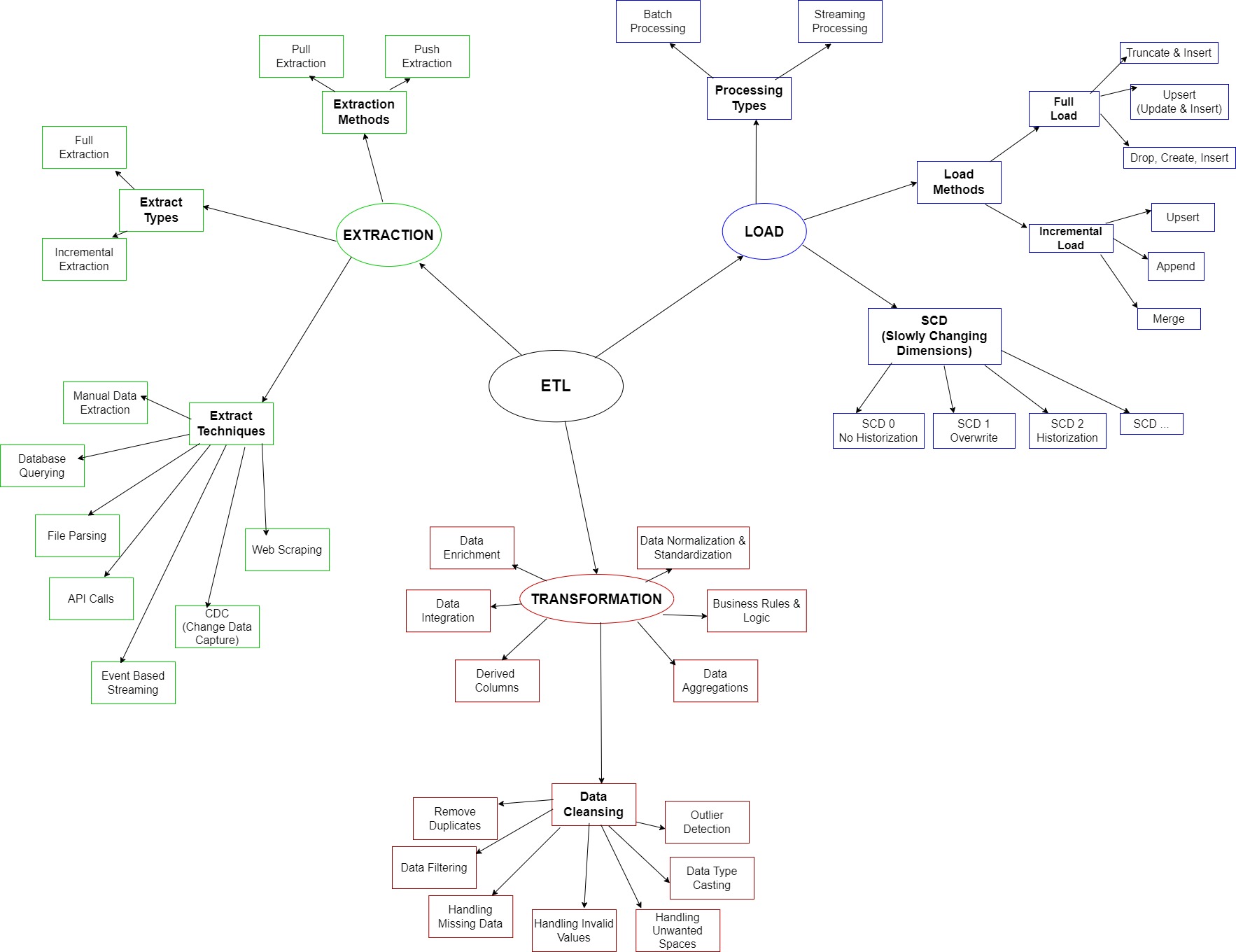
**SQL Data Warehouse Project**

**ETL Diagram**



The image is a conceptual diagram explaining the **ETL (Extract, Transform, Load) process**, which is commonly used in data warehousing and data integration. The diagram is divided into three main sections:

**1. Extraction (EXTRACTION)**

This phase is responsible for retrieving data from source systems. It has three subcategories:

* **Extraction Methods:**
  + **Pull Extraction:** The system actively requests data from the source.
  + **Push Extraction:** The source system sends data without a request.
* **Extract Types:**
  + **Full Extraction:** Extracts the entire dataset every time.
  + **Incremental Extraction:** Only extracts new or modified records since the last extraction.
* **Extract Techniques:**
  + **Manual Data Extraction:** Extracting data manually.
  + **Database Querying:** Using SQL queries to retrieve specific data.
  + **File Parsing:** Extracting data from structured files (CSV, JSON, XML, etc.).
  + **API Calls:** Connecting to external services via APIs to extract data.
  + **Event-Based Streaming:** Extracting data in real-time based on events.
  + **CDC (Change Data Capture):** Capturing only changes in the data rather than the entire dataset.
  + **Web Scraping:** Extracting data from websites.

