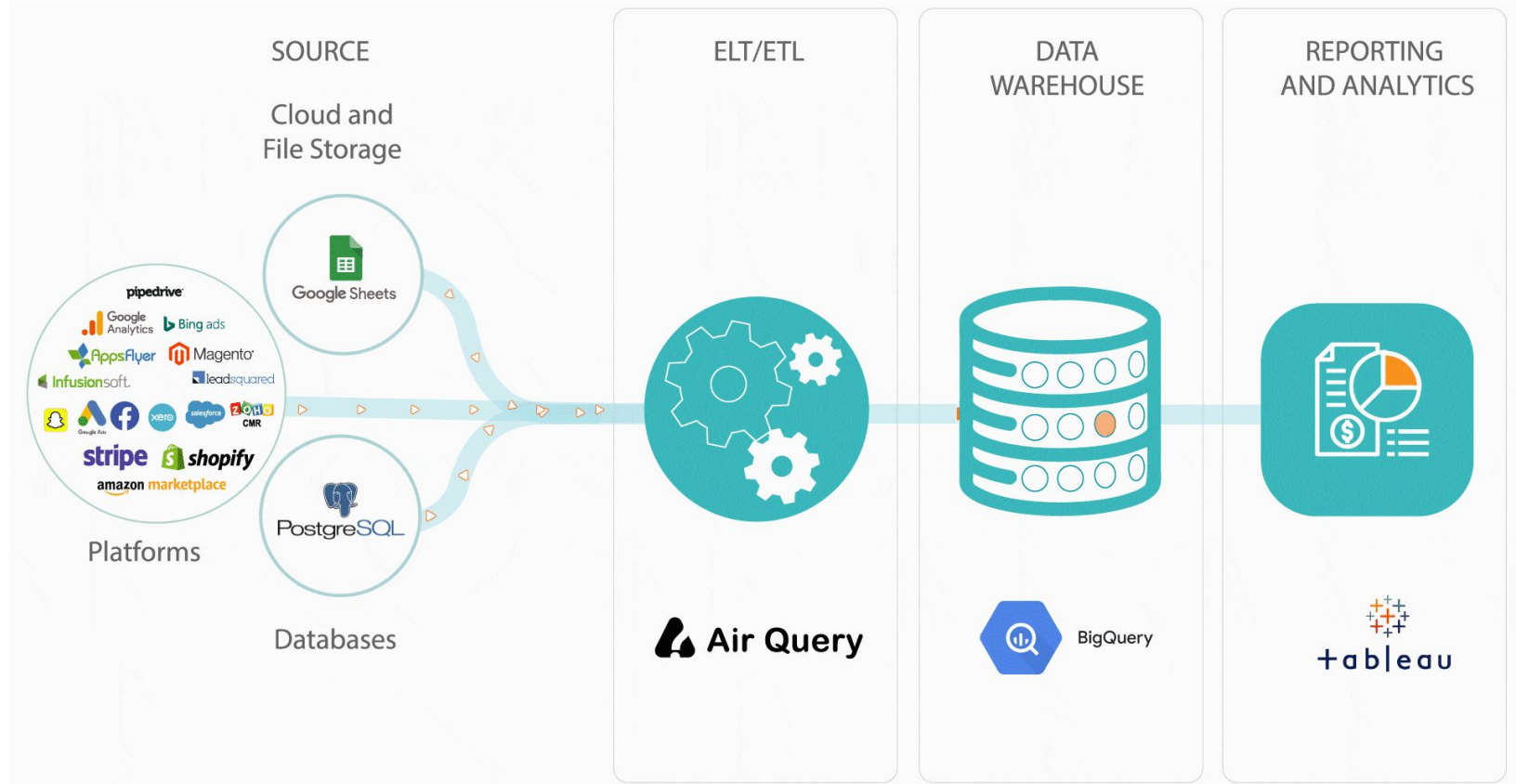
2.3.2 Arquitecturas. Resumen

Aspecto / Arquitectura	Arquitectura de Una Capa	Arquitectura de Dos Capas	Arquitectura de Tres Capas	Data Marts Independientes	Arquitectura en Bus	Hub and Spoke	Centralizada	Federada
Propósito	Almacenamiento simple de datos	Separación de Datos de Proceso	Separación de Datos, Lógica y Presentación	Almacenes para áreas específicas	Enviar datos a múltiples destinos	Centralización de datos con nodos satélite	Centralización de datos	Unión de múltiples fuentes
Estructura de Datos	Estructura única y simple	Estructura de datos y procesos separados	Estructura de datos, lógica de negocios y presentación separadas	Estructura específica para cada Data Mart	Datos enviados a múltiples destinos según necesidad	Centralización con nodos satélite conectados al nodo central	Estructura centralizada	Datos mantienen su estructura original
Capas Principales	N/A (una capa única)	Capa de Datos y Aplicación	Capa de Datos, Lógica de Negocios y Presentación	Capa de Almacenamiento de Datos	Capa de Transporte y Comunicación	N/A (enfoque en rutas de datos)	Capa de Almacenamiento de Datos	Capa de Acceso de Datos
Uso Principal	Pequeñas aplicaciones o prototipos	Aplicaciones departamentales	Aplicaciones empresariales	Departamentos o áreas específicas	Integración de sistemas	Transferencia de datos entre sistemas	Grandes organizaciones	Organizaciones con múltiples fuentes de datos
Flexibilidad y Escalabilidad	Limitada	Mayor flexibilidad	Mayor flexibilidad	Limitada flexibilidad	Mayor flexibilidad	Puede ser escalable si se planifica	Limitada flexibilidad	Mayor flexibilidad
Mantenimiento y Complejidad	Menos complejidad	Moderada complejidad	Mayor complejidad	Varía según Data Marts	Moderada complejidad	Moderada complejidad	Mayor complejidad	Moderada complejidad
Ejemplo	Base de datos plana	ETL + Data Warehouse	Data Warehouse + Capa de Aplicación	Data Mart de Ventas	Integración de datos mediante ESB	Hub central con múltiples aplicaciones	Data Warehouse Centralizado	Federación de datos de múltiples sucursales

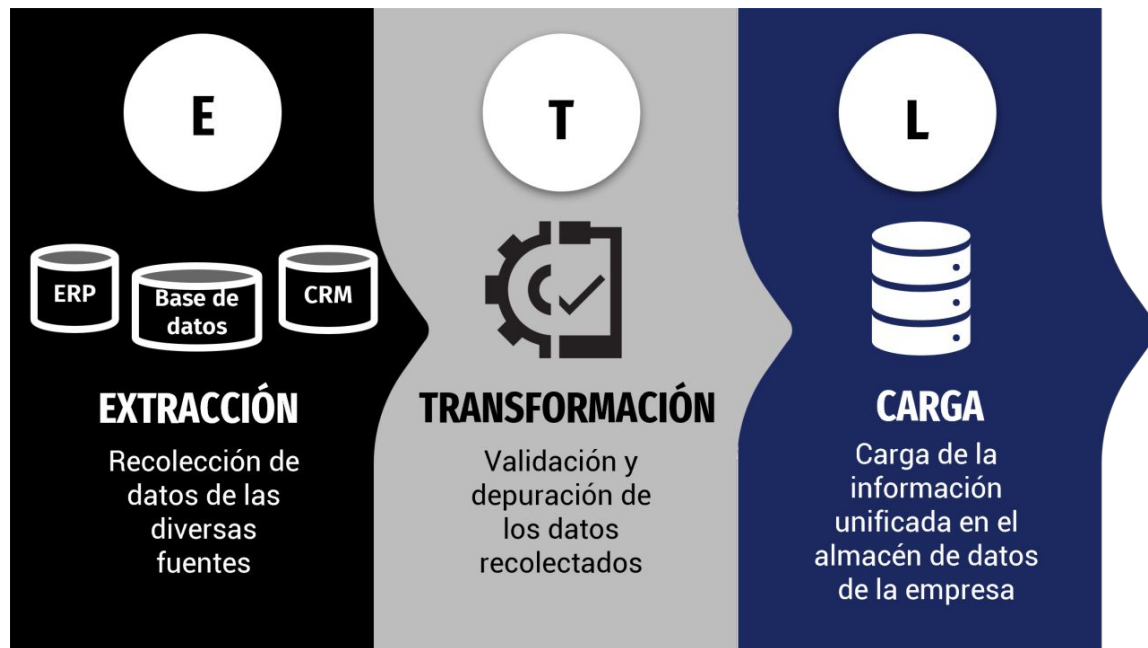
El proceso ETL. Introducción



El proceso ETL. Introducción



El proceso ETL. Introducción





2.4 Almacenes de datos. Diseño e implementación

2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

El proceso ETL es el encargado de extraer, limpiar e integrar los datos provenientes de las fuentes de datos para alimentar el almacén de datos. Este proceso también es el encargado de alimentar la capa de datos reconciliados en la arquitectura de tres capas. El proceso ETL tiene lugar cuando se prueba el almacén de datos y se lleva a cabo cada vez que el almacén de datos se actualiza. A continuación, se describen detalladamente cada una de las fases de las que consta este proceso.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Extracción

Etapa que consiste en la lectura de los datos de las distintas fuentes de las que provienen. Cuando un almacén de datos se rellena por primera vez, se suele utilizar la técnica de extracción estática, la cual consiste en extraer una instantánea de los datos operacionales. A partir de entonces, se utiliza la extracción incremental para actualizar periódicamente los datos del almacén de datos, recogiendo los cambios aplicados desde la última extracción. Para ello, se utiliza el registro mantenido por el SGBD que, por ejemplo, asocia una marca de tiempo (timestamp) a los datos operacionales para registrar cuando fueron modificados y agilizar el proceso de extracción.

En la actualidad, existe una gran cantidad de conjuntos de datos o data sets públicos, conocidos bajo el nombre de Open Data, que abarcan una gran cantidad de dominios y con los que es posible trabajar para construir soluciones big data.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Open Data: Se trata de datos que han sido generados por una fuente en particular, que abarcan un dominio temático o disciplinar y tienen atributos, dentro de los cuales está la frecuencia de actualización. Además, cuentan con una licencia específica que indica las condiciones de reutilización de los mismos.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

La **f fuente de los datos** es en muchos de los casos **el estado nacional, provincial, municipal u organizaciones comerciales**. En otras ocasiones, **la fuente de los datos es fruto del estudio o medición por parte de particulares**. Los atributos de los conjuntos de datos deben especificar cómo fueron obtenidos, incluyendo fechas de obtención, actualización y validez, así como el público involucrado, la metodología de recogida o muestreo, etc.

Algunas de las fuentes más utilizadas en la actualidad para la obtención de datos abiertos provienen de los centros nacionales e internacionales de estadísticas, como son el Instituto Nacional de Estadística de España (INE)¹, eurostat², la oficina europea de estadísticas, la Organización Mundial de la Salud (OMS)³...

Por otra parte, existen repositorios públicos de datos abiertos y a disposición de los usuarios como UCI o Kaggle, que es una comunidad de científicos de datos donde empresas y organizaciones suben sus datos y plantean problemas que son resueltos por los miembros de la comunidad.

2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Por otra parte, existen repositorios públicos de datos abiertos y a disposición de los usuarios como UCI o Kaggle, que es una comunidad de científicos de datos donde empresas y organizaciones suben sus datos y plantean problemas que son resueltos por los miembros de la comunidad.

<https://www.ine.es>

<https://ec.europa.eu/eurostat>

<https://www.who.int/es/data/gho/publications/world-health-statistics>

<https://archive.ics.uci.edu/ml/index.php>

<https://www.kaggle.com>



Figura 2.4: Fuentes para la obtención de datos abiertos.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Transformación:

La etapa de transformación es la fase clave para transformar los datos operativos en datos con un formato específico para alimentar un almacén de datos. En esta etapa, los datos se limpian y se transforman, añadiéndoles contexto y significado. En caso de implementar un almacén de datos siguiendo una arquitectura de tres capas, el proceso de transformación es el encargado de obtener la capa de datos reconciliados.

La etapa de **transformación** engloba todos los procesos de limpieza y manipulación de los datos, con el objetivo de transformar los datos operativos propios de sistemas relacionales (OLTP) en datos preparados para ser incluidos dentro del almacén de datos (OLAP).



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Transformación:

La limpieza de los datos o data cleaning engloba todos aquellos procedimientos necesarios para detectar y resolver situaciones problemáticas con los datos de partida que pudieran suponer problemas potenciales a la hora de analizarlos. Así pues, los datos de partida pueden ser incompletos, es decir, pueden contener atributos sin valor o valores agregados, incorrectos que incluyan errores o valores sin ningún significado, lo cual es común cuando los datos se introducen manualmente en el sistema, o inconsistentes cuando los cambios no son propagados a todos los módulos del sistema, los rangos de un determinado atributo son cambiantes, existen datos duplicados...



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Transformación:

En general, se distinguen dos tipos de situaciones cuando existen valores perdidos: datos perdidos completamente aleatorios y datos perdidos de forma no completamente aleatoria. En el segundo caso, puede ser interesante intentar analizar la razón de la pérdida de los datos, la cual puede ser indicativa. En muchas ocasiones, los valores perdidos tienen relación con un subconjunto de variables predictoras y no se encuentran aleatoriamente distribuidos por todas ellas. Por todo ello, las aproximaciones más comunes a la hora de gestionar datos perdidos son:



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Transformación:

Eliminación de instancias que contengan valores perdidos: la eliminación de instancias que contienen valores perdidos implica establecer un umbral de porcentaje, y si una instancia tiene un número de valores perdidos que supera este umbral, se elimina. Esta técnica es útil para conjuntos de datos grandes con pocos valores perdidos, pero debe usarse con precaución, ya que puede resultar en una pérdida significativa de información en conjuntos de datos más pequeños, con muchas instancias con valores perdidos o con un umbral muy bajo.

