Apuntes ingeniería de Datos

Sección 1: Fundamentos de Almacenamiento y Modelado de Datos

**1.1 OLTP vs OLAP**

Antes de hablar de Data Warehouses y Data Lakes, primero debemos entender estos dos tipos de bases de datos:

**📖 Teoría**

* **OLTP (Online Transaction Processing)** → Diseñado para manejar transacciones en tiempo real con muchas operaciones pequeñas y rápidas.
* **OLAP (Online Analytical Processing)** → Diseñado para el análisis de grandes volúmenes de datos con consultas complejas.

Interfaz de usuario gráfica

Descripción generada automáticamente

**Fases de Normalización en Bases de Datos**

La **normalización** es el proceso de estructurar una base de datos para minimizar la redundancia y mejorar la integridad de los datos. Se hace a través de **formas normales (NF, por sus siglas en inglés)**, donde cada nivel mejora la organización respecto al anterior.

A continuación, te explico las **tres primeras fases de normalización** (las más utilizadas en OLTP), y la última fase que no se aplica en bases NoSQL.

**📌 Primera Forma Normal (1NF) - Eliminar duplicados y crear claves primarias**

**Regla principal:**

Cada columna debe contener valores atómicos y cada fila debe ser única.

Ejemplo de tabla sin normalizar:  
Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

* Aquí **no cumple con 1NF**, porque la columna "Teléfonos" tiene múltiples valores en una celda (no son atómicos).

**Tabla en 1NF (Corrección)**

Se descompone la tabla para que cada campo tenga solo un valor por celda:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Segunda Forma Normal (2NF) - Eliminar dependencias parciales**

**Regla principal:**

Debe cumplir con 1NF **y** cada columna debe depender completamente de la clave primaria.

**Ejemplo de tabla en 1NF pero no en 2NF:**

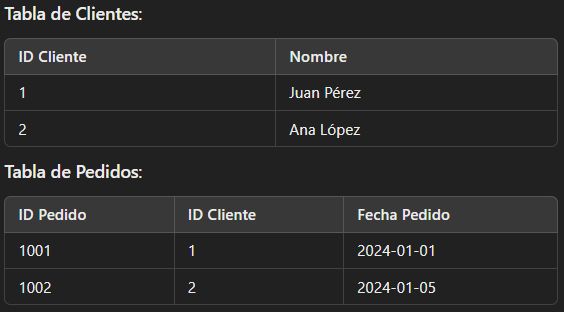
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

📌 **Problema:** La columna "Nombre Cliente" depende solo de "ID Cliente", no de "ID Pedido", lo que **viola 2NF**.

**Tabla en 2NF (Corrección)**

Se separa la información en dos tablas para eliminar dependencias parciales:



**📌 Tercera Forma Normal (3NF) - Eliminar dependencias transitivas**

**Regla principal:**

Debe cumplir con 2NF y **no debe haber dependencias transitivas** (columnas que dependen de otra que no sea la clave primaria).

**Ejemplo de tabla en 2NF pero no en 3NF:**

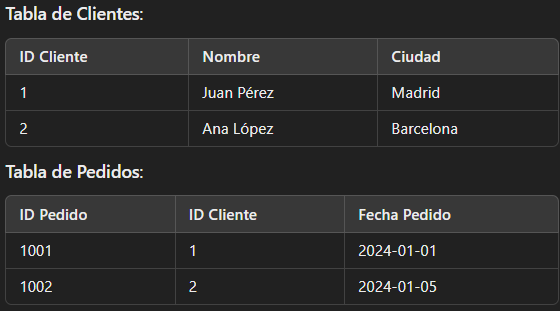
Captura de pantalla de un celular

Descripción generada automáticamente

📌 **Problema:** "Ciudad" depende de "Nombre Cliente", pero "Nombre Cliente" no es la clave primaria, **rompiendo 3NF**.

**Tabla en 3NF (Corrección)**

Separamos la ciudad en otra tabla:



**🔗 Conceptos Relacionados**

* **OLTP se usa en sistemas transaccionales como MySQL y PostgreSQL.**
* **OLAP se usa en Data Warehouses como Snowflake, Redshift o BigQuery.**
* **Los datos en OLTP suelen moverse a OLAP mediante ETL o ELT.**

**🎯 Ejemplo Práctico**

Imagina que tienes una tienda en línea:

1. Tu base de datos **OLTP** es un **MySQL** donde guardas las compras de los clientes:

Texto

Descripción generada automáticamente

1. Luego quieres analizar cuántos productos se venden por mes. Para eso **cargas los datos en un Data Warehouse OLAP** (ejemplo en BigQuery):

Texto

Descripción generada automáticamente

**📌 1.2 Data Warehouse**

Un **Data Warehouse (DWH)** es una base de datos optimizada para el análisis de datos históricos y generación de reportes.

**📖 Teoría**

* Un DWH almacena datos estructurados de diferentes fuentes.
* Optimizado para consultas rápidas con grandes volúmenes de datos.
* Usa estructuras como **esquema estrella y copo de nieve**.
* Ejemplos: **BigQuery, Snowflake, Redshift, Azure Synapse.**

🔹 **Ejemplo de Arquitectura de un DWH:**

1. **Fuente de datos:** OLTP (MySQL/PostgreSQL).
2. **Proceso ETL:** Se extraen los datos y se limpian.
3. **Almacenamiento:** Se guarda en un Data Warehouse.
4. **Consulta y Análisis:** Se usa Power BI o SQL para analizar.

🔹

**Esquema Estrella (Star Schema)**

**📖 Teoría**

El esquema estrella es **simple y eficiente** para consultas rápidas. Se llama así porque su estructura se asemeja a una estrella, con una tabla central (**tabla de hechos**) conectada a varias **tablas de dimensiones**.

**Características:**

* La **tabla de hechos** contiene datos transaccionales y métricas (ventas, ingresos, etc.).
* Las **tablas de dimensiones** almacenan información descriptiva (clientes, productos, tiempo).
* Es fácil de entender y optimiza consultas SQL con agregaciones rápidas.
* Utilizado en **reportes de Business Intelligence** y **análisis de datos en tiempo real**.

🛒 **Ejemplo Práctico: Ventas en un Supermercado**

Imagina que queremos analizar las ventas de una cadena de supermercados. Usamos un esquema estrella con una tabla de hechos llamada Ventas y varias tablas de dimensiones:

Tabla

Descripción generada automáticamente con confianza media

Gráfico de rectángulos

Descripción generada automáticamente

**📊 Ventajas del Esquema Estrella**

✅ **Consultas rápidas**: Gracias a la estructura simple, SQL puede hacer agregaciones eficientes.  
✅ **Fácil de entender y mantener**: Ideal para analistas de negocio.  
✅ **Optimizado para herramientas BI** como Power BI, Tableau, etc.