**2. Transformation (TRANSFORMATION)**

This stage processes and refines the extracted data before loading it into the final system. It includes:

* **Data Cleansing:**
  + Removing duplicates
  + Data filtering
  + Handling missing data
  + Handling invalid values
  + Outlier detection
  + Data type casting
  + Handling unwanted spaces
* **Data Integration:** Combining data from different sources into a unified format.
* **Derived Columns:** Creating new columns from existing data.
* **Data Enrichment:** Enhancing data with additional information.
* **Data Normalization & Standardization:** Ensuring consistency in data formats.
* **Business Rules & Logic:** Applying custom rules to process data.
* **Data Aggregations:** Summarizing and grouping data for analysis.

**3. Load (LOAD)**

This stage loads the transformed data into a destination system (Data Warehouse, database, etc.). It includes:

* **Processing Types:**
  + **Batch Processing:** Loading data in scheduled batches.
  + **Streaming Processing:** Loading data in real-time.
* **Load Methods:**
  + **Full Load:** Completely replacing the existing data.
    - Truncate & Insert
    - Upsert (Update & Insert)
    - Drop, Create, Insert
  + **Incremental Load:** Updating only changed data.
    - Upsert
    - Append
    - Merge
* **SCD (Slowly Changing Dimensions):** A technique to manage historical data changes in dimensions.

1. **SCD 0: No historization.** In this case, the information is completely overwritten every time there’s a change. No trace of the previous data is kept. Each update simply replaces the old one, with no versions saved.
2. **SCD 1: Overwrites old data.** The key difference is that although old data is overwritten, updates are generally allowed without keeping historical records. However, in some cases, rules might be implemented to allow a slight history (for example, only keeping the last version). This might seem similar to SCD 0, but the intent is that the data is updated in a more controlled way rather than just being replaced with no tracking.
3. **SCD 2: Versioning (keeping historical records).** In this case, multiple versions of a data point are stored, so you can see how it has changed over time. This is typically done by adding columns like start\_date and end\_date, or an active field, which allows querying the history of changes.
4. **SCD 3:** Keeps a limited number of versions. This approach stores a limited number of versions (usually the current and the previous one). It allows for some historization, but doesn't keep all past changes.
5. **SCD 4:** Keeps change records in a separate table. Instead of storing historical data in the same table, changes are recorded in a separate table. This can be useful for reducing the complexity of the main table, but may require more effort when querying.

Each technique has its advantages depending on the use case and data system requirements.

**Summary**

This diagram visually represents the **ETL pipeline**, showing how data is extracted, transformed, and loaded into a final system using different techniques and processes.

**Arquitecturas de Datos en Data Warehouses.**

Las arquitecturas de datos son modelos que definen cómo se organizan, almacenan, gestionan y procesan los datos dentro de una organización. La imagen muestra cuatro tipos principales:

**1. Data Warehouse (Almacén de Datos)**

📌 Características:

* Es un repositorio centralizado que almacena datos estructurados de múltiples fuentes.
* Usa un modelo relacional (SQL) con esquemas bien definidos (ejemplo: esquema en estrella, copo de nieve).
* Los datos se integran mediante ETL (Extract, Transform, Load), asegurando limpieza y consistencia.
* Es altamente optimizado para consultas analíticas y reportes de BI.
* Soporta ACID transactions para integridad de datos.

📌 Ventajas:

✅ Alto rendimiento en consultas.  
✅ Estructura bien organizada para análisis y reporting.  
✅ Seguridad y control de acceso estrictos.

📌 Desventajas:

❌ No maneja bien datos no estructurados.  
❌ Requiere transformación previa antes de almacenar los datos.  
❌ Costoso y menos flexible que otras opciones.