Asignación de valores fijos: consiste en asignar un valor fijo a todos los valores perdidos de todas las variables. Este valor puede ser el número 0 o incluso un valor desconocido Unknown o no numérico en función del lenguaje de programación utilizado.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Transformación:

Asignación de valores de referencia: asigna un valor de referencia a los valores perdidos para cada variable. Estos valores de referencia suelen ser medidas de centralización como la media o la mediana de los valores de cada variable.

Imputación de valores perdidos: consiste en la aplicación de técnicas más sofisticadas, como pueden ser técnicas estadísticas o de aprendizaje automático para predecir o averiguar los valores que se han perdido.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Finalmente, la etapa de limpieza de datos también se encarga de la detección de valores anómalos o outliers. Se trata de valores que se han introducido de forma errónea o bien a una deformación en la distribución de valores.

El proceso de detección de anomalías consiste, fundamentalmente, en dos etapas: en primer lugar, asumir que existe un modelo generador de datos, como podría ser una distribución de probabilidad. En segundo lugar, considerar que las anomalías representan un modelo generador distinto, que no coincide con el original. Existen multitud de técnicas para detectar y descartar o imputar valores anómalos, como lo son técnicas estadísticas basadas en la desviación y el rango intercuartílico o técnicas de aprendizaje automático.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Procesos de transformación más comunes:

- Estandarización de códigos y formatos de representación.
- Conversiones y combinaciones de campos.
- Correcciones.
- Integración de varias fuentes.
- Eliminación de datos y/o registros duplicados.
- Escalado y centrado.
- Discretización.
- Selección de características.

[Documento](#)



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Carga:

Se trata de la última fase de cara a incluir datos en el almacén de datos. La carga inicial de los datos puede requerir bastante tiempo al cargar de forma progresiva todos los datos históricos, por lo que es normal realizarla en horas de baja carga de los sistemas. Una vez que el almacén de datos ha sido inicialmente cargado, las sucesivas cargas de datos se pueden realizar de dos formas:

Refresco: El almacén de datos se reescribe completamente, de forma que se reemplazan los datos antiguos. El refresco es utilizado habitualmente junto con la extracción estática y suele ser una estrategia muy utilizada para la carga inicial del almacén de datos, aunque puede también realizarse a posteriori.

Actualización: Se añaden al almacén de datos solamente aquellos datos nuevos que se pretenden incluir, sin eliminar ni modificar los datos ya existentes. Esta técnica es utilizada frecuentemente junto con la extracción incremental para actualizar regularmente el almacén de datos.



2.4.1. El proceso de Extracción, Transformación y Carga (ETL)

Frameworks ETL

Bubbles (<http://bubbles.databrewery.org>): Se trata de un framework ETL escrito en Python, aunque es posible utilizarlo desde otros lenguajes.

Apache Camel (<https://camel.apache.org>): Este framework escrito en lenguaje Java y de acceso abierto se enfoca en facilitar la integración entre distintas fuentes de datos, haciendo el proceso más accesible a los desarrolladores, ofreciendo distintas herramientas para dar soporte al proceso ETL.

Keetle (<https://community.hitachivantara.com/s/article/data-integration-kettle>): Es el framework de Pentaho para dar soporte al proceso ETL. Keetle es de acceso abierto.



2.4.2. Diseño en estrella

A la hora de diseñar un almacén de datos, existen dos alternativas ampliamente utilizadas: el diseño en estrella, que promueve el diseño directo de estructuras lógicas sobre el modelo relacional, y el diseño en copo de nieve, como variante del diseño en estrella.

El proceso de desarrollo de un almacén de datos siguiendo el diseño en estrella puede estructurarse según las siguientes fases:

1. **Elegir un proceso de negocio a modelar.**
2. **Escoger la granularidad del proceso de negocio:** Los datos almacenados en el almacén de datos deben expresarse siempre al mismo nivel de detalle.
3. **Selección de medidas/hechos:** indican qué se necesita medir o evaluar en el proceso de negocio con el fin de dar respuesta a las necesidades de información y toma de decisiones por las cuales se pretende diseñar el almacén de datos.
4. **Elegir las dimensiones que se aplicarán a cada medida o hecho:** las dimensiones especifican las distintas propiedades de las medidas o hechos que se pretenden almacenar e integrar dentro del almacén de datos.



2.4.2. Diseño en estrella

Se crea una tabla central conocida como tabla de hechos, que almacena datos o medidas relacionados con diferentes combinaciones de dimensiones. Estos datos pueden ser completamente aditivos (como las ventas totales de un departamento), no aditivos (como el margen de beneficios expresado como porcentaje) o semiaditivos (datos intermedios que se pueden agregar con datos de otras dimensiones).

Además de la tabla de hechos, hay una tabla separada para cada dimensión definida. La clave primaria de la tabla de hechos se forma combinando las claves primarias de las dimensiones relacionadas, lo que crea una relación "muchos a muchos" entre todas las tablas de dimensiones relacionadas y una relación "muchos a uno" con cada tabla de dimensión por separado. Este diseño se conoce como un esquema en forma de estrella.

2.4.2. Diseño en estrella

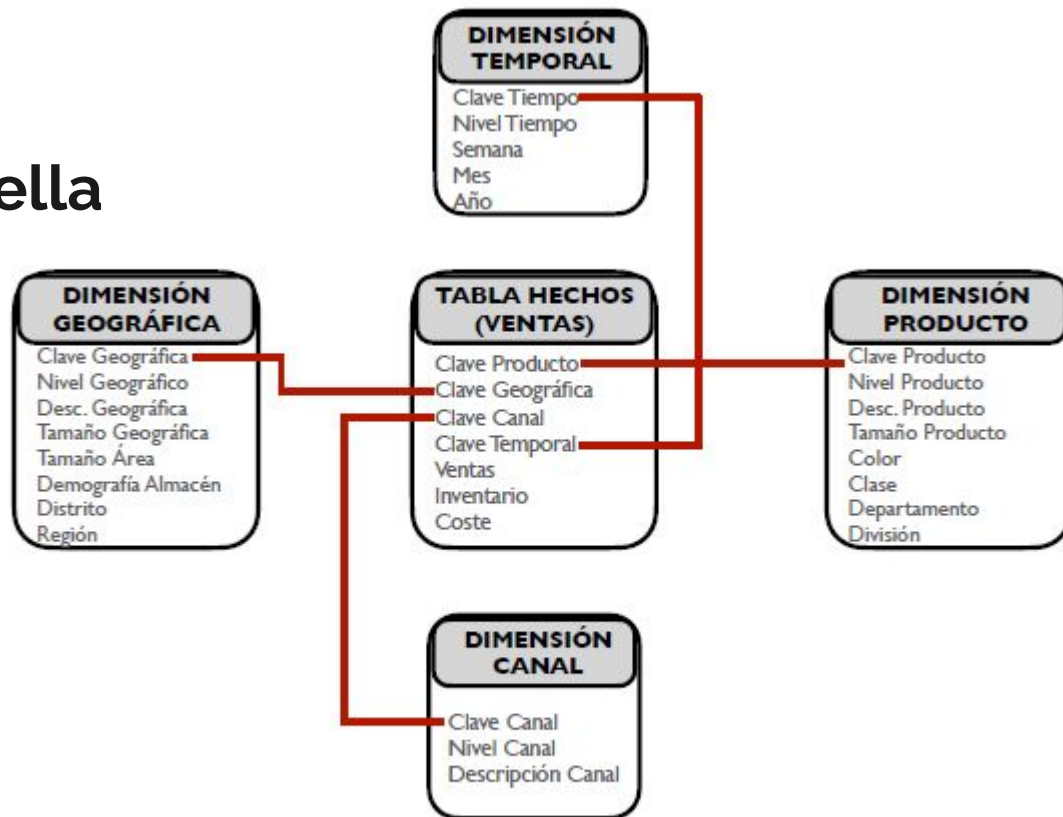


Figura 2.5: Ejemplo de almacén de datos diseñado en estrella.



2.4.2. Diseño en estrella

Entre las ventajas del diseño en estrella, destaca ser un diseño fácil de modificar que proporciona tiempos rápidos de respuesta, simplificando la navegación de los metadatos. Además, este diseño permite simular vistas de los datos que obtendrán los usuarios finales y facilita la interacción con otras herramientas.

Sin embargo, el diseño en estrella presenta algunos inconvenientes que surgen, por ejemplo, cuando se combinan dimensiones con distintos niveles de detalle. Además, cuando se pretende limitar los niveles de una dimensión se debe añadir un campo de nivel o utilizar un modelo de tipo constelación, donde se incluyen diferentes estrellas que almacenan tablas de hechos agregadas, lo cual añade complejidad al esquema.



2.4.3. Diseño en Copo de Nieve

Este diseño es muy utilizado cuando se dispone de tablas de dimensión con una gran cantidad de atributos. En esta circunstancia, el diseño de copo de nieve permite normalizar las tablas de dimensión, ofreciendo un mejor rendimiento cuando las consultas realizadas sobre el almacén de datos requieren del uso de operadores de agregación. De esta forma, la tabla de hechos deja de ser la única tabla que presenta relaciones con las demás, apareciendo nuevas relaciones que se dan entre las tablas normalizadas que componen las tablas de dimensiones. Este esquema, que incluye ramificaciones en cada dimensión que se corresponden con las tablas necesarias para su normalización, guarda un gran parecido con la estructura de un copo de nieve, lo cual da nombre a este diseño



2.4.3. Diseño en Copo de Nieve

La diferencia entre los diseños en estrella y copo de nieve radica fundamentalmente en la modelización de las tablas de dimensiones. Para obtener un diseño en copo de nieve, basta con partir de un diseño en estrella en el que, tras un proceso de normalización, se obtienen varias tablas de dimensión a partir de una tabla desnormalizada. Dentro del diseño de copo de nieve, es posible distinguir entre copo de nieve parcial, en el que solo algunas de las dimensiones son normalizadas y copo de nieve completo, en el que todas las dimensiones del esquema son normalizadas.

2.4.3. Diseño en Copo de Nieve

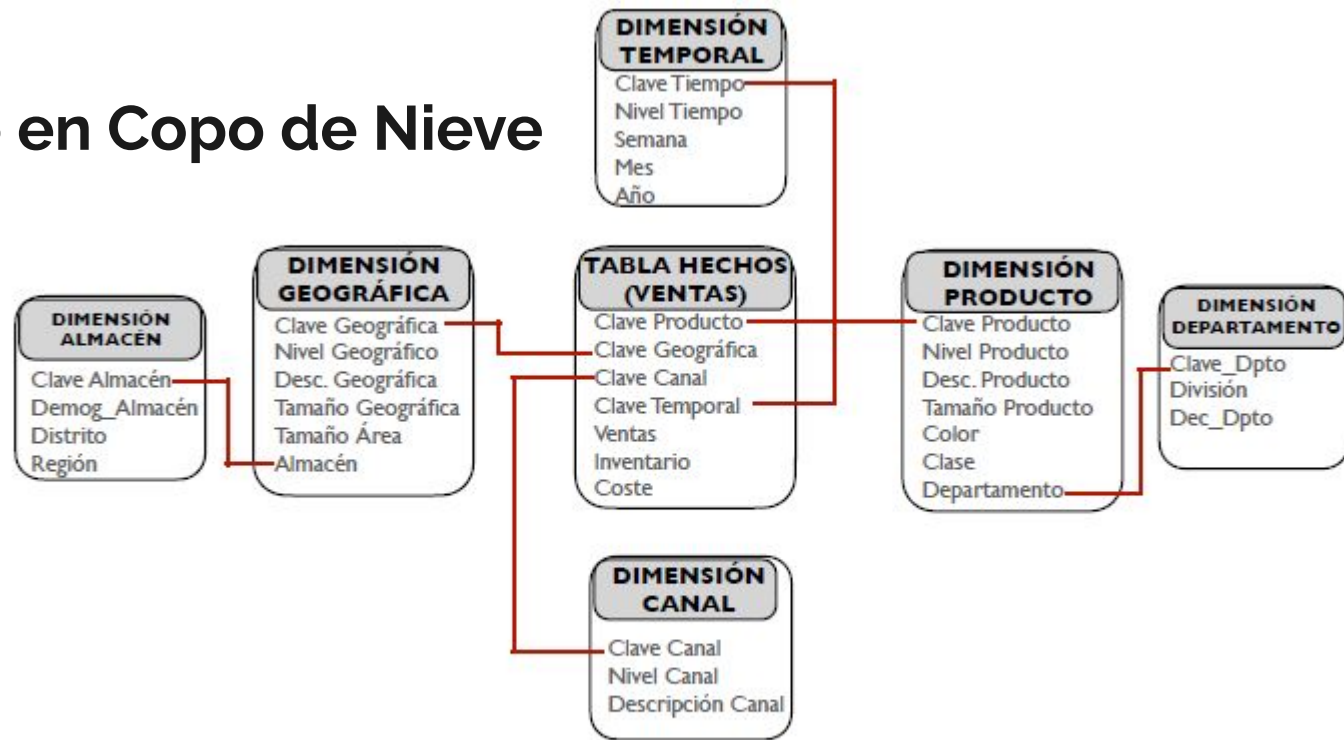


Figura 2.6: Ejemplo de almacén de datos diseñado en copo de nieve.



2.4.3. Diseño en Copo de Nieve

La normalización de las tablas de dimensiones proporciona al diseño de copo de nieve la mejora de la eficiencia en la realización de consultas complejas que requieren el uso de operadores de agregación. Sin embargo, este diseño es conceptualmente más difícil de implementar, ya que incluye un mayor número de tablas y requiere de realizar muchos joins entre ellas, lo que incrementa los tiempos de consulta.