📌 **Ejemplo de consulta SQL rápida:**

Obtener el total de ventas por producto en enero 2024.

Texto

Descripción generada automáticamente

**Esquema Snowflake (Copo de Nieve)**

**📖 Teoría**

El esquema Snowflake es una versión más avanzada del esquema estrella donde **las dimensiones están normalizadas** en varias tablas.

**Características:**

* La **tabla de hechos** sigue siendo el centro, pero las dimensiones se dividen en subtablas.
* Reduce la redundancia de datos, pero las consultas pueden volverse más complejas y lentas.
* Se usa cuando hay **grandes volúmenes de datos** y se necesita mayor eficiencia en el almacenamiento.

**🛒 Ejemplo Práctico: Snowflake en el mismo Supermercado**

Ahora, en lugar de almacenar toda la información de clientes, productos y fechas en una sola tabla de dimensión, **las dividimos**:

**📌 Tabla de Hechos: Ventas (igual que en el esquema estrella)**

Captura de pantalla de un celular

Descripción generada automáticamente

**📊 Ventajas del Esquema Snowflake**

✅ **Menos redundancia**: Se almacena menos información repetida.  
✅ **Mejor para almacenamiento eficiente**: Especialmente útil en bases de datos grandes.  
✅ **Más flexible para cambios** en las dimensiones.

📌 **Ejemplo de consulta SQL más compleja:**

Obtener el total de ventas por categoría de producto en enero 2024.

Texto

Descripción generada automáticamente

**📌 1.3 Data Lake**

Un **Data Lake** es un repositorio que almacena grandes volúmenes de datos en su estado original.

**📖 Teoría**

* Guarda **datos estructurados y no estructurados**.
* Se usa para análisis avanzado, machine learning, y big data.
* Puede contener archivos JSON, Parquet, CSV, logs, imágenes, videos.
* **Ejemplos:** AWS S3, Azure Data Lake, Google Cloud Storage.

🔹 **Comparación Data Warehouse vs Data Lake:**

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

**📌 1.4 Data Lakehouse**

Un **Data Lakehouse** es un híbrido entre Data Warehouse y Data Lake.

**📖 Teoría**

* Usa la escalabilidad y flexibilidad de un **Data Lake**.
* Tiene la estructura y velocidad de consultas de un **Data Warehouse**.
* Soporta **procesamiento en tiempo real** y **batch**.
* Ejemplo: **Databricks Delta Lake, Snowflake, Google BigQuery.**

🔹 **Ventajas:**

✅ Menor costo que un Data Warehouse puro.  
✅ Mantiene flexibilidad para almacenar datos crudos.  
✅ Mejor rendimiento con tecnologías como **Apache Iceberg, Delta Lake, Hudi**.

🔹 **Casos de uso:**

* Análisis de logs en grandes volúmenes.
* Machine Learning en datos históricos.
* Reportes empresariales con dashboards en Power BI.

**📌 1.5 NoSQL**

Las bases de datos **NoSQL** son optimizadas para escalabilidad y flexibilidad.

**📖 Teoría**

* A diferencia de las bases de datos SQL, **NoSQL** no usa tablas y relaciones.
* **Tipos de bases de datos NoSQL:**
  + **Clave-Valor:** Redis, DynamoDB.
  + **Documentos:** MongoDB, Firebase.
  + **Columnares:** Apache Cassandra, HBase.
  + **Grafos:** Neo4j.

🔹 **Comparación SQL vs NoSQL**

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

Bases de Datos NoSQL con Grafos

Una base de datos de grafos usa una estructura compuesta por:

🔹 **Nodos (Entities)** → Son los objetos o entidades principales (Ejemplo: Personas, Productos, Lugares).  
🔹 **Aristas o Relaciones (Edges)** → Conectan los nodos y pueden tener atributos (Ejemplo: "Amigo de", "Compró", "Trabaja en").  
🔹 **Propiedades (Properties)** → Son atributos de los nodos o relaciones (Ejemplo: Nombre, Edad, Fecha de compra).

**Bases de Datos Columnares**

Las bases de datos **columnares**, como **Google BigQuery**, están diseñadas para **consultas analíticas rápidas sobre grandes volúmenes de datos**. A diferencia de las bases de datos tradicionales, almacenan los datos **por columna en lugar de por fila**.

**📖 Teoría: ¿Cómo funcionan las bases de datos columnares?**

🔹 En bases de datos relacionales tradicionales (**SQL como PostgreSQL, MySQL**), los datos se almacenan **por filas**.  
🔹 En bases de datos **columnares** (**BigQuery, Apache Cassandra, Redshift, ClickHouse**), los datos se almacenan **por columna**.

📌 **Ejemplo de almacenamiento por filas (SQL tradicional)**

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Ejemplo de almacenamiento por columnas (BigQuery)

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

En lugar de leer toda la fila, **BigQuery solo lee las columnas necesarias**, lo que lo hace más eficiente en consultas analíticas.

**Sección 2: Procesamiento de Datos - ETL, ELT y Streaming**

📌 2.1 ETL vs ELT vs EtLT

Son los **tres métodos más comunes** para mover y transformar datos.

**📖 Teoría**

🔹 **ETL (Extract - Transform - Load)**

* Se extraen los datos desde fuentes externas.
* Se transforman (limpieza, agregaciones, cálculos).
* Luego se cargan en un Data Warehouse.
* **Ejemplo**: Extraer datos de MySQL → Transformarlos con Pandas → Cargarlos en BigQuery.

🔹 **ELT (Extract - Load - Transform)**

* Se extraen y **se cargan directamente en el Data Warehouse**.
* La transformación ocurre dentro del DWH con SQL o herramientas como **dbt**.
* **Ejemplo**: Extraer datos de MySQL → Cargar sin cambios en Snowflake → Transformar con SQL.

🔹 **EtLT (Extract - transform-Little - Load - Transform)**

* Un híbrido: Se hace una **pequeña transformación antes de cargar** y luego se termina en el DWH.
* **Ejemplo**: Convertir formatos de fechas antes de cargar en Redshift.

**🔗 Conceptos Relacionados**

* **Data Warehouses (BigQuery, Snowflake)** almacenan los datos transformados en ETL/ELT.
* **Data Lakes (S3, GCS, Azure Blob)** pueden usarse para almacenar datos sin transformar en ELT.
* **Apache Airflow, Informatica Cloud** se usan para gestionar ETL/ELT.