📌 Ejemplos:

* Amazon Redshift
* Google BigQuery
* Snowflake

**2. Data Lake**

**📌** Características:

* Es un repositorio de almacenamiento masivo para datos sin procesar.
* Puede almacenar datos estructurados, semiestructurados y no estructurados en su formato original.
* Usa un enfoque de schema-on-read, lo que permite flexibilidad al analizar los datos.
* Se integra con tecnologías de big data y machine learning.
* Los datos pueden ser accedidos mediante Apache Spark, Hadoop o SQL engines.

📌 Ventajas:

✅ Almacena grandes volúmenes de datos a bajo costo.  
✅ No requiere transformación previa (ingesta en crudo).  
✅ Ideal para procesamiento en paralelo y aprendizaje automático.

📌 Desventajas:

❌ Riesgo de convertirse en un "data swamp" (desorden de datos).  
❌ Consultas más lentas en comparación con un Data Warehouse.  
❌ Puede ser complejo de gestionar sin una buena gobernanza.

📌 Ejemplos:

* AWS S3 + AWS Glue
* Azure Data Lake
* Google Cloud Storage

**3. Data Lakehouse**

**📌** Características:

* Es una combinación de Data Lake + Data Warehouse.
* Permite el almacenamiento flexible de datos sin procesar como un Data Lake, pero con la estructura y optimización de un Data Warehouse.
* Soporta transacciones ACID y procesamiento SQL.
* Usa formatos optimizados como Delta Lake, Apache Iceberg y Hudi.
* Es ideal para análisis en tiempo real y machine learning.

📌 Ventajas:

✅ Combina lo mejor de Data Lake y Data Warehouse.  
✅ Mayor eficiencia en consultas SQL en comparación con un Data Lake.  
✅ Menor costo en comparación con un Data Warehouse puro.

📌 Desventajas:

❌ Aún en desarrollo, menos maduro que Data Warehouses tradicionales.  
❌ Puede requerir herramientas especializadas (ejemplo: Databricks).

📌 Ejemplos:

* Databricks Lakehouse
* Delta Lake (sobre Apache Spark)
* Google BigLake

**4. Data Mesh**

**📌** Características:

* Es un enfoque descentralizado para gestionar datos a gran escala.
* En lugar de centralizar todo en un solo repositorio, los datos se gestionan como productos independientes por diferentes equipos (data domains).
* Usa una arquitectura distribuida con gobernanza federada.
* Promueve autonomía de equipos, escalabilidad y flexibilidad.
* Requiere fuertes estándares de interoperabilidad y gobernanza.

📌 Ventajas:

✅ Escalable para organizaciones grandes.  
✅ Mejora la calidad del dato al estar gestionado por equipos expertos en cada dominio.  
✅ Evita los cuellos de botella de un equipo centralizado de datos.

📌 Desventajas:

❌ Difícil de implementar sin una buena estrategia de gobernanza.  
❌ Puede generar redundancia si no hay control adecuado.  
❌ No es ideal para empresas pequeñas o con datos centralizados.

📌 Ejemplos:

* Organizaciones que manejan datos por dominio, como Netflix o Uber
* Uso de tecnologías descentralizadas como Kafka, Snowflake Data Sharing

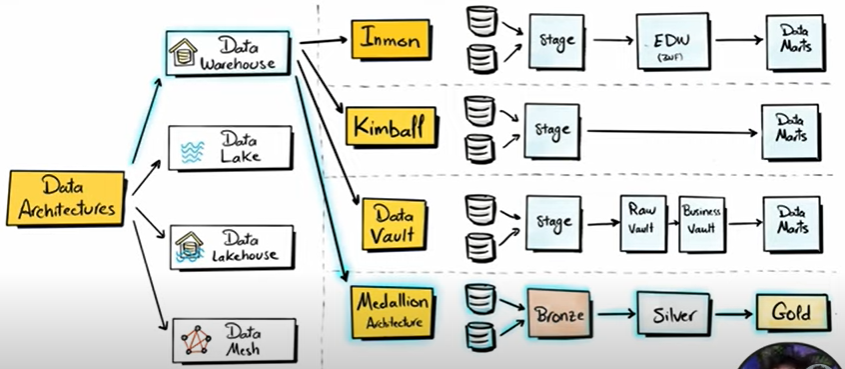
**📊 Comparación de las 4 arquitecturas**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**📌 Conclusión**

Cada arquitectura tiene un propósito distinto y la mejor opción depende del caso de uso:

* Data Warehouse → Empresas que necesitan reporting estructurado y análisis de BI.
* Data Lake → Organizaciones con datos diversos y proyectos de Machine Learning.
* Data Lakehouse → Empresas que buscan flexibilidad sin sacrificar estructura.
* Data Mesh → Grandes organizaciones con múltiples equipos gestionando datos.