Resumen. Diseños

El diseño en estrella permite crear un almacén de datos con una estructura centralizada. El diseño se compone de una tabla de hechos central, que recoge los valores de las medidas del proceso de negocio que se pretende modelar. Rodeando a la tabla de hechos, se incluyen tantas tablas como dimensiones se hayan especificado en el modelo, las cuales se relacionan entre sí a través de la tabla de hechos.

El diseño en copo de nieve permite crear un almacén de datos a partir de un diseño en estrella. Para ello, las tablas de dimensiones se normalizan, generando más de una tabla para cada una de las dimensiones, mejorando la eficiencia de consultas complejas que requieren el uso de operadores avanzados.



Big Data Aplicado

Curso de especialización en Inteligencia Artificial y Big Data.



3. Gestión de Soluciones

3.1 Bases de datos NoSQL (“Not only SQL”.)

NoSQL no engloba solamente un conjunto de bases de datos y sistemas de almacenamiento que no utilizan SQL para su gestión sino que es una categoría general de sistemas de bases de datos que, además de SQL, utilizan otra serie de tecnologías para almacenar y gestionar los datos.

3. Gestión de Soluciones

3.1 Bases de datos NoSQL (“Not only SQL”.)





3.1 Bases de datos NoSQL (“Not only SQL”.)

Las transacciones realizadas en los sistemas de bases de datos relacionales garantizan la consistencia y escalabilidad de las operaciones a través de una serie de características especificadas en el modelo ACID.

Atomicidad (A): Garantiza que una transacción se ha realizado completamente o no, evitando que algunas operaciones se ejecuten a medias.

Consistencia (C): También conocida como integridad, asegura que cualquier transacción realizada desde un estado seguro en la base de datos conducirá a esta a otro estado también seguro o válido.

Aislamiento (I: Isolation): Permite que el resultado en el estado del sistema sea el mismo, independientemente del modo de ejecución de las transacciones (secuencial o concurrente).

Durabilidad (D): O persistencia, garantiza que a pesar de que se produzca un fallo en el sistema, los cambios realizados por una transacción persistirán.



3.1 Bases de datos NoSQL (“Not only SQL”.)

Las bases de datos NoSQL siguen el modelo conocido como BASE, más flexible que ACID en aquellos sistemas de bases de datos que no se adhieren estrictamente al modelo relacional.

Disponibilidad básica (BAsic Availability): El sistema siempre ofrece una respuesta a una petición de datos, aunque los datos sean inconsistentes, estén en fase de cambio o incluso se produzca un fallo.

Soft-state (S): La consistencia de este tipo de bases de datos es eventual, por lo que el estado del sistema cambia a lo largo del tiempo, aunque no haya habido entradas de datos.

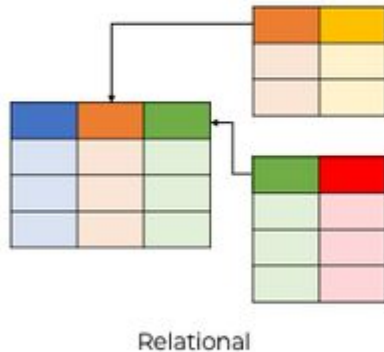
Eventual Consistency (E): Cuando se dejan de recibir datos, el sistema se vuelve consistente. Así pues, los datos se propagan a todo el sistema, pero este continúa recibiendo datos sin evaluar la consistencia de los mismos para cada transacción antes de avanzar a la siguiente.

3.1 Bases de datos NoSQL (“Not only SQL.”)

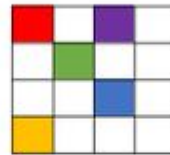


3.1 Bases de datos NoSQL (“Not only SQL.”)

SQL DATABASES



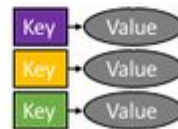
NoSQL DATABASES



Column



Graph



Key-Value



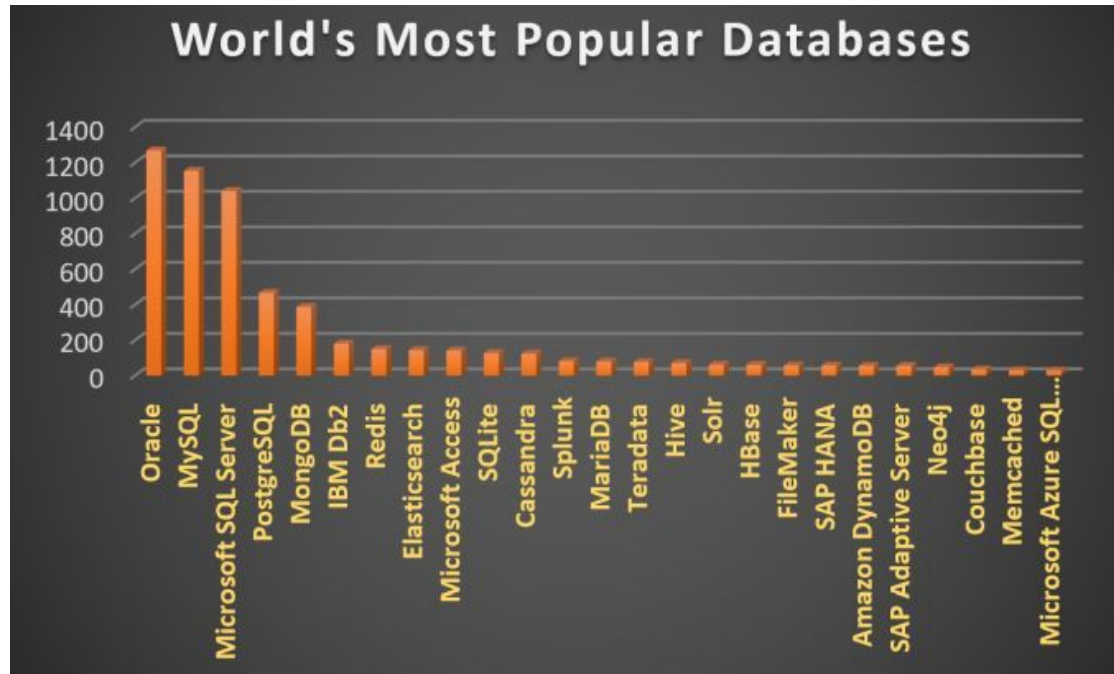
Document

3.1 Bases de datos NoSQL (“Not only SQL”.)



Base de datos NoSQL: Sistema de bases de datos no relacional que utiliza distintas tecnologías para la gestión y almacenamiento distribuido de datos masivos sin un esquema estricto subyacente. Esta familia de base de datos permite incrementar la disponibilidad y escalabilidad del sistema, mejorando su rendimiento.

3.1 Bases de datos NoSQL (“Not only SQL”.)



source:

<https://www.c-sharpcorner.com/article/what-is-the-most-popular-database-in-the-world/>



3.2. Bases de datos XML (eXtensible Markup Language)

El lenguaje XML es un lenguaje de marcas extensible derivado del lenguaje SGML (Standard Generalized Markup Language). Se trata de un lenguaje ideado para la definición y gestión de documentos que permite representar datos estructurados y semi-estructurados. En la actualidad, XML se ha convertido en un formato estándar para la comunicación entre aplicaciones, facilitando su integración. Entre las principales ventajas de XML destacan:

Datos autodocumentados: Gracias a la posibilidad de definir cualquier tipo de etiquetas, no es necesario conocer el esquema para entender el significado de los datos.

Extensión: Permite adaptar el lenguaje fácilmente a cualquier dominio.

Anidamiento: La definición de estructuras anidadas dota a los documentos de gran flexibilidad para transferir y consultar información.

Flexibilidad: Los documentos XML no tienen un formato estricto ni rígido, permitiendo añadir o ignorar información en ellos

3.2. Bases de datos XML (eXtensible Markup Language)

Estructura de un documento XML

La estructura de un documento XML **está compuesta de elementos**. Un elemento es una porción de documento delimitado por un par de etiquetas **<> (apertura)** y **< / > (cierre)** entre las cuales se incluye un texto llamado **contexto del elemento**. Un elemento sin contexto puede abreviarse mediante el uso de una única etiqueta `<nombre_elemento/>`. La estructura de un XML se forma anidando elementos para especificar la estructura del documento, formando una estructura de árbol. El fragmento de código 3.1 representa la estructura básica de un documento XML que representa un listado de asignaturas.

Listado 3.1: Listado de Asignaturas

```
1 <Asignaturas_Primer>
2   <Asignatura>
3     <titulo> Cálculo </titulo>
4     <tipo> Anual </tipo>
5     <profesor>
6       <nombre> Ana </nombre>
7       <apellido> Sanz </apellido>
8     </profesor>
9     <profesor>
10      <nombre> Roberto </nombre>
11      <apellido> Hernández </apellido>
12    </profesor>
13  </Asignatura>
14  <Asignatura>
15    <titulo> Física </titulo>
16    <tipo> Semestral </tipo>
17    <profesor>
18      <nombre> Carmelo </nombre>
19      <apellido> Moya </apellido>
20    </profesor>
21  </Asignatura>
22 </Asignaturas_Primer>
```



3.2. Bases de datos XML (eXtensible Markup Language)

Los elementos de un documento pueden contener atributos. Los atributos se incluyen en la definición del elemento, dentro de la etiqueta de apertura. Sintácticamente, se representan mediante un par nombre = valor. A diferencia de un subelemento, un atributo sólo puede aparecer una vez en una etiqueta. El uso de atributos y su diferencia con respecto a los elementos no siempre es evidente. A nivel de documento, los atributos se introducen para añadir texto que no se visualizará en el documento, mientras que los subelementos formarán parte del contenido del documento. A nivel de datos, la diferencia es insustancial, puesto que la misma información se puede representar utilizando atributos o subelementos.



3.2. Bases de datos XML (eXtensible Markup Language)

Por ejemplo, el subelemento < tipo > de una asignatura, podría representarse como un atributo del elemento asignatura. El listado 3.2 representa la asignatura de Cálculo del ejemplo anterior, incluyendo el título de la asignatura como atributo de la misma.

Listado 3.2: Uso de atributos para la definición de asignaturas

```
1 <Asignaturas_Primer>
2   <Asignatura>
3     <titulo> Cálculo tipo = 'anual' </titulo>
4     <tipo> Anual </tipo>
5     <profesor>
6       <nombre> Ana </nombre>
7       <apellido> Sanz </apellido>
8     </profesor>
9     <profesor>
10      <nombre> Roberto </nombre>
11      <apellido> Hernández </apellido>
12    </profesor>
13  </Asignaturas_Primer>
```



3.2. Bases de datos XML (eXtensible Markup Language)

Dado que una de las principales aplicaciones de XML es el intercambio de información entre organizaciones, y ante la flexibilidad del lenguaje que permite definir cualquier nombre para una etiqueta, es posible que existan conflictos entre los nombres de éstas. Para ello, se recurre a los espacios de nombres, los cuales permiten a las organizaciones especificar nombres de etiquetas únicos. De esta forma, se añade al nombre de la etiqueta o atributo un identificador de recursos universal.



3.2. Bases de datos XML (eXtensible Markup Language)

El listado muestra de forma resumida un ejemplo de utilización del espacio de nombres para la definición de asignaturas.

Listado 3.3: Uso de un espacio de nombres

```
1 <Asignatura xmlns:AS="http://www.listado-asignaturas.com">
2   <AS:titulo> Cálculo tipo = 'anual' </AS:titulo>
3   <AS:tipo> Anual </AS:tipo>
4   <AS:profesor>
5     <AS:nombre> Ana </AS:nombre>
6     <AS:apellido> Sanz </AS:apellido>
7   </AS:profesor>
8   <AS:profesor>
9     <AS:nombre> Roberto </AS:nombre>
10    <AS:apellido> Hernández </AS:apellido>
11  </AS:profesor>
12 </Asignatura>
```



3.2.1. Definition Type Document (DTD)

Se dice que un documento XML es válido cuando está bien formado y cumple con la estructura de documento que ha sido dada a través de un lenguaje de definición de datos. Esta definición de la estructura de los documentos XML es opcional, si bien es cierto que es recomendable. **La definición de tipos de documento o DTD permite definir la estructura de un documento XML. Para ello, la DTD especifica la estructura de los elementos y atributos de un documento: qué elementos pueden aparecer, qué atributos, qué subelementos, multiplicidad de ellos...**

Existen multitud de validadores on-line de documentos XML y sus correspondientes DTDs y/o XML Schemas. <https://www.xmlvalidation.com> - <http://xmlvalidator.new-studio.org>



3.2.1. Definition Type Document (DTD)

En segundo lugar, la DTD puede declararse de forma externa, definiendo la misma en un fichero .DTD aparte que después es enlazado en el documento. El listado 3.5 muestra un ejemplo de declaración externa de una DTD.

Siguiendo con el ejemplo del listado 3.1, el listado 3.6 muestra el archivo .dtd que validaría la estructura del documento de asignaturas.