**📌 2.2 Batch Processing vs Streaming**

Existen **dos formas principales** de procesar datos:

**📖 Teoría**

🔹 **Batch Processing (Procesamiento por lotes)**

* Procesa grandes volúmenes de datos **de una sola vez**.
* Se usa en **ETL clásico** con Apache Spark, SQL, Airflow.
* **Ejemplo:** Procesar todas las ventas del día cada noche.

🔹 **Stream Processing (Procesamiento en tiempo real)**

* Procesa datos **en tiempo real** a medida que llegan.
* Se usa con Kafka, Apache Flink, Google Pub/Sub.
* **Ejemplo:** Detectar fraude en transacciones bancarias en tiempo real.

**🔗 Conceptos Relacionados**

* **ETL suele ser batch**, pero **ELT y streaming pueden ser en tiempo real**.
* **Herramientas batch:** Apache Spark, Airflow.
* **Herramientas streaming:** Kafka, Pub/Sub, Flink.

**📌 2.3 Data Orchestration**

La **orquestación de datos** se refiere a la **automatización** de flujos de datos en ETL y ELT.

**📖 Teoría**

* Permite **programar y monitorear tareas ETL/ELT**.
* Usa **flujos de trabajo (DAGs)** para definir dependencias entre tareas.
* **Herramientas:** Apache Airflow, Dagster, Prefect.

Texto

Descripción generada automáticamente

**📌 2.4 Stream Processing (Procesamiento de Datos en Tiempo Real)**

Para **procesar datos en tiempo real**, usamos herramientas como **Kafka, Flink y Pub/Sub**.

**📖 Teoría**

* **Apache Kafka**: Sistema de mensajería en tiempo real basado en eventos.
* **Apache Flink**: Plataforma de procesamiento de stream para cálculos en tiempo real.
* **Google Pub/Sub**: Servicio en la nube para transmisión de datos en tiempo real.

🔹 **Ejemplo de Arquitectura Streaming**

1. **Kafka recibe eventos de compras en una tienda online.**
2. **Flink procesa los datos en tiempo real y detecta fraudes.**
3. **BigQuery almacena eventos para análisis posterior.**

**📖 Arquitectura del Caso de Uso**

📌 **Objetivo:**

1. Airflow ejecuta **Spring Batch** para procesar datos ETL desde Firebird y PostgreSQL.
2. Spring Batch envía eventos a **Kafka** en tiempo real para monitoreo.
3. Una aplicación en **Flink** consume los eventos de Kafka y los almacena en **BigQuery**.

📌 **Tecnologías utilizadas:**  
✅ **Apache Airflow** → Orquestador de Workflows.  
✅ **Spring Batch** → Procesa ETL (migración de datos desde Firebird).  
✅ **Apache Kafka** → Mensajería en tiempo real (envía eventos ETL).  
✅ **Apache Flink** → Procesamiento en tiempo real de los eventos de Kafka.  
✅ **Google BigQuery** → Almacena eventos procesados para análisis posterior.

**📖 8️ Change Data Capture (CDC)**

* **Detecta cambios en los datos** y solo mueve lo que ha cambiado.
* **Ejemplo**: Si una fila en una base de datos SQL es actualizada, CDC solo captura esa actualización en lugar de mover toda la tabla.

🔹 **Herramientas CDC:**

* **Debezium** (para Kafka)
* **Fivetran** (para sincronización de bases de datos)

🔹 **Ejemplo de CDC con MySQL:**

Captura de pantalla de un celular

Descripción generada automáticamente

**Aquí, cada vez que un registro es actualizado, la columna actualizado se modifica, permitiendo detectar cambios para CDC.**

**Sección 3: Arquitectura de Datos y Modern Data Stack**

**📌 3.1 Modern Data Stack (MDS)**

El **Modern Data Stack (MDS)** es un enfoque basado en herramientas **cloud-native** para gestionar datos con bajo costo y alta escalabilidad.

**📖 Teoría**

* Usa herramientas **SaaS y serverless** en la nube.
* Se basa en **ETL/ELT, almacenamiento en Data Warehouses en la nube y análisis con BI**.
* Reduce la necesidad de gestionar infraestructura compleja.

🔹 **Ejemplo de Arquitectura MDS:** 1️⃣ **Ingesta:** Fivetran (automatiza la extracción de datos).  
2️ **Almacenamiento:** Snowflake, BigQuery o Redshift.  
3️ **Transformación:** dbt (define modelos de datos en SQL).  
4️ **Análisis:** Looker, Power BI, Tableau.

**📌 3.2 Open Data Stack**

El **Open Data Stack** es similar al MDS, pero usa herramientas **open-source** en lugar de servicios SaaS.

🔹 **Ejemplo de Open Data Stack con herramientas open-source:** 1️⃣ **Ingesta:** Airbyte  
2️⃣ **Almacenamiento:** DuckDB o ClickHouse  
3️⃣ **Transformación:** dbt-core (versión open-source)  
4️⃣ **Orquestación:** Dagster  
5️⃣ **Análisis:** Superset o Metabase

🔹 **Comparación MDS vs Open Data Stack**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**📌 3.3 DataOps (DevOps para Datos)**

DataOps es un **conjunto de prácticas para gestionar datos como código**, similar a DevOps en software.

**📖 Teoría**

* **Automatiza el ciclo de vida de los datos** (ingesta, transformación, entrega).
* Usa herramientas como **CI/CD, pruebas automatizadas y control de versiones**.
* Se aplica en **ETL/ELT, Data Warehouses y machine learning**.

🔹 **Ejemplo de DataOps con CI/CD en dbt**

Captura de pantalla de un celular

Descripción generada automáticamente

🔹 **Aquí validamos que la columna id nunca tenga valores nulos en dbt.**

**📌 3.4 Data Product**

Un **Data Product** es un **conjunto de datos diseñado para ser reutilizado** en múltiples aplicaciones, de manera estructurada y confiable.

**📖 Explicación Sencilla**

Piensa en un **Data Product como un servicio de datos listo para ser consumido**. En lugar de simplemente almacenar datos en una base de datos, los organizamos y los hacemos accesibles para que otros equipos los usen fácilmente.

🔹 **Ejemplo en la vida real:**

* **Netflix** tiene un **Data Product de recomendaciones** que usa datos de historial de usuarios y los expone para que la app pueda sugerir series o películas.
* **Amazon** tiene un **Data Product de inventario**, donde cada sistema dentro de la empresa puede acceder a información de stock en tiempo real.

🔹 **Ejemplo en una empresa:**

* Un equipo de marketing necesita saber qué clientes hicieron más compras.
* En lugar de consultar manualmente una base de datos, se crea un **Data Product "Clientes Premium"** con esa información lista para usar.