**📊 Modelos de Arquitectura de Datos en Profundidad**

Cada modelo en la imagen representa una estrategia diferente para almacenar, organizar y utilizar datos dentro de una empresa. Dependiendo de los **objetivos del negocio**, **tamaño de los datos**, **frecuencia de actualización** y **tecnologías disponibles**, una empresa puede elegir uno u otro.

A continuación, exploramos cada modelo con más detalles, ventajas, desventajas y casos de uso.

**1️⃣ Inmon's Approach – Enterprise Data Warehouse (EDW)**

🔹 **Creador:** Bill Inmon  
🔹 **Concepto clave:** Datos **altamente normalizados** en un **almacén de datos centralizado (EDW)** antes de ser usados en análisis.

**🔍 Explicación detallada**

* Se extraen datos desde distintas fuentes (ERP, CRM, sensores, etc.).
* Se procesan en un **Stage Area** para limpieza y validación.
* Se almacenan en un **Enterprise Data Warehouse (EDW)** en formato **3NF (Tercera Forma Normal)**, lo que minimiza la redundancia.
* Desde el EDW, se generan **Data Marts**, que son subconjuntos específicos para cada departamento (Ventas, Finanzas, Recursos Humanos, etc.).

**✅ Ventajas**

✔ **Alta calidad e integridad de datos**, útil para informes estratégicos de largo plazo.  
✔ **Menos redundancia**, ya que los datos se almacenan en un formato relacional eficiente.  
✔ **Buena gobernanza y seguridad de datos**, adecuado para industrias altamente reguladas.

**❌ Desventajas**

❌ **Implementación costosa y lenta**, requiere diseñar un esquema de base de datos muy estructurado.  
❌ **No es ideal para análisis en tiempo real**, ya que los datos pasan por muchas etapas antes de llegar a los Data Marts.

**🏢 Ejemplo de uso**

* **Banco Santander**: Implementó un **EDW** para integrar datos de clientes de múltiples sucursales y ofrecer mejores productos financieros basados en el historial de cada cliente.
* **Industria farmacéutica**: Utiliza EDW para gestionar datos clínicos y regulatorios, asegurando cumplimiento con normativas como la FDA.

**🛠 Tecnologías asociadas**

* **Bases de datos:** Oracle Exadata, IBM Db2, Microsoft SQL Server
* **ETL:** Informatica PowerCenter, Talend, AWS Glue

**2️⃣ Kimball's Dimensional Model (Data Marts)**

🔹 **Creador:** Ralph Kimball  
🔹 **Concepto clave:** Enfoque **Bottom-Up** en el que los datos se organizan en **Data Marts** específicos sin necesidad de un almacén centralizado.

**🔍 Explicación detallada**

* Se extraen datos desde distintas fuentes.
* Se limpian en un **Stage Area**.
* Se almacenan directamente en **Data Marts** optimizados para consultas rápidas.
* Se usa un **modelo dimensional** (Star Schema o Snowflake Schema), con **Tablas de Hechos** y **Tablas de Dimensión**.

**✅ Ventajas**

✔ **Más rápido de implementar** que el modelo Inmon.  
✔ **Optimizado para reportes y dashboards**, ya que se diseña para responder preguntas específicas de negocio.  
✔ **Mejor rendimiento en BI**, pues los esquemas dimensionales están optimizados para consultas rápidas.

**❌ Desventajas**

❌ **Redundancia de datos**, ya que cada Data Mart puede repetir información.  
❌ **Difícil de escalar a grandes volúmenes de datos**, porque no hay una estructura central unificadora.

**🏢 Ejemplo de uso**

* **Retail (Amazon, Walmart)**: Cada departamento (ventas, logística, marketing) tiene su propio **Data Mart** para generar informes de ventas y stock de manera independiente.
* **E-commerce**: Tiendas en línea pueden crear Data Marts específicos para analizar tendencias de compra en distintas categorías de productos.

**🛠 Tecnologías asociadas**

* **Bases de datos:** Snowflake, Amazon Redshift, Google BigQuery
* **Herramientas BI:** Tableau, Power BI, Looker

**3️⃣ Data Vault – Modelo Híbrido**

🔹 **Creador:** Dan Linstedt  
🔹 **Concepto clave:** Un modelo **altamente flexible y escalable** que separa los datos en **tres capas**:

1. **Raw Vault:** Datos en bruto, sin transformación.
2. **Business Vault:** Datos refinados con reglas de negocio aplicadas.
3. **Data Marts:** Datos optimizados para análisis.