Listado 3.5: Declaración Externa DTD

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
2 <!DOCTYPE Elemento_Raiz SYSTEM "Mi_DTD.dtd">
3 <!-------Aquí va el documento XML----->
```

Listado 3.6: DTD Listado de Asignaturas

```
1 <!DOCTYPE Listado_Asignaturas[
2 <!-- ELEMENT Asignaturas_Primer (Asignatura+)>
3 <!-- ELEMENT Asignatura (titulo, tipo, profesor+)>
4 <!-- ELEMENT titulo (#PCDATA)>
5 <!-- ELEMENT tipo (#PCDATA)>
6 <!-- ELEMENT profesor (nombre, apellido)>
7 <!-- ELEMENT nombre (#PCDATA)>
8 <!-- ELEMENT apellido (#PCDATA)>
9 ]>
```


3.2.1. Definition Type Document (DTD)

Tabla 3.1: Descripción DTD de elementos y atributos XML

ELEMENTO			ATRIBUTO		
Tipos	#PCDATA	Cadena de caracteres	Tipos	CDATA	Cadena de caracteres
	EMPTY	Elemento sin contexto		ID	Identificador
	ANY	Cualquier tipo		IDREF	Referencia a un ID
		Disyunción		IDREFS	Referencia a múltiples ID.
Operadores	+	Uno o más	Valores	#REQUIRED	Obligatorio
	*	Cero o más		#IMPLIED	Opcional
	?	Cero o uno		#FIXED	Valor fijo
				#DEFAULT	Valor por defecto

3.2.1. Definition Type Document (DTD)

Tabla 3.1: Descripción DTD de elementos y atributos XML

ELEMENTO			ATRIBUTO		
Tipos	#PCDATA	Cadena de caracteres	Tipos	CDATA	Cadena de caracteres
	EMPTY	Elemento sin contexto		ID	Identificador
	ANY	Cualquier tipo		IDREF	Referencia a un ID
		Disyunción		IDREFS	Referencia a múltiples ID.
Operadores	+	Uno o más	Valores	#REQUIRED	Obligatorio
	*	Cero o más		#IMPLIED	Opcional
	?	Cero o uno		#FIXED	Valor fijo
				#DEFAULT	Valor por defecto



3.2.2. XML Schema

Se trata de un lenguaje de definición de la estructura de documentos XML más sofisticado que DTD. Aunque es más complejo, soluciona muchos de los inconvenientes de las DTD. De hecho, XML Schema es un superconjunto del lenguaje DTD especificado mediante la sintaxis de XML. Entre otros aspectos, XML Schema soporta no solo el tipo cadena de caracteres (soportado por DTD), sino también tipos numéricos, tipos complejos como listas y además permite al usuario definir sus propios tipos así como extender tipos de datos complejos mediante un mecanismo similar a la herencia. Además, XML Schema soporta el concepto de espacios de nombre, permitiendo reutilizar elementos de un esquema en otros.



3.2.2. XML Schema

Un XML Schema se especifica en un documento aparte del documento XML con extensión .xsd. Dentro de este archivo, que sigue las reglas sintácticas de cualquier documento XML, se especifican las definiciones de cada elemento del esquema, qué atributos incluye así como sus tipos de datos válidos asociados tanto a los elementos como a los atributos. De esta forma, XML Schema permite definir un esquema para validar documentos XML, de forma más sofisticada a como lo hace DTD. El listado 3.8 muestra un XML Schema para el ejemplo del listado de asignaturas.

3.2.2. XML Schema

Listado 3.8: XML Schema: Listado de asignaturas

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="
    http://www.w3.org/2001/XMLSchema">
3   <xs:element name="Asignaturas_Primer">
4     <xs:complexType>
5       <xs:sequence>
6         <xs:element maxOccurs="unbounded" name="Asignatura">
7           <xs:complexType>
8             <xs:sequence>
9               <xs:element name="titulo" type="xs:string" />
10              <xs:element name="tipo" type="xs:string" />
11              <xs:element maxOccurs="unbounded" name="profesor">
12                <xs:complexType>
13                  <xs:sequence>
14                    <xs:element name="nombre" type="xs:string" />
15                    <xs:element name="apellido" type="xs:string" />
16                  </xs:sequence>
17                </xs:complexType>
18              </xs:element>
19            </xs:sequence>
20          </xs:complexType>
21        </xs:element>
22      </xs:sequence>
23    </xs:complexType>
24  </xs:element>
25 </xs:schema>
```



3.2.2. XML Schema

XML Schema permite representar la cardinalidad de un elemento, definiendo el número mínimo y máximo de ocurrencias **mediante los atributos minOccurs y maxOccurs**. Además, también es posible añadir restricciones de forma que se limiten los valores de un elemento o atributo, se permita elegir entre una lista de posibles. Finalmente, XML Schema dispone de los contenedores sequence para definir una secuencia ordenada de subelementos, para definir la elección entre posibles grupos de elementos o elementos y all para definir un conjunto no ordenado de elementos. Al igual que sucedía con DTD, existen distintas herramientas on-line que no solo permiten validar un documento XML a partir de su esquema XSD, sino que también permiten generar el esquema XSD a partir del archivo XML.



3.2.3. XPath

En cualquier sistema de bases de datos, además de los lenguajes de definición de datos, los lenguajes de manipulación de datos son esenciales para poder realizar consultas y recuperar datos, realizar actualizaciones, etc.

XPath es un lenguaje de manipulación de datos XML. La principal característica de este lenguaje es que interpreta un documento XML como una estructura de datos de tipo árbol, donde los nodos y sus respectivas hojas se corresponden con los elementos y subelementos del documento XML. De esta forma, y utilizando comandos que permiten recorrer el documento de forma análoga a los comandos que se utilizan en una terminal de comandos para recorrer el sistema de archivos de un computador, es posible realizar consultas. La tabla 3.2 muestra un listado de los principales operadores utilizados en XPath.



3.2.3. XPath

Tabla 3.2: XPath: Operadores básicos

Operador	Significado
/	Navegación entre nodos del documento. Selecciona los nodos del nivel inferior
//	Navegación entre nodos del documento. Selecciona los nodos del nivel inferior de cualquier nodo especificado a continuación
.	Selecciona el nodo actual
..	Selecciona el nodo padre o nodo de nivel superior
	Expresa alternativas
@	Selecciona un atributo de un nodo
*	Selecciona todos los nodos
[]	Agrupar otros operadores

3.2.3. XPath



Consulta 1. Recuperar los nombres de todas las asignaturas

```
/Asignaturas_Primerο/Asignatura/titulo/text()
```



Consulta 2. ¿Cuántas asignaturas son impartidas por más de un profesor?

```
/Asignaturas_Primerο/Asignatura[count(profesor)>1]
```



Consulta 3. ¿Quiénes son los profesores que imparten la asignatura de física?

```
/Asignaturas_Primerο/Asignatura[titulo="Física"]/profesor/nombre
```



3.2.3. XPath

Es posible utilizar XPath de forma on-line a través de visualizadores o, por ejemplo, instalando el plug-in XPatherizer de Notepad++, en caso de utilizar este último editor de código XML.



3.2.4. XSLT

El lenguaje XSLT (XML Stylesheet Transformations) proviene del lenguaje XSL (XML Stylesheet). Este último permite especificar las opciones de formato de un documento XML en un archivo separado, aislando así la especificación del contenido y la presentación de un archivo XML. Podría decirse que XSL es a XML lo que CSS es a HTML. El lenguaje XSLT permite especificar las opciones de formato de un documento XML, pudiendo transformar este documento en otros formatos como HTML, PDF, etc. Dada la potencia de este lenguaje, que es capaz de transformar un archivo XML en otro que también pudiera ser XML, se utiliza frecuentemente como lenguaje de consulta.




3.2.4. XSLT

Las transformaciones XSLT se definen por medio de plantillas, que permiten a su vez recuperar contenido del documento XML a través de la utilización de expresiones XPath. La principal característica de XSLT es que la aplicación de plantillas se realiza de forma recursiva, lo cual recibe el nombre de recursividad estructural. El listado 3.9 presenta un ejemplo de aplicación de una transformación XSLT por medio de una plantilla.

Listado 3.9: XSLT: Ejemplo de plantilla

```
1 <xsl: template match = /Asignaturas_Primer/Asignatura >
2 <titulo_asignatura>
3   <xsl:value-of select = titulo/>
4 </titulo_asignatura>
5 </xsl:template>
```



3.2.4. XSLT

Tabla 3.3: Principales Constructores XSLT

Constructor XSLT	Significado
<code>< xsl : template match = "expresion X Path" ></code>	Selecciona nodos a devolver
<code>< xsl : value - of select = "valor" ></code>	Devuelve valores de los nodos seleccionados
<code>< /xsl : template >< xsl : template match = "*" / ></code>	Devuelve todos los nodos que no coinciden con alguna otra plantilla
<code>< xsl : attribute nombre = "nombre atributo" ></code>	Añade un atributo al elemento precedente
<code>< xsl : apply - templates / ></code>	Aplica la recursividad estructural
<code>< xsl : keyname = "nombre clave" match = "elementos aplicar" use = "valor clave" / ></code>	Define una clave
<code>< xsl : value - of select = "key("nombre clave" "valor")"</code>	Devuelve los nodos que coincidan con el valor especificado. Especifica operaciones de combinación (JOINS)
<code>< xsl : sort select = "elemento o atributo" / ></code>	Devuelve los nodos resultados, ordenados por un elemento o atributo



3.2.5. XQuery

Es un lenguaje de consulta de propósito general sobre datos XML estandarizado por el consorcio W3C. XQuery procede del lenguaje de programación Quilt y, por este motivo, toma prestadas muchas características de otros lenguajes como XPath o SQL. De hecho, la sintaxis de XQuery es mucho más similar a la de SQL, al contrario que ocurría con lenguajes como XPath o XSLT.



3.2.5. XQuery

Para definir una consulta en XQuery se construye una expresión, denominada “FLWR”(flower, flor en inglés). A continuación, se describen cada una de las sentencias de las que se compone la expresión:

FOR: Obtiene una serie de variables cuyos valores son el resultado de una expresión XPath. Es el equivalente a la cláusula FROM del lenguaje SQL.

LET: Define y renombra expresiones complejas para ser utilizadas en el resto de la consulta, reduciendo la complejidad y aliviando la sintaxis. Es una cláusula opcional.

WHERE: En esta cláusula, llamada igualmente en SQL, se especifican condiciones sobre los resultados obtenidos en la cláusula FOR.

RETURN: Especifica y define el resultado que se va a obtener de la consulta. Se trata del equivalente a la cláusula SELECT en SQL.



3.2.5. XQuery

Siguiendo la sintaxis anteriormente mencionada, el listado 3.10 muestra un ejemplo para obtener un listado con las asignaturas anuales.

Listado 3.10: XQuery: Ejemplo de consulta

```
1 for $x in /Asignaturas_Primer/Asignatura
2 let $nombre_asignatura:= $x/titulo
3 where $x/tipo='anual'
4 return <asignatura_anual> $nombre_asignatura </asignatura_anual>
```

En el listado anterior, **la cláusula for permitirá recorrer todos los elementos asignatura**. Por su parte, **let define una expresión en la que se asigna a nombre_asignatura el título de la asignatura que se está recorriendo en cada momento**. La cláusula **where impone el criterio de que el tipo de asignatura sea anual** y, finalmente, **result define como resultado un fragmento de código XML donde el nombre de la asignatura anual (según la expresión indicada en let) aparecerá como resultado encerrado entre dos etiquetas XML llamadas asignatura_anual**. Tal y como se puede apreciar, XQuery también permite generar nuevos documentos XML a partir de consultas.



3.3 Bases de datos documentales: MongoDB

Los sistemas de bases de datos documentales u orientados a documentos son sistemas de bases de datos NoSQL que almacenan datos, los cuales se estructuran en forma de documentos. En los últimos años, han proliferado una gran variedad de sistemas de este tipo como pueden ser Cassandra, CouchDB, Riak o MongoDB, sobre el cual trata esta sección.

MongoDB es un sistema de base de datos NoSQL, open source, orientado a documentos y escrito en lenguaje C++. Este sistema de bases de datos está disponible no solo para múltiples plataformas y sistemas operativos (Windows, Linux, OS X) sino también como servicio empresarial en la nube y se puede integrar con otros servicios como, por ejemplo, Amazon Web Services (AWS).



3.3.1 MongoDB: Características y aplicaciones

MongoDB es un sistema de bases de datos documental u orientado a documentos. Esta orientación hace que los datos se almacenen de forma estructurada en forma de documentos, los cuales se acoplan sin problema en los tipos de datos utilizados por los lenguajes de programación. Además, esta concepción hace que una base de datos MongoDB disponga de un esquema dinámico y fácilmente modificable.

Casos de uso (aplicaciones): almacenamiento y registro de eventos, comercio electrónico, juegos, aplicaciones móviles, almacenes de datos, gestión de estadísticas en tiempo real y cualquier aplicación que requiera llevar a cabo analíticas sobre grandes volúmenes de datos.



3.3.1 MongoDB: Características y aplicaciones

Principales características de MongoDB:

- **Alto rendimiento:** Gracias a la definición de los documentos y la creación de índices que hace que las lecturas y escrituras se realicen de forma más rápida.
- **Alta disponibilidad:** MongoDB dispone de servidores replicados con restablecimiento automático maestro.
- **Fácil escalabilidad:** Permitiendo distribuir colecciones de documentos entre diferentes máquinas de forma muy sencilla.
- **Indexación:** Similar al concepto de índice en una base de datos relacional, permite crear índices e índices secundarios para mejorar el rendimiento de las consultas.



3.3.1 MongoDB: Características y aplicaciones

Principales características de MongoDB:

- **Consultas ad-hoc:** Al igual que en una base de datos relacional, MongoDB da soporte a la búsqueda por campos, consultas de rangos, uso de expresiones regulares... A todo lo anterior, se añade la posibilidad de ejecutar y devolver una función JavaScript definida por el programador.
- **Replicación:** Siguiendo el modelo maestro-esclavo. El maestro puede ejecutar comandos de lectura y escritura mientras que el esclavo sólo tiene acceso de lectura y la posibilidad de realizar copias de seguridad. En caso de que el maestro caiga, el esclavo puede elegir un nuevo maestro para mantener el servicio de replicación.



3.3.2 MongoDB: Conceptos básicos, utilidades y herramientas.

En un servidor MongoDB es posible crear tantas bases de datos como se desee. El concepto de base de datos es en este caso equivalente al de base de datos en los sistemas relacionales.