**🎯 Ejemplo Práctico**

**Imagina que trabajamos en un e-commerce y queremos ofrecer datos de ventas como un Data Product a otros departamentos.**

**1️ Creamos un modelo de datos con la información relevante**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**2️ Publicamos esta información en una API con FastAPI**

**Texto

Descripción generada automáticamente**

📌 **3.5 Data Lifecycle y Data Engineering Architectures Overview**

🔹 **Data Lifecycle (Ciclo de vida de los datos)**

* **Ingesta** → Captura de datos desde múltiples fuentes.
* **Almacenamiento** → Bases de datos, Data Warehouses o Data Lakes.
* **Procesamiento** → Transformación y limpieza.
* **Consumo** → BI, ML, APIs.
* **Gobernanza** → Seguridad y cumplimiento de regulaciones.

🔹 **Data Engineering Architectures Overview**  
Existen múltiples arquitecturas de datos:

1️ **Lambda Architecture**: Para datos batch y streaming en tiempo real.  
2️ **Kappa Architecture**: Solo streaming, elimina batch.  
3️ **Medallion Architecture**: Organiza datos en niveles **Bronze (crudos), Silver (procesados), Gold (agregados).**

🔹 **Ejemplo de arquitectura Lambda con Kafka y Spark Streaming**

**📌 Declarative vs Imperative Data Pipelines**

🔹 **Explicación**

* **Imperative:** Definimos **paso a paso** cómo transformar datos (Ejemplo: Python, SQL procedural).
* **Declarative:** Solo definimos el **resultado esperado** (Ejemplo: SQL, dbt).

🔹 **Ejemplo Comparativo** ✅ **Imperative (Python)**

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

✅ **Declarative (SQL)**

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

**📌 DAD Stack (Data Activation & Delivery Stack)**

🔹 **Explicación**

* Combina herramientas de **Reverse ETL y Activación de Datos**.
* **Ejemplo de herramientas:** Census, Hightouch (mueve datos desde DWH a sistemas operacionales).

**📌 NGODS (Next-Gen Operational Data Stack)**

🔹 **Explicación**

* Evolución de OLTP para datos operacionales en tiempo real.
* Usa **bases de datos de baja latencia como ClickHouse o Materialize**.

**📌 Microsoft Fabric**

🔹 **Explicación**

* Plataforma unificada de Microsoft para **almacenamiento, transformación y análisis** de datos.
* Combina **Power BI, Synapse y Data Factory** en una sola herramienta.

🔹 **Ejemplo de Arquitectura con Microsoft Fabric**

1. **Ingesta:** Conectamos datos de SQL Server.
2. **Transformación:** Usamos Data Factory.
3. **Análisis:** Creamos dashboards en Power BI.

✅ **Conclusión:** Es una solución **todo en uno** para empresas que usan el ecosistema de Microsoft.

**Data Governance: Gestión y Seguridad de Datos**

**📌 ¿Qué es Data Governance?**

🔹 **Explicación**

* Es el **conjunto de políticas, procesos y herramientas** que aseguran que los datos sean confiables, seguros y accesibles solo para quienes los necesitan.
* Incluye **calidad, seguridad, privacidad, cumplimiento legal y linaje de datos**.

🔹 **Ejemplo en la vida real**

* Una empresa de salud **no puede compartir datos de pacientes sin permiso** (HIPAA).
* Una tienda online **debe encriptar datos de clientes** para cumplir con GDPR.

**📌 Componentes Clave de Data Governance**

* **Data Quality** → Asegura datos limpios y sin errores.
* **Data Security** → Protege los datos contra accesos no autorizados.
* **Data Lineage** → Rastreabilidad: ¿de dónde vienen y a dónde van los datos?
* **Master Data Management (MDM)** → Gestión de datos centrales, como clientes y productos.
* **Compliance** → Cumplimiento de regulaciones como **GDPR, CCPA, HIPAA**.

📌 Ejemplo Práctico de Data Governance

🔹 1. Implementar Data Quality en dbt

Texto

Descripción generada automáticamente con confianza media

✅ **Aquí aseguramos que los emails sean únicos y no nulos.**

🔹 **2. Implementar Data Security con Column-Level Security en Snowflake**

Texto

Descripción generada automáticamente

🚀 **Sección 4: Data Management (Gestión de Datos)**

📌 **4.1 Data Catalog (Catálogo de Datos)**

Un **Data Catalog** es como un **diccionario de datos** para toda una empresa. Es una herramienta que **organiza, documenta y facilita la búsqueda de datos** en un ecosistema de datos (Data Warehouse, Data Lake, bases de datos transaccionales, etc.).

**📖 Explicación Detallada**

📌 **¿Para qué sirve un Data Catalog?**  
✅ Facilita la **búsqueda de datos** dentro de una organización.  
✅ Permite ver **quién ha usado los datos y cómo se han transformado** (**Data Lineage**).  
✅ Evita **duplicación de datos** y mejora la calidad de los informes.  
✅ Ayuda en el **cumplimiento de normativas (GDPR, ISO 27001, etc.)**.  
✅ Permite gestionar **permisos y accesos** a los datos.

📌 Ejemplo Práctico con Google Data Catalog

Si usas **BigQuery**, puedes consultar metadatos de las tablas con **INFORMATION\_SCHEMA**.

📌 **Ejemplo de consulta en Google Data Catalog para ver metadatos de una tabla**:

✅ Esto devuelve **toda la información sobre las columnas** en la tabla google\_analytics\_sample.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

📌 Ejemplo de Data Lineage (Linaje de Datos)

📌 **Si tienes estos datos en BigQuery:**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**📌 4.12 Bitemporal Modeling (Modelado Bitemporal)**

El **modelado bitemporal** permite **registrar dos líneas de tiempo en los datos**:

1. **Tiempo de negocio (Valid Time):** Cuándo ocurrió realmente el evento.
2. **Tiempo del sistema (Transaction Time):** Cuándo el evento fue registrado en la base de datos.

**📌 4.13 Bridge Tables (Tablas Puente)**

Las **Tablas Puente** son usadas en **modelado de datos dimensional** para resolver relaciones **muchos a muchos** entre dimensiones.

**📖 Explicación**

* En modelado **dimensional**, una dimensión puede relacionarse con **varias** filas en otra tabla.
* En lugar de almacenar **listas dentro de una tabla**, se usa una **Bridge Table** para normalizar.