**🔍 Explicación detallada**

* Se extraen datos y se almacenan **sin transformación** en el **Raw Vault**.
* Luego, en el **Business Vault**, se aplican reglas de negocio y se generan relaciones entre datos.
* Finalmente, los datos van a los **Data Marts**, listos para análisis.

**✅ Ventajas**

✔ **Flexible y adaptable**, ideal para entornos con múltiples fuentes de datos en constante cambio.  
✔ **Escalable para Big Data y sistemas distribuidos**.  
✔ **Buen soporte para auditoría y trazabilidad**, ya que los datos en el Raw Vault nunca se modifican.

**❌ Desventajas**

❌ **Mayor complejidad**, ya que requiere más procesamiento para consultas simples.  
❌ **Curva de aprendizaje alta** para diseñar correctamente la arquitectura.

**🏢 Ejemplo de uso**

* **Airbnb**: Usa Data Vault para integrar datos de reservas, usuarios y propiedades en una plataforma escalable.
* **Industria financiera**: Bancos y aseguradoras usan Data Vault para manejar datos regulatorios con trazabilidad total.

**🛠 Tecnologías asociadas**

* **Bases de datos:** Snowflake, Azure Synapse, PostgreSQL
* **ETL/ELT:** dbt, Fivetran, Apache NiFi

**4️⃣ Medallion Architecture – Data Lakes & Lakehouses**

🔹 **Concepto clave:** Organización de datos en **tres capas** dentro de un **Data Lake**:

1. **Bronze Layer:** Datos en crudo (logs, JSON, CSV, etc.).
2. **Silver Layer:** Datos limpios y estructurados.
3. **Gold Layer:** Datos listos para análisis y Machine Learning.

**🔍 Explicación detallada**

* Los datos entran al **Bronze Layer** sin modificaciones.
* Se limpian, transforman y almacenan en el **Silver Layer**.
* Se refinan en el **Gold Layer** y están listos para dashboards, reportes o modelos de ML.

**✅ Ventajas**

✔ **Ideal para análisis de Big Data y Machine Learning**.  
✔ **Permite manejar datos estructurados y no estructurados**.  
✔ **Escalable en entornos cloud**.

**❌ Desventajas**

❌ **Complejidad en la gobernanza de datos**.  
❌ **Requiere herramientas avanzadas para procesamiento en paralelo**.

**🏢 Ejemplo de uso**

* **Netflix**: Usa Medallion Architecture para analizar logs de usuarios y optimizar recomendaciones.
* **Empresas de IoT**: Sensores industriales generan datos en bruto (Bronze), que luego se procesan para mantenimiento predictivo (Gold).

**🛠 Tecnologías asociadas**

* **Data Lakes:** AWS S3, Azure Data Lake, Google Cloud Storage
* **Procesamiento:** Apache Spark, Databricks, Delta Lake

**📌 Conclusión**

Cada modelo tiene su propósito. Empresas **tradicionales** prefieren **Inmon/Kimball**, mientras que empresas **modernas con Big Data** usan **Data Vault o Medallion Architecture**.

**SOC en el contexto de Data Architecture**

El principio de **Separation of Concerns (SOC)** también se aplica en **arquitectura de datos**, pero de una manera diferente a cómo se usa en el desarrollo de software. En este caso, SOC implica **dividir los procesos de gestión de datos en capas o componentes independientes** para que cada uno tenga una responsabilidad clara y definida.

**📌 Ejemplos de SOC en Data Architecture**

**1️⃣ Arquitectura en Capas para Procesamiento de Datos**

Muchas arquitecturas de datos siguen un enfoque SOC al dividirse en capas como:

* **Capa de Ingesta:** Recopila datos desde diferentes fuentes (APIs, bases de datos, sensores).
* **Capa de Procesamiento:** Limpia y transforma los datos.
* **Capa de Almacenamiento:** Guarda los datos en data lakes, warehouses o bases de datos.
* **Capa de Análisis y Consumo:** Hace consultas, genera dashboards y reportes.

📌 **Ejemplo:**  
En una empresa que usa **Data Lakehouse**, los datos brutos van al **Data Lake**, se transforman en **Data Warehouse**, y los analistas consumen los datos a través de **Power BI o Tableau**.