Una vez creada, la base de datos estará compuesta por una o más colecciones. El término colección en el ámbito NoSQL es el equivalente al concepto de tabla en los sistemas relacionales. Cada colección, por tanto, estará formada por un conjunto de documentos (o incluso ninguno, en cuyo caso la colección estaría vacía). El concepto de documento en NoSQL se corresponde con el concepto de registro en una base de datos relacional. Un documento estará compuesto por una serie de campos, al igual que lo están los registros de una base de datos relacional. En el ámbito NoSQL, y más concretamente en MongoDB, cada documento viene dado por un archivo JSON (<http://www.json.org/>) en el cual, siguiendo una estructura clave-valor se especifican las características de cada documento.



3.3.2 MongoDB: Conceptos básicos, utilidades y herramientas.

El listado 3.11 muestra un ejemplo de documento que define un barco:

Listado 3.11: Definición json de un documento barco

```
1 {  
2   name: 'USS Enterprise-D',  
3   operator: 'Starfleet',  
4   type: 'Explorer',  
5   class: 'Galaxy',  
6   crew: 750,  
7   codes: [10,11,12]  
8 }
```



3.3.2 MongoDB: Conceptos básicos, utilidades y herramientas.

Para la administración del sistema de bases de datos, MongoDB pone a disposición de los usuarios las siguientes utilidades:

- **mongo:** Se trata del shell interactivo de MongoDB que permite insertar, eliminar, actualizar datos y realizar consultas, además de replicar la información, apagar servidores y ejecutar código JavaScript.
- **mongostat:** Herramienta de línea de comandos que muestra las estadísticas de una instancia en ejecución de MongoDB
- **mongotop:** Herramienta de línea de comandos que muestra la cantidad de tiempo empleado en la lectura y escritura de datos por parte de la instancia en ejecución



3.3.2 MongoDB: Conceptos básicos, utilidades y herramientas.

Para la administración del sistema de bases de datos, MongoDB pone a disposición de los usuarios las siguientes utilidades:

- **mongosniff:** Herramienta de línea de comandos que permite hacer un rastreo del tráfico de la red que va desde y hacia MongoDB.
- **mongoimport/mongoexport:** Herramienta de línea de comandos para importar y exportar contenido en o desde .json, .csv o .tsv entre otros.
- **mongodump/mongorestore:** Herramienta de línea de comandos para la creación de una exportación binaria del contenido de la base de datos.

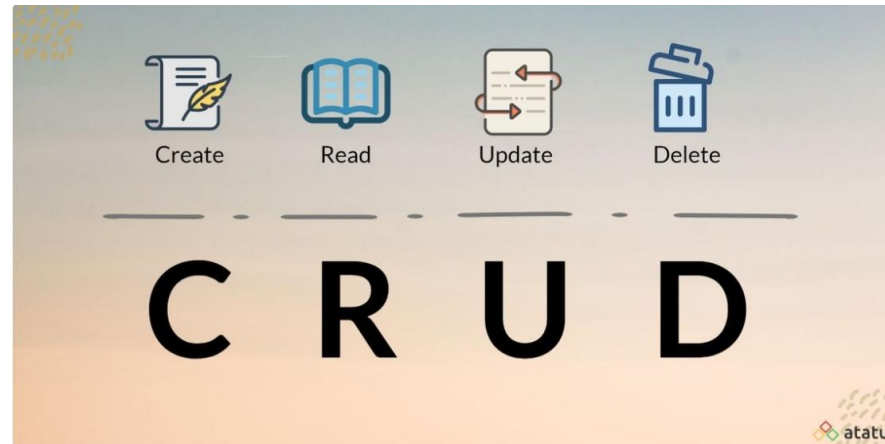


3.3.2 MongoDB: Conceptos básicos, utilidades y herramientas.

Aunque MongoDB tiene una interfaz administrativa accesible desde <http://localhost:28017/> (siempre que se haya iniciado con la opción *mongod - - rest*, existen algunas herramientas gráficas para la administración y uso de este sistema de base de datos como UMongo (<https://github.com/agirbal/umongo>) , el cual es una aplicación open source de sobremesa para navegar y administrar un clúster MongoDB o Robomongo (<https://robomongo.org>), que también es una herramienta gráfica open source que incluye una terminal de comandos completamente compatible con el shell de mongo.

3.3.3 MongoDB: Operaciones CRUD

Cualquier sistema de datos permite definir operaciones básicas como son la creación de tablas e inserción de registros, lectura de datos, actualización y eliminación de registros. Estas operaciones básicas se conocen con el nombre de CRUD (Create, Read, Update, Delete).





3.3.3 MongoDB: Operaciones CRUD

Una vez iniciada una sesión del Shell de Mongo, es posible crear una base de datos utilizando el siguiente comando.

```
test>use DB_Ejemplo  
  
switched to db DB_Ejemplo
```

Para crear una colección de documentos “Barco” dentro de la base de datos DB_Ejemplo se puede escribir.

```
DB_Ejemplo>db.createCollection(“ships”)  
  
{ ok: 1 }
```



3.3.3 MongoDB: Operaciones CRUD

Para visualizar las bases de datos almacenadas en el servidor se puede utilizar el comando *show dbs* mientras que para obtener un listado de las colecciones de la base de datos sobre la que se está trabajando, es posible utilizar el comando *show collections*. Por otra parte, para insertar un documento dentro de la colección, como el mostrado en el listado [3.11](#), se ejecuta el siguiente comando.

```
db.ships.insert({ name:'USS-Enterprise-D',operator:'Starfleet',type:'Explorer',class:'Galaxy',crew:750,codes:[10,11,12]})
```

Listado 3.11: Definición json de un documento barco

```
1 {  
2   name:'USS Enterprise-D',  
3   operator:'Starfleet',  
4   type:'Explorer',  
5   class:'Galaxy',  
6   crew:750,  
7   codes:[10,11,12]  
8 }
```



3.3.3 MongoDB: Operaciones CRUD

Para actualizar el nombre del barco “USS Prometheus” por “USS Something” es posible escribir el comando.

```
db.ships.update({name : 'USS Prometheus'}, {name : 'USS Something'})
```

Mientras que si se pretenden establecer o cambiar varios atributos de un mismo documento, como por ejemplo el operador y la clase, se puede utilizar el comando *update* de la siguiente forma.

```
db.ships.update({name : 'USS Something'}, {$set : {operator : 'Starfleet', class : 'Prometheus' }})
```



3.3.3 MongoDB: Operaciones CRUD

Finalmente, la eliminación de algún atributo de un documento también es una operación de actualización que puede realizarse de la siguiente forma.

```
db.ships.update({name : 'USS Something' }, { $unset : {operator : 1} })
```

La eliminación de documentos de una colección puede realizarse de forma directa o bien utilizando expresiones regulares. A continuación se muestran dos comandos para ilustrar ambas formas de eliminación. El segundo de ellos elimina aquellos documentos que comienzan por “a”.

```
db.ships.remove({ name : 'USS Prometheus' })
```

```
db.ships.remove({ name:{$regex:'A*'}})
```



3.3.3 MongoDB: Operaciones CRUD

Para leer y mostrar documentos, se utiliza el comando *find()*. A continuación se muestra un ejemplo en el que, el primer comando muestra un documento al azar de los existentes, el segundo muestra todos los documentos y lo hace de forma indexada en lugar de como texto seguido, el tercero muestra solo los nombres de los barcos y el último, encuentra un documento cuyo nombre sea “USS Defiant”.

```
db.ships.findOne()
```

```
db.ships.find().prettyPrint()
```

```
db.ships.find({ }, { name:true })
```

```
db.ships.findOne({ 'name': 'USS Defiant' })
```



3.3.3 MongoDB: Consultas y Agregación

A continuación, se muestran dos consultas que recuperan aquellos barcos que permitan subir a bordo a más de cien pasajeros y otra consulta para recuperar aquellos que dejen subir tan solo a 100 pasajeros o menos.

```
db.ships.find({crew:{ $gt:100}})
```

```
db.ships.find({crew:{ $lte:100}})
```

El operador `$exists` permite devolver aquellos documentos para los que existe o no un determinado atributo. El siguiente comando permite encontrar aquellos barcos para los cuales existe el campo “class”.

```
db.ships.find({class:{ $exists:true}})
```




3.3.3 MongoDB: Consultas y Agregación

Las funciones de agregación son especialmente útiles en los lenguajes de consulta y manipulación de datos. De esta forma, `$sum` permite agregar mediante la operación suma una serie de valores, `$avg` permite obtener la media, `$min` y `$max` encontrar el valor máximo y mínimo de un atributo para el conjunto de elementos, `$push` introduce en un array los resultados de la consulta que se ha realizado, `$addToSet` es similar al anterior solo que sin incluir valores duplicados y, por último, `$first` y `$last` permiten obtener el primer y último documento en una consulta.



3.3.3 MongoDB: Consultas y Agregación

A continuación, se muestra un ejemplo del uso de cada uno de estos operadores de agregación.

```
db.ships.aggregate([{$group: {_id: "$operator", num_ships: {$sum: "$crew"}}}])  
  
db.ships.aggregate([{$group : {_id : "$operator", num_ships : {$avg : "$crew"}}}])  
  
db.ships.aggregate([{$group : {_id : "$operator", num_ships : {$min : "$crew"}}}])  
  
db.ships.aggregate([{$group : {_id : "$operator", classes : {$push: "$class"}}}])  
  
db.ships.aggregate([{$group : {_id : "$operator", classes : {$addToSet :  
"$class"}}}])  
  
db.ships.aggregate([{$group: {_id: "$operator", last_class: {$last: "$class"}}}])
```



3.3.3 MongoDB: Consultas y Agregación

Finalmente, MongoDB pone a disposición de los usuarios multitud de funciones útiles en la realización de consultas. La tabla [3.4](#) muestra algunas de las más utilizadas.

Tabla 3.4: Funciones útiles en MongoDB

Función	Significado
\$project	Cambia el conjunto de documentos modificando sus claves y valores
\$match	Operación de filtrado para reducir el número de elementos recuperados
\$group	Operador de agregación para agrupar resultados
\$sort	Ordena los documentos
\$skip	Recupera los documentos a partir de un numero especificado por el usuario
\$limit	Limita los resultados de la consulta al valor pasado como parámetro a la función
\$unwind	Utilizado como equivalente al join de SQL



Big Data Aplicado

Curso de especialización en Inteligencia Artificial y Big Data.



4. Gestión de Soluciones

4.1 Bases de datos orientadas a grafos

Una base de datos orientada a grafos **es un tipo de base de datos NoSQL** como lo son también, por ejemplo, las bases de datos documentales. Un sistema de gestión de bases de datos orientadas a grafos **es aquel que define métodos y operaciones CRUD sobre un modelo de datos representado mediante un grafo**. Por norma general, **este tipo de bases de datos se implementan para su uso junto con sistemas OLTP**, por lo que están diseñadas en términos de relaciones de integridad y **optimizadas para ofrecer un rendimiento operacional**.

Un grafo es un conjunto de nodos o vértices, que representan entidades de un dominio, y un conjunto de aristas o enlaces, que representan las relaciones o interacciones que se producen entre las entidades

4. Gestión de Soluciones

4.1 Bases de datos orientadas a grafos

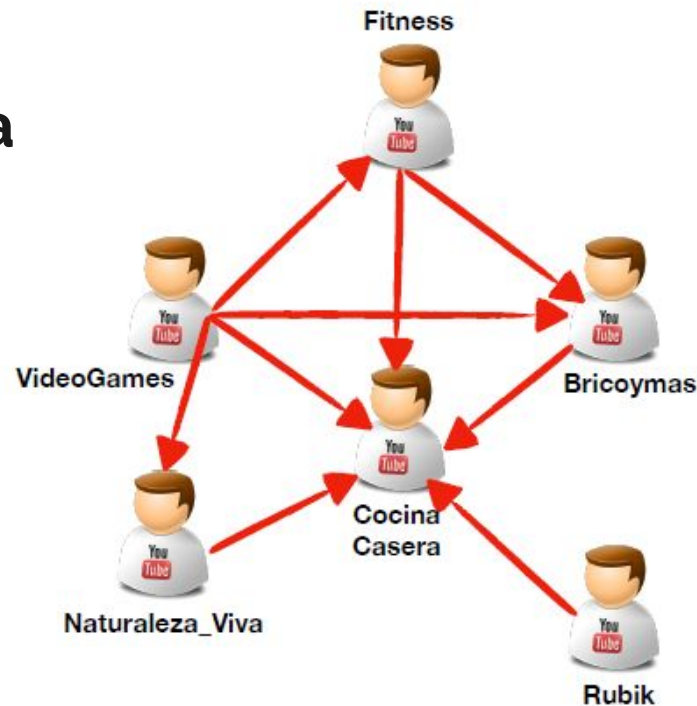


Figura 4.1: Ejemplo de grafo: relaciones de seguimiento entre canales YouTube



4. Gestión de Soluciones

4.1 Bases de datos orientadas a grafos

Uno de los modelos de datos basado en grafos más popular es el **modelo de grafo de propiedades etiquetadas**. Un grafo representado mediante este modelo, debe cumplir las siguientes características principales:

- 1) debe contener un conjunto de nodos y aristas o relaciones entre dichos nodos.
- 2) los nodos contienen propiedades, definidas a través de pares "clave-valor".
- 3) los nodos pueden estar etiquetados con una o más etiquetas.
- 4) las relaciones entre los nodos están nombradas y son dirigidas, teniendo siempre un nodo de inicio y otro nodo de fin.
- 5) las relaciones del grafo también pueden contener propiedades.

Este modelo es sencillo de entender y permite modelar cualquier problema y/o conjunto de datos.



4.2 Características principales

- **Almacenamiento:** Las bases de datos orientadas a grafo pueden ser de almacenamiento nativo, en cuyo caso los datos se almacenan internamente con estructura de grafo. No obstante, existen bases de datos de grafos que mapean los datos para adaptarlos a una estructura relacional.
- **Procesamiento:** El almacenamiento nativo permite obtener una estructura de datos que no requiere índices de adyacencia para referenciar los nodos del grafo, mejorando el rendimiento en la ejecución de consultas.
- **Rendimiento:** La ejecución de consultas sobre datos estructurados en forma de grafo proporciona escalabilidad al sistema de bases de datos, ya que aunque el número de datos aumente, las consultas se ejecutan sobre la porción del grafo de interés.