**📌 4.2 Data Contracts (Contratos de Datos)**

**📖 Teoría**

Un **Data Contract** es un **acuerdo formal entre productores y consumidores de datos** que define:

* **Estructura:** Tipos de datos, nombres de columnas, etc.
* **Requisitos de calidad:** Valores no nulos, claves únicas.
* **Reglas de transformación.**

**📌 4.4 Data Quality (Calidad de Datos)**

**📖 Teoría**

**Data Quality** asegura que los datos sean:

* **Exactos:** Sin errores.
* **Completo:** Sin valores nulos inesperados.
* **Consistente:** Sin contradicciones entre diferentes fuentes.

**📌 4.10 Idempotency**

📖 Teoría

La idempotencia garantiza que repetir una operación varias veces no tendrá efectos secundarios. Es fundamental en procesos ETL para evitar duplicaciones.

🎯 Ejemplo Práctico: Operación Idempotente en SQL

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

🔄 Si la fila ya existe, **solo actualiza el registro** en lugar de duplicarlo.

**📌 4.12 Data Assets (Activos de Datos)**

**📖 Teoría**

Un **Data Asset** es cualquier **recurso de datos valioso**, como bases de datos, APIs, dashboards, etc.

**📌 4.13 Database Triggers (Disparadores de Bases de Datos)**

**📖 Teoría**

Un **Trigger** es un **evento automático** que se ejecuta cuando ocurre una acción en la base de datos (insert, update, delete).

**🎯 Ejemplo Práctico: Trigger en PostgreSQL**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**📌 4.6 Slowly Changing Dimensions (SCDs)**

Los **Slowly Changing Dimensions (SCDs)** son un enfoque clave en **Data Warehousing** para manejar **cambios en los datos históricos**. Se utilizan especialmente en el **modelado dimensional**, como en esquemas estrella (Star Schema), donde tenemos tablas de hechos (fact tables) y tablas de dimensiones (dimension tables).

**📖 ¿Qué son Slowly Changing Dimensions (SCDs)?**

Cuando hablamos de datos históricos, como información de **clientes, productos o empleados**, estos datos **cambian lentamente** con el tiempo:

* Un cliente cambia su dirección.
* Un producto cambia de categoría.
* Un empleado recibe una promoción.

El desafío es decidir **cómo manejar esos cambios**:

* ¿Deberíamos **sobrescribir** los datos antiguos?
* ¿Deberíamos **guardar un historial completo** de los cambios?
* ¿O tal vez solo queremos **conservar el cambio más reciente** además del valor actual?

Aquí es donde entran en juego los diferentes **tipos de SCD**.

**SCD Type 1: Sobrescribir el dato antiguo**

**📖 Teoría**

* **No se conserva el historial de cambios.**
* Los datos antiguos se **sobrescriben** con los nuevos valores.
* Se usa cuando **no es importante mantener el historial**.

**🎯 Ejemplo Práctico - Actualizar el Email de un Cliente**

Supongamos que el cliente "Juan Pérez" cambia su correo electrónico.

**Tabla antes del cambio (Type 1):**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Tabla después del cambio:

Interfaz de usuario gráfica

Descripción generada automáticamente

🔑 **Resultado:**  
El correo antiguo (juan@gmail.com) **se pierde** porque fue sobrescrito.

**SCD Type 2: Mantener el historial de cambios**

**📖 Teoría**

* **Se guarda cada cambio como un nuevo registro**, manteniendo un historial completo.
* Se añaden columnas para la **fecha de inicio**, **fecha de fin**, o un indicador de **registro actual**.
* Es el tipo más común en sistemas que requieren **auditoría o seguimiento de cambios históricos**.

**🎯 Ejemplo Práctico - Cambiar la Dirección de un Cliente**

Supongamos que Juan Pérez se muda a una nueva dirección.

Tabla

Descripción generada automáticamente

**3️ SCD Type 3: Guardar solo el cambio más reciente**

**📖 Teoría**

* Guarda **solo el valor actual y el valor anterior** del campo que cambió.
* No mantiene un historial completo, pero es útil para ver el **cambio más reciente**.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Sección 5: Data Modeling & Warehousing**

**📌5.1 Metrics Layer**

**📖 Teoría**

El **Metrics Layer** es una capa de abstracción que **centraliza el cálculo de métricas** de negocio para que todos los equipos de una organización usen definiciones consistentes.

* Define métricas como **ingresos, crecimiento de usuarios, tasas de conversión**, etc.
* Facilita que estas métricas estén **disponibles para dashboards, análisis o APIs**.

**🎯 Ejemplo Práctico con dbt Metrics:**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**📌 5.2 Semantic SQL**

**📖 Teoría**

**Semantic SQL** se refiere a la capacidad de **definir relaciones y significados** en el SQL, más allá de solo manipular datos.

* Permite que las herramientas de BI **entiendan el significado de los datos**, no solo su estructura.
* Se usa en plataformas como **Looker** o **dbt**.

**🎯 Ejemplo Práctico: Semantic Model en LookML (Looker)**

**Texto

Descripción generada automáticamente**

**📌 5.3 Semantic Warehouse**

**📖 Teoría**

Un **Semantic Warehouse** añade una capa semántica sobre el Data Warehouse, lo que permite que los usuarios puedan hacer **consultas complejas sin escribir SQL**.

* **Ejemplo:** Power BI y Looker permiten consultar datos usando definiciones semánticas predefinidas.

**🎯 Ejemplo Práctico: Semantic Layer en Power BI**

En Power BI, defines **medidas (measures)** usando **DAX** para que estén disponibles en todos los dashboards:

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

**📌 5.4 Data Virtualization**

**📖 Teoría**

La **Data Virtualization** permite consultar datos de **múltiples fuentes sin necesidad de moverlos** físicamente.

* Se puede acceder a bases de datos SQL, NoSQL, APIs, etc., como si fueran una sola.
* Herramientas: **Denodo, Dremio, Apache Drill**.

**🎯 Ejemplo Práctico con Apache Drill**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

🔹 **Aquí unimos datos de MySQL y AWS S3 sin mover los datos físicamente.**

**📌 5.5 Data Modeling (Modelado de Datos)**

**📖 Teoría**

El **Data Modeling** organiza los datos para optimizar su almacenamiento y consulta.  
Se divide en tres niveles:

1. **Conceptual:** ¿Qué entidades existen? (Clientes, Productos)
2. **Lógico:** ¿Cómo se relacionan? (Tablas, claves foráneas)
3. **Físico:** ¿Cómo se almacenan? (Índices, particiones)

**📌 5.7 Data Vault**

**📖 Teoría**

**Data Vault** es un enfoque para modelar datos históricos en entornos de Big Data.

* **Hubs:** Entidades principales (clientes, productos).
* **Links:** Relaciones entre hubs.
* **Satélites:** Información adicional (atributos históricos).

**🎯 Ejemplo Práctico: Estructura Data Vault**

Texto

Descripción generada automáticamente

**📌 5.10 Materialized Views**

**📖 Teoría**

Una **Materialized View (MV)** es como una **tabla derivada** de una consulta SQL, pero con la **diferencia clave** de que **almacena físicamente los resultados de esa consulta** en la base de datos.

* A diferencia de una **vista normal**, que se recalcula cada vez que la consultas, la MV **guarda los datos "precalentados"**.
* Esto significa que **las consultas son mucho más rápidas** porque no necesita volver a hacer todos los cálculos cada vez.
* Sin embargo, como los datos están **almacenados físicamente**, **pueden quedar desactualizados si los datos originales cambian**.
* Por eso, necesitas **refrescar (actualizar) la vista** de vez en cuando.

**🎯 Ejemplo Práctico: Materialized View en PostgreSQL**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

🔄 **Se puede refrescar manualmente:**

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

**📌 5.11 Granularity & Cardinality**

**📖 Teoría**

* **Granularidad:** Nivel de detalle de los datos (ej: ventas por día vs por hora).
* **Cardinalidad:** Cantidad de valores únicos en una columna (alta en IDs, baja en género).

**📌 5.18 CTE (Common Table Expressions)**

**📖 Teoría**

Un **CTE** permite crear subconsultas temporales para mejorar la legibilidad del SQL.

🎯 **Ejemplo Práctico: CTE en SQL**

Texto

Descripción generada automáticamente

**One Big Table (OBT)**

**📖 ¿Qué es una One Big Table?**

Una **One Big Table (OBT)** es un enfoque de modelado de datos donde se **combina toda la información relevante en una sola tabla grande** en lugar de tener varias tablas normalizadas y relacionadas.

* **Objetivo:** **Optimizar la velocidad de consulta** al evitar la necesidad de realizar múltiples JOINs entre tablas.
* **Uso común:** En **dashboards de BI** (Power BI, Tableau), donde el rendimiento es crucial.

**🧠 ¿Cómo funciona?**

* Se realiza un proceso de **desnormalización**, uniendo tablas de hechos (Fact Tables) con todas las dimensiones relevantes (Dimension Tables).
* Esto genera una tabla plana con todos los datos necesarios para el análisis.

**🎯 Ejemplo Práctico: One Big Table**

Supongamos que tenemos un esquema tradicional con varias tablas relacionadas:

1️**Tabla de Ventas (Fact Table):**

Texto

Descripción generada automáticamente

Tabla de Clientes (Dimensión):

Texto

Descripción generada automáticamente

**Super Table**

**📖 ¿Qué es una Super Table?**

Una **Super Table** es una evolución de la OBT, diseñada para manejar **grandes volúmenes de datos** con optimizaciones adicionales para mejorar el rendimiento.

**🧠 ¿Cómo se diferencia de OBT?**

* **Indexación avanzada:** Incluye índices en columnas clave para mejorar la velocidad de las consultas.
* **Particionamiento:** Divide la tabla en partes más pequeñas basadas en criterios (por fecha, región, etc.).
* **Compresión de datos:** Reduce el uso de almacenamiento y mejora el rendimiento.
* **Columnar Storage:** En bases de datos columnar (como BigQuery, Redshift), se almacena por columnas en lugar de filas.

Sección 6: Performance & Optimization

**📌 6.1 Push-downs**

**📖 Teoría**

El **Push-down** es una técnica de optimización que consiste en **llevar las operaciones de filtrado, agregación o transformación directamente a la base de datos** en lugar de hacerlo en la aplicación o herramienta de análisis.

* **¿Por qué es importante?** Reduce la cantidad de datos que se transfieren y mejora el rendimiento de las consultas.

**🎯 Ejemplo Práctico: Sin Push-down vs Con Push-down**

✅ **Con Push-down (Eficiente):**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **Resultado:** Solo se transfieren los datos necesarios, mejorando el rendimiento.

**📌 6.2 Metrics & KPIs**

**📖 Teoría**

* **Métricas (Metrics):** Medidas cuantitativas que se calculan directamente de los datos (ej: total de ventas, número de clientes).
* **KPIs (Key Performance Indicators):** Métricas clave que ayudan a evaluar el rendimiento del negocio en función de sus objetivos.

🎯 **Ejemplo Práctico: Definir KPIs en SQL**

Texto

Descripción generada automáticamente

✅ **Diferencia:** total\_ventas es una métrica simple, mientras que tasa\_conversion es un KPI que mide el éxito del negocio.

**📌 6.3 CAP Theorem (Teorema CAP)**

**📖 Teoría**

El **Teorema CAP** establece que un sistema distribuido solo puede garantizar **dos de estas tres propiedades a la vez**:

1. **Consistencia (Consistency):** Todos los nodos muestran la misma información en cualquier momento.
2. **Disponibilidad (Availability):** El sistema responde a todas las solicitudes, incluso si algunos nodos fallan.
3. **Tolerancia a Particiones (Partition Tolerance):** El sistema sigue funcionando incluso si hay fallos en la red.

**🚀 Ejemplos Prácticos:**

* **CP (Consistencia + Tolerancia a Particiones):** Bases de datos como **HBase**.
* **AP (Disponibilidad + Tolerancia a Particiones):** Sistemas como **Cassandra** o **DynamoDB**.
* **CA (Consistencia + Disponibilidad):** Difícil de lograr en sistemas distribuidos, porque no manejan bien las particiones de red.

**📌 6.4 Full Table Scan**

📖 Teoría

Un Full Table Scan ocurre cuando la base de datos tiene que leer todas las filas de una tabla para resolver una consulta. Esto puede ser muy costoso en términos de rendimiento, especialmente en tablas grandes.

🎯 Ejemplo Práctico: Cómo evitar un Full Table Scan

❌ Consulta que provoca Full Table Scan:

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

* **Problema:** La función LOWER() impide que el índice se use.

✅ **Optimización con índice:**

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

**Resultado:** La base de datos usa el índice, evitando el Full Table Scan y mejorando la velocidad de la consulta.

**📌 6.5 Delta Load**

**📖 Teoría**

El **Delta Load** es una técnica de ETL que consiste en **cargar solo los datos que han cambiado desde la última actualización** en lugar de recargar toda la tabla.

* **Ventaja:** Reduce el tiempo de procesamiento y la carga en el sistema.

**🎯 Ejemplo Práctico: Delta Load en SQL**

Supongamos que queremos cargar solo las ventas realizadas desde la última ejecución del ETL.