4.1.1 Características principales

- **Flexibilidad:** Los modelos de datos basados en grafos permiten rápidamente añadir nuevos nodos, enlaces y subgrafos a un modelo existente con facilidad y sin pérdida de funcionalidad ni rendimiento.
- **Rapidez:** La naturaleza libre de esquema de las bases de datos orientadas a grafos, junto con las APIs desarrolladas para su manipulación y los lenguajes de consulta, hacen que estos sistemas sean ágiles y rápidos, integrándolos con las metodologías de desarrollo de software iterativas e incrementales.



4.1.2 Áreas de aplicación

Redes sociales: El análisis de redes sociales mediante bases de datos orientadas a grafos permite identificar relaciones explícitas e implícitas entre usuarios y grupos de usuarios, así como identificar la forma en la que ellos interactúan pudiendo inferir el comportamiento de un usuario en base a sus conexiones. En una red social, dos usuarios presentan una conexión explícita si están directamente conectados. Por otra parte, una relación implícita es aquella que se produce entre dos usuarios a través de un intermediario, como puede ser otro usuario, un post donde han hecho un comentario, un “like” o un artículo que ambos han comprado.



4.1.2 Áreas de aplicación

Sistemas de recomendación: Estos sistemas permiten modelar en forma de grafo las relaciones que se establecen entre personas o usuarios y cosas, como pueden ser productos, servicios, contenido multimedia o cualquier otro concepto relevante en función del dominio de aplicación. Las relaciones se establecen en función del comportamiento de los usuarios al comprar, consumir contenido, puntuarlo o evaluarlo etc. De esta forma, los sistemas de recomendación identifican recursos de interés para un usuario específico y pueden predecir su comportamiento al comprar un producto o contratar un servicio.



4.1.2 Áreas de aplicación

Información geográfica: Se trata del caso de aplicación más popular de la teoría de grafos. Las aplicaciones de las bases de datos orientadas a grafos en información geográfica van desde calcular rutas óptimas entre dos puntos en cualquier tipo de red (red de carreteras, de ferrocarril, aérea, logística...) hasta encontrar todos los puntos de interés en un área concreta, encontrar el centro de una región u obtener la intersección entre dos o más regiones, entre otras muchas. Así, las bases de datos orientadas a grafos permiten modelar estos casos de aplicación como grafos dirigidos sobre los cuales operar para obtener los resultados deseados.



4.2 Neo4j

Neo4j es una plataforma de bases de datos orientada a grafos lanzada en 2007 e implementada en Java. Entre sus características principales destacan el almacenamiento y procesamiento de grafos nativo, lo que la convierte en una alternativa potente para el trabajo en entornos reales, donde se requiere de estas características para mejorar el rendimiento y la escalabilidad de los sistemas. Además, a pesar de lo anterior, Neo4j también permite trabajar con modelos de datos en forma de grafo de manera transaccional, por lo que también es útil en sistemas transaccionales.

Neo4j **dispone de amplias librerías que permiten crear también modelos de aprendizaje automático sobre grafos.** Esta plataforma puede integrarse con múltiples lenguajes de programación como Python o C++, entre otros, y ofrece una serie de herramientas muy útiles para usuarios y desarrolladores.



4.2 Neo4j. Librerías

Graph Data Science Library: Se trata de la librería principal que contiene una gran cantidad de algoritmos de búsqueda sobre grafos además de modelos de aprendizaje automático. Se trata de algoritmos y modelos eficientemente programados y escalables para el trabajo con grafos.

Neo4j Bloom: Es una aplicación de visualización y exploración de grafos que permite la visualización de estos desde distintas perspectivas y puntos de vista, ofreciendo así una herramienta visual muy útil para visualizar los resultados de los algoritmos así como mostrar resultados a clientes.

Cypher: Es el lenguaje de creación de consultas utilizado en Neo4j. Se trata de un lenguaje inspirado en SQL, solo que más sencillo y optimizado para el trabajo con grafos.



4.2 Neo4j

Integradores: Con el objetivo de integrar y conectar Neo4j con otras plataformas y lenguajes, se han desarrollado distintos conectores para integrar esta plataforma con tecnologías como Apache Spark, Apache Kafka, Bl...

Herramientas para desarrolladores: Neo4j dispone de versiones para escritorio y web así como un sandbox en el que se proporciona un entorno controlado e integrado de los servicios Neo4j para desarrolladores.

Neo4j Aura: Neo4j ofrece un servicio de computación en la nube para la creación de grafos y ejecución de algoritmos y modelos sobre ellos. Se trata de Neo4j Aura, el cual se encuentra disponible de forma gratuita y está integrado con Google Cloud Platform.

4.2.2 Importación de datos. Creación de grafos

Para comenzar a trabajar con Neo4j, es necesario crear un grafo o bien partir de un grafo ya existente que se importa dentro del área de trabajo de Neo4j. A continuación, se muestra cómo implementar ambos procedimientos.

Importación de un grafo

Sea el caso en el que se pretende importar un grafo a partir de un fichero .csv. En el ejemplo que se muestra a continuación, se dispone de dos ficheros .csv que contienen, respectivamente, la definición de los nodos y de las aristas del grafo que se pretende importar. Así pues, el listado 4.1 muestra los nodos del grafo. Para cada nodo se almacena su identificador, latitud, longitud y población.

Listado 4.1: Definición de nodos de un grafo

1	id,latitude,longitude,population
2	"Amsterdam",52.379189,4.899431,821752
3	"Utrecht",52.092876,5.104480,334176
4	"Den Haag",52.078663,4.288788,514861
5	"Immingham",53.61239,-0.22219,9642
6	"Doncaster",53.52285,-1.13116,302400
7	"Hoek van Holland",51.9775,4.13333,9382
8	"Felixstowe",51.96375,1.3511,23689
9	"Ipswich",52.05917,1.15545,133384
10	"Colchester",51.88921,0.90421,104390
11	"London",51.509865,-0.118092,8787892
12	"Rotterdam",51.9225,4.47917,623652
13	"Gouda",52.01667,4.70833,70939

4.2.2 Importación de datos. Creación de grafos

Por su parte, el listado 4.2 especifica cada uno de los enlaces entre los nodos del grafo. Para cada uno de los enlaces se identifica el origen y destino del enlace, el tipo de relación que se define entre ellos y el coste.

Listado 4.2: Definición de aristas de un grafo

```
1  src,dst,relationship,cost
2  "Amsterdam","Utrecht","EROAD",46
3  "Amsterdam","Den Haag","EROAD",59
4  "Den Haag","Rotterdam","EROAD",26
5  "Amsterdam","Immingham","EROAD",369
6  "Immingham","Doncaster","EROAD",74
7  "Doncaster","London","EROAD",277
8  "Hoek van Holland","Den Haag","EROAD",27
9  "Felixstowe","Hoek van Holland","EROAD",207
10 "Ipswich","Felixstowe","EROAD",22
11 "Colchester","Ipswich","EROAD",32
12 "London","Colchester","EROAD",106
13 "Gouda","Rotterdam","EROAD",25
14 "Gouda","Utrecht","EROAD",35
15 "Den Haag","Gouda","EROAD",32
16 "Hoek van Holland","Rotterdam","EROAD",33
```



4.2.2 Importación de datos. Creación de grafos

Dados estos dos ficheros, es posible importar los nodos del grafo según se indica en el listado 4.3.

Listado 4.3: Importación de nodos

```
1 WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/transport-nodes.csv"
  AS uri
2 LOAD CSV WITH HEADERS FROM uri AS row
3 MERGE (place:Place {id:row.id})
4 SET place.latitude = toFloat(row.latitude),
5   place.longitude = toFloat(row.longitude),
6   place.population = toInteger(row.population)
```

En el listado anterior se define como identificador de recurso la dirección donde se encuentra el archivo que se pretende cargar, el cual es cargado a través de la instrucción LOAD CSV. La instrucción MERGE permite definir cada fila como nodo de un grafo de tipo lugar (place) y añadir a cada nodo la propiedad id cuyo valor será el identificador de la fila en la que se encuentra cada nodo dentro del fichero csv. Finalmente, los valores de latitud y longitud se convierten en valores de tipo real y la población en un valor de tipo entero.



4.2.2 Importación de datos. Creación de grafos

Análogamente a la importación de nodos, el listado 4.4 muestra el código necesario para la importación del fichero que contiene los enlaces entre los nodos del grafo.

Listado 4.4: Importación de aristas

```
1 WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/transport-relationships.csv" AS uri
2 LOAD CSV WITH HEADERS FROM uri AS row
3 MATCH (origin:Place {id: row.src})
4 MATCH (destination:Place {id: row.dst})
5 MERGE (origin)-[:EROAD {distance: toInteger(row.cost)}]->(destination)
```

La diferencia principal de este listado con respecto al anterior, es que en este se necesita identificar mediante la instrucción MATCH los nodos de origen y destino de cada uno de los enlaces, lo cual es posible mediante los atributos "src" y "dst" incluidos en el archivo .csv. Finalmente, se utiliza la sentencia MERGE para crear una relación entre "origin" y "destination" de tipo EROAD que contiene la propiedad "distance" cuyo valor es el coste que aparecía en el fichero .csv.

4.2.2 Importación de datos. Creación de grafos

Creación de un grafo: Otra alternativa es la creación de un grafo desde cero, especificando para ello sus nodos y enlaces. La figura 4.2 muestra un ejemplo de un grafo de coocurrencia de hashtags en el que cada uno de los nodos es un hashtag y los enlaces entre ellos se producen cuando dos hashtags aparecen en un mismo tweet. Sobre cada enlace, se puede apreciar un valor que indica en cuántos tweets concurrieron los hashtags conectados por medio de dicha arista.



Figura 4.2: Grafo de co-ocurrencia de hashtags

4.2.2 Importación de datos. Creación de grafos

Para representar este grafo en Neo4j, se puede escribir el siguiente fragmento de código que muestra el listado 4.5.

Listado 4.5: Creación grafo de co-ocurrencia de hashtags

```
1 CREATE
2     (JS:Hashtag {name: 'JoaquinSabina'}),
3     (RS:Hashtag {name: 'Rusia2018'}),
4     (AG:Hashtag {name: 'Argentina'}),
5     (FD:Hashtag {name: 'Feliz Domingo'}),
6     (MS:Hashtag {name: 'Messi'}),
7
8     (JS)-[:C00C {ntweet: 52}]->(FD),
9     (FD)-[:C00C {ntweet: 52}]->(JS),
10    (RS)-[:C00C {ntweet: 183}]->(FD),
11    (FD)-[:C00C {ntweet: 183}]->(RS),
12    (RS)-[:C00C {ntweet: 73}]->(AG),
13    (AG)-[:C00C {ntweet: 73}]->(RS),
14    (AG)-[:C00C {ntweet: 112}]->(MS),
15    (MS)-[:C00C {ntweet: 112}]->(AG),
16    (FD)-[:C00C {ntweet: 81}]->(MS),
17    (MS)-[:C00C {ntweet: 81}]->(FD)
```

4.2.2 Importación de datos. Creación de grafos

Tal y como se puede ver, en primer lugar se definen los nodos y sus propiedades y a continuación los enlaces y sus propiedades. Todo grafo en Neo4j tiene aristas dirigidas, si bien es cierto que cuando se ejecutan las consultas puede no tenerse en cuenta la dirección de las aristas. Por este motivo, se han especificado las aristas en ambas direcciones. En caso de querer crear una arista que sea interpretada como no dirigida, se debe crear sin el operador flecha que determina el sentido de la arista. Esto es posible hacerlo solo fuera de la instrucción CREATE.