1️ I**dentificar la última fecha de carga:**

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

Cargar solo los datos nuevos:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

✅ **Resultado:** Solo se cargan los registros nuevos o actualizados desde la última fecha.

**Sección 7: Cloud & Big Data Technologies**

**📌 7.1 Cloud Data Warehouses**

**📖 Teoría**

Un **Cloud Data Warehouse (CDW)** es una base de datos optimizada para el análisis de grandes volúmenes de datos, que se aloja en la nube.

* **Escalable:** Puedes aumentar o disminuir los recursos fácilmente.
* **Flexible:** Pago por uso, sin preocuparte por la infraestructura física.
* **Ejemplos populares:** **Google BigQuery, Amazon Redshift, Snowflake, Azure Synapse.**

**🎯 Ejemplo Práctico: Consulta en BigQuery**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

✅ **Beneficio:** BigQuery procesa rápidamente millones de registros gracias a su arquitectura distribuida.

**📌 7.2 Cloud Data Providers**

**📖 Teoría**

Un **Cloud Data Provider** es el servicio que ofrece infraestructura en la nube para almacenar, procesar y analizar datos.

* **Principales proveedores:**
  + **AWS (Amazon Web Services):** Redshift, S3, Glue.
  + **Google Cloud Platform (GCP):** BigQuery, Dataflow, Dataproc.
  + **Microsoft Azure:** Synapse, Data Lake, Data Factory.

**📖 ¿Qué es un Bucket?**

Un **bucket** es un **contenedor de almacenamiento** en la nube que se utiliza para guardar y organizar archivos (conocidos como **objetos**).

📦 **Piensa en un bucket como una "carpeta gigante" en la nube**, donde puedes almacenar cualquier tipo de archivo:

* Documentos (.pdf, .docx)
* Imágenes (.png, .jpg)
* Videos (.mp4)
* Archivos de datos (.csv, .json, .parquet)
* Backups o registros de aplicaciones

🔑 Cada bucket tiene un **nombre único** dentro del proveedor de la nube.

**📌 Ejemplo en AWS S3**

📦 Imagina que creas un bucket llamado mi-bucket-datos en **AWS S3**.  
Dentro del bucket, puedes tener varios archivos organizados por carpetas virtuales:

Texto

Descripción generada automáticamente

**Política de Acceso (ACL)**

Un bucket puede tener políticas para definir **quién puede acceder** a los datos.

* **Privado:** Solo el propietario puede acceder (por defecto).
* **Público:** Cualquier persona con el enlace puede ver el archivo.
* **Acceso limitado:** Solo ciertos usuarios o aplicaciones pueden acceder.

**📌 7.3 Data Fabric**

**📖 Teoría**

El **Data Fabric** es una arquitectura que permite **acceder y gestionar datos de múltiples fuentes de forma unificada**, sin importar dónde estén almacenados.

* **Automatiza:** La integración y el movimiento de datos.
* **Gobernanza:** Mejora la seguridad y la calidad de los datos.

**🎯 Ejemplo Práctico: Integración con Data Fabric (Conceptual)**

* Con Data Fabric, puedes **consultar datos de MongoDB, SQL Server y un Data Lake de AWS** desde un solo punto de acceso.

Patrón de fondo

Descripción generada automáticamente

✅ **Sin mover los datos físicamente**, se puede acceder a ellos como si estuvieran en la misma base de datos.

**📌 7.4 Data Mesh**

**📖 Teoría**

El **Data Mesh** es un enfoque descentralizado de gestión de datos donde **cada equipo es responsable de sus propios datos** como si fueran productos (Data as a Product).

* **Principios Clave:**
  + **Descentralización:** Cada dominio de negocio gestiona sus datos.
  + **Interoperabilidad:** Estándares comunes para compartir datos entre equipos.
  + **Autonomía:** Equipos independientes pueden crear y consumir datos.

**🎯 Ejemplo Práctico: Data Mesh en una empresa**

* **Equipo de Finanzas:** Responsable de los datos financieros.
* **Equipo de Marketing:** Responsable de los datos de campañas publicitarias.  
  Cada equipo publica sus datasets como "productos de datos" que otros equipos pueden usar.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

✅ **Cada equipo gestiona y documenta su propio "producto de datos".**

**📌 7.5 MLOps (Machine Learning Operations)**

**📖 Teoría**

**MLOps** es la combinación de **Machine Learning** y **DevOps** para automatizar el ciclo de vida completo de los modelos de IA, incluyendo:

* **Entrenamiento de modelos.**
* **Despliegue automatizado (CI/CD).**
* **Monitoreo de modelos en producción.**

**🎯 Ejemplo Práctico: Pipeline de MLOps con Python**

Texto

Descripción generada automáticamente

✅ Con MLOps, este modelo se puede **automatizar para entrenamiento, despliegue y monitoreo** continuo en producción.

**📌 7.6 Microsoft OneLake**

**📖 Teoría**

**Microsoft OneLake** es una **plataforma de almacenamiento de datos unificada** que permite gestionar grandes volúmenes de datos de diferentes fuentes en la nube de Azure.

* **Integración nativa:** Con Power BI, Azure Synapse y otras herramientas de Microsoft.
* **Escalabilidad:** Ideal para Data Lakes empresariales.

**🎯 Ejemplo Práctico: Carga de Datos en OneLake (Conceptual)**

Texto

Descripción generada automáticamente

✅ **Este comando carga datos directamente en Microsoft OneLake para su análisis.**

**📌 7.7 Cube & Cube Store**

**📖 Teoría**

**Cube** es una plataforma de análisis de datos que actúa como un **cubo OLAP** moderno para agilizar la creación de dashboards y reportes.

* **Cube Store:** Es la base de datos interna de Cube que optimiza la caché y el rendimiento de las consultas.
* **Ideal para:** Herramientas de BI como **Tableau, Looker, Power BI**.

🎯 **Ejemplo Práctico: Definir un Cubo OLAP**

Texto

Descripción generada automáticamente

**Sección 8: Data Integration & Federation**

📌 **8.1 Data Orchestrators**

**📖 Teoría**

Los **Data Orchestrators** son herramientas que permiten **automatizar, programar y monitorear flujos de trabajo de datos (ETL/ELT)**.

* **Gestión de dependencias:** Controlan el orden de ejecución de las tareas.
* **Automatización:** Programan tareas de extracción, transformación y carga.
* **Monitoreo:** Alertas en caso de fallos.

**🚀 Herramientas populares:**

* **Apache Airflow**
* **Dagster**
* **Prefect**

**🎯 Ejemplo Práctico: DAG en Apache Airflow**

**Texto

Descripción generada automáticamente**

✅ **Este código define un flujo ETL diario con dos tareas en Airflow.**

**📌 8.2 Data Lineage**

**📖 Teoría**

El **Data Lineage (linaje de datos)** es el proceso de **rastrear el origen, la transformación y el destino de los datos** en todo su ciclo de vida.