Listado 4.5: Creación grafo de co-ocurrencia de hashtags

```
1 CREATE
2     (JS:Hashtag {name: 'JoaquinSabina'}),
3     (RS:Hashtag {name: 'Rusia2018'}),
4     (AG:Hashtag {name: 'Argentina'}),
5     (FD:Hashtag {name: 'Feliz Domingo'}),
6     (MS:Hashtag {name: 'Messi'}),
7
8     (JS)-[:C00C {ntweet: 52}]->(FD),
9     (FD)-[:C00C {ntweet: 52}]->(JS),
10    (RS)-[:C00C {ntweet: 183}]->(FD),
11    (FD)-[:C00C {ntweet: 183}]->(RS),
12    (RS)-[:C00C {ntweet: 73}]->(AG),
13    (AG)-[:C00C {ntweet: 73}]->(RS),
14    (AG)-[:C00C {ntweet: 112}]->(MS),
15    (MS)-[:C00C {ntweet: 112}]->(AG),
16    (FD)-[:C00C {ntweet: 81}]->(MS),
17    (MS)-[:C00C {ntweet: 81}]->(FD)
```



4.2.2 Importación de datos. Creación de grafos

El listado 4.6 muestra cómo crear la primera relación del grafo solo que de forma no dirigida

Listado 4.6: Creación de una arista no dirigida

```
1 MATCH(JS:Hashtag{name:'JoaquinSabina'})  
2 MATCH(FD:Hashtag{name:'Feliz Domingo'})  
3 MERGE (JS)-[:C00C{ntweet:52}]- (FD)
```



4.2.2 Importación de datos. Creación de grafos

Para **mostrar el grafo**, es posible utilizar el siguiente comando. La figura [4.3](#) muestra cómo quedaría el grafo creado en el listado [4.5](#).

```
MATCH(n) return (n)
```

Para **eliminar un nodo** creado, es posible utilizar el comando:

```
MATCH(JS:Hashtag{name:'Joaquín Sabina'}) delete (JS)
```




4.2.2 Importación de datos. Creación de grafos

Sin embargo, **un nodo solo puede ser eliminado si se eliminan las relaciones que contiene**. Para ello, es posible utilizar el comando:

```
MATCH(JS:Hashtag{name:'Joaquín Sabina'}) detach delete (JS)
```

Mientras que, **para eliminar todos los nodos y aristas**, es posible escribir:

```
MATCH(n) detach delete(n)
```

4.2.2 Importación de datos. Creación de grafos

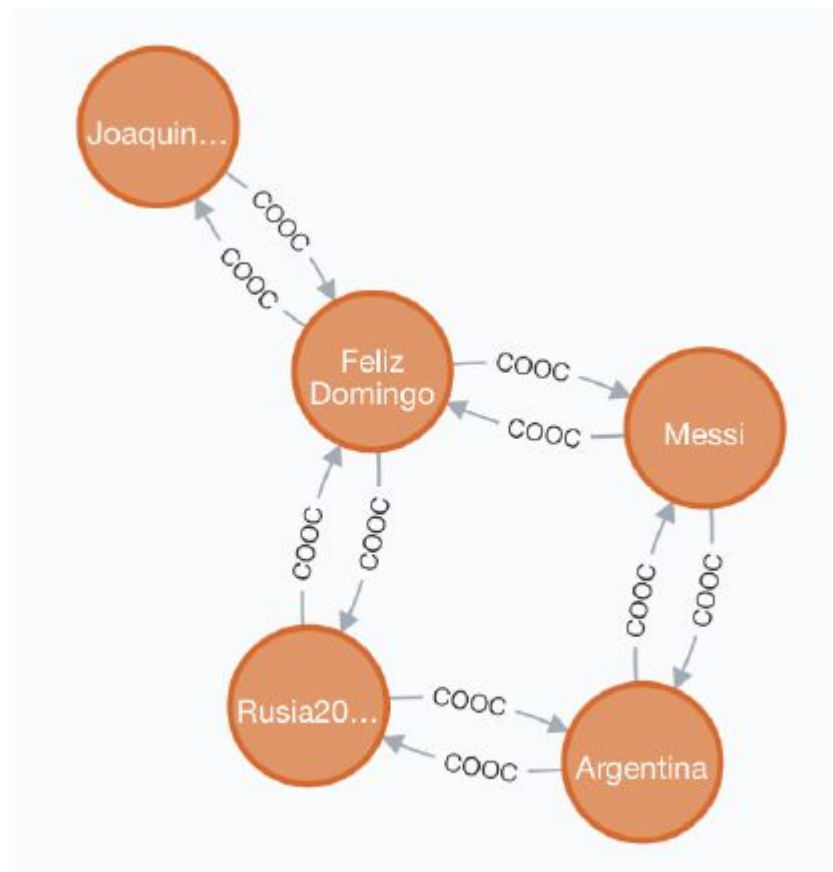


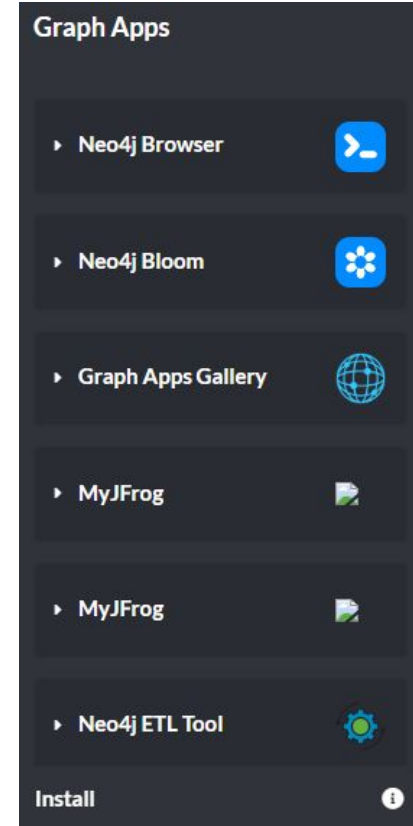
Figura 4.3: Visualización del grafo de co-ocurrencia de hashtags

4.2.2 Importación de datos. Creación de grafos

- Instalación del software Neo4j Desktop

<https://neo4j.com/download/>

- Verificar que tiene instalado las siguientes Apps
 - Neo4j Browser
 - Neo4j Bloom
 - Graph Apps Gallery
 - MyJFrog
 - Neo4j ETL Tool





4.2.2 Importación de datos. Creación de grafos

- Importación de grafo

```
WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base
WITH base + "social-nodes.csv" AS uri
LOAD CSV WITH HEADERS FROM uri AS row
MERGE (:User {id: row.id})
```

4.2.2 Importación de datos. Creación de grafos

- Importación de grafo

```
WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/" AS base
WITH base + "social-relationships.csv" AS uri
LOAD CSV WITH HEADERS FROM uri AS row
MATCH (source:User {id: row.src})
MATCH (destination:User {id: row.dst})
MERGE (source)-[:FOLLOWS]->(destination)
```



4.2.2 Importación de datos. Creación de grafos

- Añadir nueva conexión

```
MATCH (src {id: "David"})
```

```
MATCH (dst {id: "Mark"})
```

```
CREATE (src)-[:FOLLOWS]->(dst)
```



4.2.2 Importación de datos. Creación de grafos

- Borrar la conexión

```
MATCH (src {id: "David"})-[rel:FOLLOWS]->(dst {id: "Mark"})  
DELETE rel
```



4.2.2 Importación de datos. Creación de grafos

- Añadir nuevo nodo

```
CREATE (:User {id: "Kevin"})
```


4.2.2 Importación de datos. Creación de grafos

- Añadir lista de nodos → en lotes

```
UNWIND [  
  ["Amsterdam", 52.379189, 4.899431, 821752],  
  ["Utrecht", 52.092876, 5.104480, 334176],  
  ["Den Haag", 52.078663, 4.288788, 514861],  
  ["Immingham", 53.61239, -0.22219, 9642],  
  ["Doncaster", 53.52285, -1.13116, 302400],  
  ["Hoek van Holland", 51.9775, 4.13333, 9382],  
  ["Felixstowe", 51.96375, 1.3511, 23689],  
  ["Ipswich", 52.05917, 1.15545, 133384],  
  ["Colchester", 51.88921, 0.90421, 104390],  
  ["London", 51.509865, -0.118092, 8787892],  
  ["Rotterdam", 51.9225, 4.47917, 623652],  
  ["Gouda", 52.01667, 4.70833, 70939]  
] AS data  
CREATE (:City {id: data[0], latitude: data[1], longitude: data[2], population: data[3]})
```

4.2.2 Importación de datos. Creación de grafos

- Añadir relaciones → en lotes

```
UNWIND [  
  ["Amsterdam", "Utrecht", "EROAD", 46],  
  ["Amsterdam", "Den Haag", "EROAD", 59],  
  ["Den Haag", "Rotterdam", "EROAD", 26],  
  ["Amsterdam", "Immingham", "EROAD", 369],  
  ["Immingham", "Doncaster", "EROAD", 74],  
  ["Doncaster", "London", "EROAD", 277],  
  ["Hoek van Holland", "Den Haag", "EROAD", 27],  
  ["Felixstowe", "Hoek van Holland", "EROAD", 207],  
  ["Ipswich", "Felixstowe", "EROAD", 22],  
  ["Colchester", "Ipswich", "EROAD", 32],  
  ["London", "Colchester", "EROAD", 106],  
  ["Gouda", "Rotterdam", "EROAD", 25],  
  ["Gouda", "Utrecht", "EROAD", 35],  
  ["Den Haag", "Gouda", "EROAD", 32],  
  ["Hoek van Holland", "Rotterdam", "EROAD", 33]  
] AS data  
MATCH (src:City {id: data[0]}), (dst:City {id: data[1]})  
CREATE (src)-[:EROAD {cost: (data[3])}]->(dst)
```



4.2.3 Recorridos sobre grafos

Los recorridos sobre grafos son una utilidad imprescindible en el trabajo con esta estructura de datos. Del mismo modo, los recorridos también son una herramienta fundamental cuando se trabaja con bases de datos orientadas a grafos.

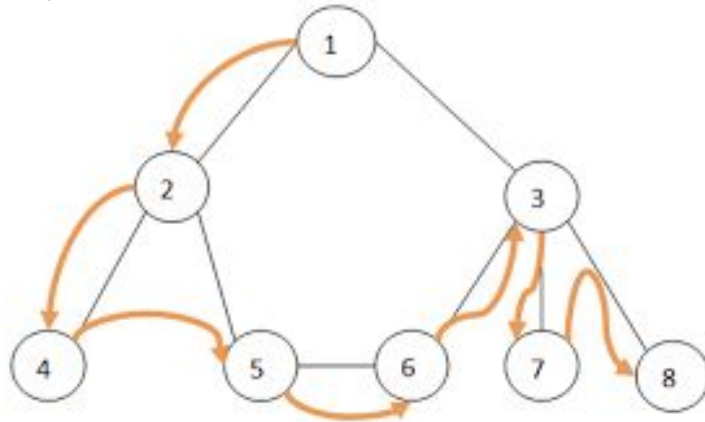
Un recorrido sobre grafos es un procedimiento sistemático que permite explorar un grafo examinando todos sus vértices y aristas, comenzando desde un vértice inicial. A la hora de recorrer un grafo, existen dos tipos de recorridos:

- **el recorrido en anchura (denotado BFS, por sus siglas en inglés) y**
- **el recorrido en profundidad (denotado DFS, también por sus siglas en inglés).**

4.2.3 Recorridos sobre grafos

Recorrido en anchura (BFS)

El recorrido en anchura, también conocido por sus siglas en inglés Breadth First Search (BFS) es un procedimiento que permite recorrer un grafo, explorando todos sus nodos y aristas. A menudo se utiliza para buscar el camino más corto entre dos nodos en un grafo no ponderado.



El resultado del recorrido es: Doncaster, London, Colchester, Ipswich, Felixstowe, Hoek van Holland, Den Haag, Rotterdam, Gouda, Utrecht.



4.2.3 Recorridos sobre grafos

Recorrido en profundidad (DFS)

El recorrido en profundidad, también conocido por sus siglas en inglés Depth First Search (DFS) es un procedimiento alternativo al recorrido en anchura para explorar un grafo, recorriendo todos sus nodos y aristas. El procedimiento para recorrer un grafo mediante el método DFS es el siguiente: en primer lugar, se parte de un nodo inicial. A partir de él, se empiezan a recorrer todos sus hijos hasta el último nivel. Una vez llegados al último nivel del grafo, se vuelve hacia atrás recorriendo todos los hijos de los nodos de niveles anteriores.

Es útil para encontrar todas las soluciones posibles en un grafo y puede utilizarse en la búsqueda de árboles y grafos.



4.2.3 Recorridos sobre grafos

Recorrido en profundidad (DFS)

```
CALL gds.graph.create('myGraph', 'City', 'EROAD', {relationshipProperties: 'cost'})
YIELD graphName, nodeCount, relationshipCount, createMillis;

MATCH (a:Place {id: 'Doncaster'})
WITH id(a) AS startNode
CALL gds.alpha dfs.stream('myGraph', {startNode: startNode})
YIELD path
UNWIND [n in nodes(path) | n.id] AS tags
RETURN tags;
```

El resultado obtenido es el mismo pero, en esta ocasión, hemos utilizado otro método de recorrido: Doncaster, London, Colchester, Ipswich, Felixstowe, Hoek van Holland, Rotterdam, Den Haag, Gouda, Utrecht.



4.2.4 Caminos mínimos

El cálculo y la obtención de caminos mínimos dentro de un grafo es uno de los problemas clásicos en teoría de grafos con multitud de aplicaciones en el mundo real.

En el caso de los grafos no pesados, es decir, aquellos cuyas aristas no tienen coste, el cálculo del camino mínimo se obtiene minimizando el número de saltos entre el nodo origen y el nodo destino. En el caso de los grafos pesados, aquellos cuyas aristas tienen coste, el cálculo del camino mínimo se obtiene a través del camino en el que la suma de los costes de sus aristas es mínima.



4.2.4 Caminos mínimos

El cálculo de caminos mínimos también es muy útil en aplicaciones que deben dar respuestas en tiempo real. Algunos

Ejemplos :

- encontrar rutas entre dos localidades, como ocurre en Google Maps u
- obtener los grados de separación entre una empresa y un empleado potencial al que se quiere contratar, como puede ocurrir en LinkedIn.

El algoritmo de Dijkstra es uno de los algoritmos más utilizados en teoría de grafos para la obtención de caminos mínimos.



4.2.4 Caminos mínimos

En caso de querer obtener el camino mínimo entre un par de nodos concretos, es posible utilizar el código mostrado a continuación:

```
// Define la ciudad de origen y destino
```

```
WITH "Amsterdam" AS startCity, "London" AS endCity
```

```
// Encuentra los nodos correspondientes a las ciudades de origen y destino
```

```
MATCH (start:City {id: startCity}), (end:City {id: endCity})
```

```
// Ejecuta el algoritmo de Dijkstra para encontrar el camino mínimo
```

```
CALL apoc.algo.dijkstra(start, end, 'EROAD', 'cost') YIELD path, weight
```

```
RETURN path, weight
```



4.2.5 Medidas de centralidad

La centralidad se define como la relevancia de un nodo dentro de un grafo.

Por tanto, el estudio de medidas de centralidad permite identificar **nodos relevantes dentro de un grafo**. Esta identificación permitirá entender el comportamiento de la red, qué nodos permiten viralizar con mayor rapidez el contenido de la red, desde qué nodos la información está más accesible, etc.

El estudio de medidas de centralidad tiene multitud de aplicaciones, aunque son especialmente notables las relacionadas con los ámbitos de **la publicidad y el marketing**, donde el estudio de estas métricas permite identificar actores relevantes dentro de una red a los que se puede contratar para anunciar un producto o identificar sobre qué actores enviar información para alcanzar a más usuarios.