* **¿Por qué es importante?**
  + **Auditoría:** Ver de dónde provienen los datos.
  + **Depuración:** Identificar rápidamente dónde ocurrió un error.
  + **Cumplimiento:** Asegura la trazabilidad para normativas (GDPR, HIPAA).

**🎯 Ejemplo Práctico: Linaje en Google Data Catalog**

Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente

✅ **Esto permite rastrear cómo los datos de la tabla ventas han sido transformados.**

**📌 8.4 GraphQL**

**📖 Teoría**

**GraphQL** es un lenguaje de consulta para APIs que permite a los clientes **obtener exactamente los datos que necesitan**, ni más ni menos.

* **Ventaja:** Reduce el "over-fetching" (obtener datos innecesarios) y el "under-fetching" (no obtener datos suficientes).

**🎯 Ejemplo Práctico: Consulta GraphQL**

Captura de pantalla de un celular

Descripción generada automáticamente

**📌 8.5 Analytics API**

**📖 Teoría**

Una **Analytics API** permite **consultar y analizar datos** de forma programática desde aplicaciones.

* **Ejemplos:** APIs de Google Analytics, Mixpanel, o APIs personalizadas para dashboards.

**🎯 Ejemplo Práctico: Llamada a una API de Analytics (Python)**

Texto

Descripción generada automáticamente

✅ **Este código obtiene datos de ventas del 1 de enero de 2024 desde una API de Analytics.**

**📌 8.6 Data Integration**

**📖 Teoría**

La **Data Integration** consiste en **combinar datos de diferentes fuentes** en un solo lugar para su análisis.

* **Ejemplos:** Integrar datos de una base SQL, un Data Lake y una API externa en un solo Data Warehouse.
* **Herramientas:** **Informatica Cloud, Talend, Fivetran, Stitch.**

**🎯 Ejemplo Práctico: Integración de datos con SQL**

Texto

Descripción generada automáticamente

✅ **Esta consulta integra datos de tres tablas para obtener una vista combinada de clientes, productos y ventas.**

**📌 8.7 Data Model Engine**

**📖 Teoría**

**Un Data Model Engine es una herramienta o motor que permite definir, gestionar y optimizar modelos de datos de forma automática.**

* **Ejemplos: Looker (LookML), dbt (Data Build Tool), Power BI (DAX).**

**🎯 Ejemplo Práctico: Modelo en dbt**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**✅ Este modelo de dbt define la estructura de un dataset de ventas agrupadas por cliente.**

**📌 8.8 Change Data Capture (CDC)**

**📖 Teoría**

**El CDC (Change Data Capture) permite detectar y capturar cambios en los datos en tiempo real para sincronizarlos con otros sistemas.**

* **Uso común: Replicación de bases de datos, ETL en tiempo real.**
* **Herramientas: Debezium, Fivetran, AWS DMS.**

**🎯 Ejemplo Práctico: CDC con PostgreSQL (Trigger)**

**Texto

Descripción generada automáticamente**

**📌 8.9 Datafusion**

**📖 Teoría**

**Datafusion es una plataforma de integración de datos que permite construir y gestionar pipelines ETL de forma visual.**

* **Proveedor popular: Google Cloud Datafusion.**
* **Permite orquestar ETL sin necesidad de escribir mucho código.**

**🎯 Ejemplo Práctico: Pipeline de ETL en Datafusion (Conceptual)**

* **Origen: MySQL**
* **Transformación: Limpieza de datos con reglas de negocio**
* **Destino: BigQuery**

**✅ Todo esto se configura de forma visual en Datafusion.**

**📌 8.10 Ballista**

**📖 Teoría**

**Ballista es un motor de computación distribuido basado en Rust y Apache Arrow, diseñado para ejecutar consultas SQL en grandes volúmenes de datos de manera eficiente.**

* **Escalable: Ideal para análisis en clusters.**
* **Optimizado: Alto rendimiento gracias a Apache Arrow.**

**🎯 Ejemplo Práctico: Consulta Distribuida con Ballista (Conceptual)**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Apache Zookeeper y por qué es importante para Kafka**

**Apache Zookeeper es un sistema de coordinación distribuido que se utiliza para gestionar la configuración, el seguimiento del estado de los nodos y la coordinación entre servicios en entornos distribuidos.**

**En el caso de Apache Kafka, Zookeeper se encarga de coordinar los brokers de Kafka y mantener el sistema funcionando de manera estable.**

**📖 1️ ¿Por qué Kafka necesita Zookeeper?**

**Cuando usas Apache Kafka, normalmente tienes múltiples brokers (servidores Kafka) trabajando juntos para procesar datos en tiempo real. Aquí es donde entra Zookeeper:**

**🔑 Zookeeper en Kafka hace tres cosas importantes:**

1. **Gestión de Brokers:**
   * **Zookeeper rastrea qué brokers están activos y cuáles han fallado.**
   * **Si un broker de Kafka falla, Zookeeper detecta el fallo y reasigna tareas a otros brokers.**
2. **Gestión de Particiones y Líderes:**
   * **En Kafka, cada topic se divide en particiones.**
   * **Zookeeper decide qué broker es el "líder" de cada partición, lo cual es fundamental para la alta disponibilidad.**
3. **Gestión de Configuración Distribuida:**
   * **Zookeeper mantiene la configuración global de Kafka.**
   * **Por ejemplo, almacena información sobre los topics, particiones y controladores de replicación.**

**🚀 2️ Ejemplo: Kafka y Zookeeper trabajando juntos**

**🎯 Imagina este escenario:**

* **Tienes un cluster de Kafka con 3 brokers: Broker 1, Broker 2, Broker 3.**
* **Hay un topic llamado ventas con 3 particiones.**

**¿Qué hace Zookeeper aquí?**

1. **Coordina la distribución de las particiones:**
   * **Partition 0 → Broker 1 (líder)**
   * **Partition 1 → Broker 2 (líder)**
   * **Partition 2 → Broker 3 (líder)**
2. **Detecta fallos:**
   * **Si Broker 1 falla, Zookeeper automáticamente asigna Partition 0 a otro broker (por ejemplo, Broker 2).**
   * **Kafka sigue funcionando sin que te des cuenta del fallo.**
3. **Controla el equilibrio del cluster:**
   * **Si agregas un nuevo broker, Zookeeper lo detecta y Kafka comienza a redistribuir la carga automáticamente.**