4.2.5 Medidas de centralidad

Respecto a las medidas de centralidad, **no existe una única medida sino que, en función de los datos y del propósito que se pretende alcanzar, se utilizan unas métricas u otras**. A continuación, se van a definir tres medidas de centralidad con las que se realizarán ejemplos en Neo4j: centralidad de grado, cercanía e intermediación.

Para trabajar con medidas de centralidad, se va a importar un grafo social que servirá de ejemplo para el cálculo de estas métricas. Este es el código necesario para su creación.

```
LOAD CSV WITH HEADERS FROM
```

```
'https://raw.githubusercontent.com/neo4j-graph-analytics/book/master/data/social-nodes.csv' AS row
```

```
MERGE (:User {id: row.id})
```

```
LOAD CSV WITH HEADERS FROM
```

```
'https://raw.githubusercontent.com/neo4j-graph-analytics/book/master/data/social-relationships.csv'
```

```
AS row
```

```
MATCH (source:User {id: row.src})
```

```
MATCH (destination:User {id: row.dst})
```

```
MERGE (source)-[:FOLLOWS]->(destination)
```



4.2.5 Medidas de centralidad

Centralidad de un grado

Este código cuenta el número de relaciones :FOLLOWS que tiene cada nodo User y lo utiliza para calcular su centralidad de grado. Luego, devuelve los resultados ordenados de mayor a menor centralidad.

```
MATCH (u:User)-[:FOLLOWS]->(other:User)
WITH u, count(*) AS degree
RETURN u.id AS UserId, degree AS DegreeCentrality
ORDER BY DegreeCentrality DESC
```

CALL apoc.node.degree(['User'], ['FOLLOWS']):
Utiliza la función apoc.node.degree para calcular la centralidad de grado de los nodos → error en el uso de la función

.....

ORDER BY DegreeCentrality DESC: Ordena los resultados en orden descendente según la centralidad de grado, de modo que los nodos más centrales aparezcan primero en la tabla de resultados.

*** error en el uso de la función***

```
CALL apoc.node.degree(['User'], ['FOLLOWS'])
YIELD node, degree
RETURN node.id AS UserId, degree AS
DegreeCentrality
ORDER BY DegreeCentrality DESC
```

4.2.5 Medidas de centralidad

Centralidad de un grado

Cercanía

La cercanía es una medida de centralidad que determina qué nodos del grafo expanden rápida y eficientemente la información a través del grafo. Para calcular esta medida, se obtiene la suma del inverso de las distancias de un nodo al resto.

La cercanía es una métrica muy utilizada para estimar tiempos de llegada en redes logísticas, para descubrir actores en posiciones privilegiadas en redes sociales o para estudiar la prominencia de palabras en un documento en el campo de la minería de textos.

```
CALL gds.alpha.closeness.stream({
  nodeProjection: 'User',
  relationshipProjection: {
    FOLLOWS: {
      type: 'FOLLOWS',
      orientation: 'UNDIRECTED' // Opcional, ajusta según tu
                                // modelo de datos
    }
  }
})
YIELD nodeId, centrality
RETURN gds.util.asNode(nodeId).id AS UserId, centrality
ORDER BY centrality DESC
```

En el grafo de la imagen, Alice, Doug y David tienen una cercanía de 1.



4.2.5 Medidas de centralidad

Centralidad de un grado

Intermediación

La intermediación es una medida de centralidad que permite detectar la influencia que tiene un nodo o actor del grafo en el flujo de información o de recursos de la red. El cálculo de la intermediación **permite identificar a nodos que hacen de puentes entre distintas porciones del grafo.** Esta medida de centralidad es muy utilizada para la identificación de influencers y el estudio de la viralización de mensajes en redes sociales. Intuitivamente, la intermediación de un nodo será mayor en tanto en cuanto dicho nodo aparezca en los caminos mínimos de cualquier otro par de nodos.

```
// Crear el grafo 'myGraph'
CALL gds.graph.create(
  'myGraph',
  'User',
  {
    FOLLOWS: {
      orientation: 'REVERSE'
    }
  }
)

CALL gds.betweenness.stream('myGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS name,
score
ORDER BY score DESC
```



4.2.6 Detección de comunidades

A la hora de trabajar con grafos reales, que presentan una gran cantidad de nodos y enlaces, muchas veces se pretende identificar comunidades dentro del grafo para aplicar algoritmos sobre ellas. Una comunidad es, por tanto, un conjunto de nodos que presentan más relaciones entre sí que con el resto de nodos fuera de la comunidad. La detección e identificación de comunidades permite identificar comportamientos emergentes y de rebaño dentro de una red. De esta forma, es posible **detectar y predecir hábitos de los usuarios.** A continuación, se van a aplicar tres métodos de detección de comunidades sobre el grafo social de ejemplo que se ha utilizado en secciones anteriores.



4.2.6 Detección de comunidades

Conteo de triángulos

Un triángulo es un conjunto de tres nodos que tienen relaciones entre sí. **El conteo de triángulos para un nodo dado, permite estudiar o inspeccionar de forma global un grafo y, aplicado sobre componentes conexas, permite inspeccionar regiones de un grafo.**

Para aplicar este método, es necesario almacenar el grafo teniendo en cuenta que las aristas deben almacenarse como UNDIRECTED.

La ejecución de este método da como resultado que Alice y Doug son aquellos que pertenecen a más triángulos, un total de 5.

```
CALL gds.graph.create(  
  'myGraph2',  
  'User',  
  {  
    FOLLOWS: {  
      orientation: 'UNDIRECTED'  
    }  
  }  
)
```

```
CALL gds.triangleCount.stream('myGraph2')  
YIELD nodeId, triangleCount  
RETURN gds.util.asNode(nodeId).id AS name,  
triangleCount  
ORDER BY triangleCount DESC
```



4.2.6 Detección de comunidades

Coeficiente local de clustering

Este coeficiente proporciona una medida cuantitativa del grado de agrupación de un nodo. Para el cálculo del coeficiente local de clustering se utiliza el conteo de triángulos, de forma que se compara el grado de agrupación de un nodo con el grado máximo de agrupación que podría tener.

Los actores Bridget, Charles, Mark y Michael tienen el mayor coeficiente local de clustering, que es 1.



4.2.6 Detección de comunidades

```
CALL gds.localClusteringCoefficient.stream('myGraph2')
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).id AS name, localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC
```

Por último, el coeficiente global de clustering se calcula como la suma normalizada de los coeficientes de clustering locales. Así, estos coeficientes nos permiten encontrar medidas cuantitativas para detectar comunidades, pudiendo especificar incluso un umbral para establecer la comunidad (por ejemplo, especificar que los nodos han de estar conectados en un 40 %).



4.2.6 Detección de comunidades

Componentes fuertemente conexas

En un grafo dirigido, una componente fuertemente conexa es aquel grupo de nodos en el que cualquier nodo puede ser alcanzado por cualquier otro en ambas direcciones. El estudio de las componentes fuertemente conexas en un grafo permite estudiar la conectividad de la red.

nodes: Cantidad total de nodos en el grafo.

communityCount: Cantidad de componentes fuertemente conexas encontradas en el grafo.

setCount: Cantidad total de conjuntos dentro de las componentes conexas.

minSetSize: Tamaño mínimo de un conjunto dentro de las componentes conexas.

maxSetSize: Tamaño máximo de un conjunto dentro de las componentes conexas.

```
CALL gds.alpha.scc.write({
  nodeProjection: 'User',
  relationshipProjection: {
    FOLLOWS: {
      type: '*',
      orientation: 'UNDIRECTED'
    }
  },
  writeProperty: 'componentId'
})
```



4.2.7 Predicción de enlaces

Los grafos son estructuras de datos que representan sistemas dinámicos, que evolucionan a lo largo del tiempo. Por este motivo, es muy común que en un grafo cualquiera aparezcan y desaparezcan nuevos nodos y conexiones entre dichos nodos. Los métodos de predicción de enlaces permiten predecir qué enlaces se formarán próximamente entre los nodos del grafo, permitiendo adelantarse a los acontecimientos y prevenir eventualidades. Como norma general, los métodos de predicción de enlaces se basan en medidas de cercanía y de centralidad asumiendo que los nuevos enlaces se producirán, mayoritariamente, sobre/desde los nodos más relevantes.

Vecinos comunes

El método de vecinos comunes se basa en la idea genérica de que dos actores de la red que tienen una relación con un usuario común tendrán más posibilidad de conectarse entre sí que quienes no.



4.2.7 Predicción de enlaces

Vecinos comunes: El resultado de este cálculo es 2.

```
MATCH (x:User {id: 'Charles'})  
MATCH (y:User {id: 'Bridget'})  
RETURN gds.alpha.linkprediction.commonNeighbors(x, y) AS score
```



4.2.7 Predicción de enlaces

Adhesión preferencial

Este método se basa en la idea general de que cuanto más conectado está un nodo, es más probable que reciba nuevos enlaces.

En el grafo social de ejemplo, el cálculo de adhesión preferencial para Charles y Bridget se puede realizar a través del siguiente código, el resultado de este cálculo es 10.

```
MATCH (x:User {id: 'Charles'})  
MATCH (y:User {id: 'Bridget'})  
RETURN gds.alpha.linkprediction.preferentialAttachment(x, y) AS score
```



4.2.7 Predicción de enlaces

Asignación de recursos


Se trata de una métrica compleja que evalúa la cercanía de un par de nodos para determinar la posibilidad de que, entre ellos, se produzca un nuevo enlace. En el grafo social de ejemplo, el cálculo de la asignación de recursos para Charles y Bridget se puede realizar a través del siguiente código, el resultado de este cálculo es 0.309.

```
MATCH (x:User {id: 'Charles'})  
MATCH (y:User {id: 'Bridget'})  
RETURN gds.alpha.linkprediction.resourceAllocation(x, y) AS score
```


Resumen

<u>Término</u>	<u>Uso</u>	<u>Caso de Uso 1</u>	<u>Caso de Uso 2</u>	<u>De Interés</u>
Recorrido en anchura (BFS)	Para recorrer o buscar en árboles o grafos	Encontrar el camino más corto en un laberinto	Detección de nivel en redes sociales	Útil para estructuras de datos no ponderados
Recorrido en profundidad (DFS)	Para recorrer o buscar en árboles o grafos	Realizar una copia de un directorio de archivos	Ordenación topológica en grafos dirigidos	Puede no encontrar el camino más corto
Caminos mínimos	Encontrar el camino más corto entre nodos	Ruteo de paquetes en redes de computadoras	Logística y optimización de rutas de transporte	Algoritmos como Dijkstra, Bellman-Ford

Resumen



<u>Término</u>	<u>Uso</u>	<u>Caso de Uso 1</u>	<u>Caso de Uso 2</u>	<u>De Interés</u>
Medidas de centralidad	Identificar nodos importantes en redes	Analizar influenciadores en redes sociales	Identificación de servidores clave en redes IT	Centralidad de grado, cercanía, intermediación
Centralidad de grado	Medir la importancia basada en conexiones	Análisis de redes sociales	Análisis de robustez de redes	Fácil de calcular, pero no considera topología
Cercanía	Evaluar la accesibilidad de un nodo	Planificación urbana y de transporte	Estudio de la eficiencia de redes de información	Útil en redes grandes para identificar centros
Intermediación	Identificar nodos que actúan como puentes	Análisis de redes de comunicaciones	Control de tráfico y congestión en redes	Revela nodos críticos para la integridad de red

Resumen

<u>Término</u>	<u>Uso</u>	<u>Caso de Uso 1</u>	<u>Caso de Uso 2</u>	<u>De Interés</u>
Conteo de triángulos	Medir la densidad de conexiones en redes	Análisis de comunidades en redes sociales	Detección de fraudes en redes financieras	Puede indicar la presencia de clusters
Coeficiente local de clustering	Medir la probabilidad de que los vecinos de un nodo también estén conectados	Teoría de redes y sociología	Estudio de coautorías en publicaciones científicas	Indica el nivel de cohesión alrededor de un nodo
Componentes fuertemente conexas	Identificar subgrupos dentro de grafos dirigidos	Análisis de la estructura de la web	Análisis de seguridad en redes de software	Detectar módulos o comunidades
Vecinos comunes	Predecir la probabilidad de formación de una nueva conexión	Recomendación de amistades en redes sociales	Análisis de co-compra en comercio electrónico	Sencillo pero efectivo para predicciones
Adhesión preferencial	Explicar cómo las redes evolucionan y crecen	Modelado del crecimiento de Internet	Dinámica de citaciones en artículos científicos	Fundamental en la teoría de redes
Asignación de recursos	Distribuir recursos en redes	Distribución de carga en redes de energía	Optimización de presupuesto en proyectos	Importante para la optimización de recursos



Big Data Aplicado

Curso de especialización en Inteligencia Artificial y Big Data.

Computación distribuida



¿Cuáles son las diferentes arquitecturas de computación paralela utilizadas en el procesamiento de Big Data y cuáles son sus características principales?

¿Qué ventajas ofrece la computación distribuida en el contexto de Big Data y cuáles son los conceptos clave asociados, como la concurrencia y la escalabilidad?

¿Cuál es la diferencia entre la computación en la nube y la computación distribuida, y cómo se aplican en el procesamiento de Big Data?

¿Cuáles son las características clave de los sistemas de archivos distribuidos (DFS) y qué ejemplos existen en el ámbito de Big Data?

¿Cuáles son las técnicas comunes para gestionar fallos en sistemas de Big Data y por qué son importantes?

¿Cuáles son los componentes clave de Hadoop y cómo contribuyen a la escalabilidad y la tolerancia a fallos en aplicaciones de Big Data?

¿Qué es Apache Spark y cómo se diferencia de Hadoop en el procesamiento de grandes conjuntos de datos?

¿Cuáles son las ventajas de usar Spark?