**2️⃣ Separación en Data Warehousing (Inmon vs. Kimball vs. Data Vault)**

El principio SOC también se aplica en estrategias de **Data Warehousing**.

* **Inmon:** Separa el almacenamiento en **Enterprise Data Warehouse (EDW)** y **Data Marts**.
* **Kimball:** Organiza los datos en **Data Marts** específicos para cada área de negocio.
* **Data Vault:** Divide los datos en **Raw Vault y Business Vault** antes de ser utilizados.

Cada enfoque separa responsabilidades en la gestión de datos.

**3️⃣ Medallion Architecture en Data Lakes**

Este modelo usa SOC para estructurar datos en tres niveles:

* **Bronze:** Datos sin procesar.
* **Silver:** Datos limpiados y estructurados.
* **Gold:** Datos optimizados para análisis.

📌 **Ejemplo:** Una empresa de logística podría almacenar datos brutos de GPS en **Bronze**, filtrar rutas y eliminar errores en **Silver**, y generar reportes de eficiencia en **Gold**.

**2️⃣ Data Mesh y Data Lakehouse: SOC en el Manejo de Datos**

En arquitecturas modernas, SOC también se refleja en **estructuras descentralizadas** como:

* **Data Mesh** → Separa la responsabilidad de los datos en **diferentes dominios de negocio** en lugar de un solo equipo centralizado.
* **Data Lakehouse** → Separa el almacenamiento escalable de datos sin procesar (Data Lake) de la consulta y optimización estructurada (Data Warehouse).

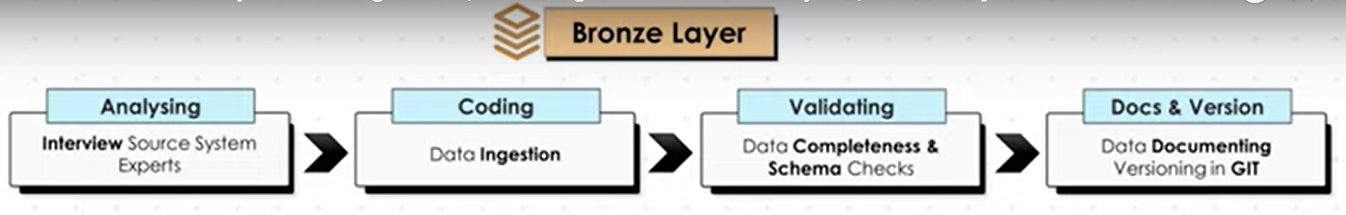
📌 **Ejemplo:**  
En una empresa global, en vez de tener un solo equipo de datos centralizado, cada departamento maneja sus propios datos de manera independiente pero siguiendo estándares comunes.

**✅ Beneficios de SOC en Arquitectura de Datos**

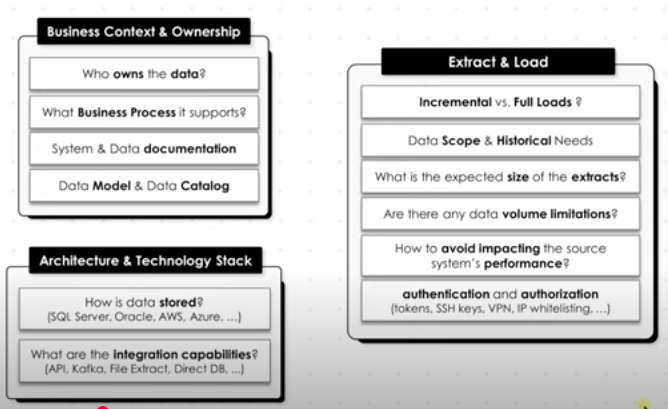
✔ Mayor eficiencia en la gestión y gobernanza de datos.  
✔ Facilita el escalamiento de la infraestructura de datos.  
✔ Reduce la dependencia de un solo equipo para manejar todos los datos.  
✔ Mejora la calidad y accesibilidad de los datos para distintos usuarios.

**✅ Conclusión**

Sí, **Separation of Concerns** se aplica en **Data Architecture**, pero no a nivel de código sino en la **estructura y flujo de datos**. Se usa para mejorar la organización, escalabilidad y eficiencia de los sistemas de datos.

The **Bronze Layer** is the **first layer in the Medallion Architecture**, where data is **ingested in its rawest form**. This image outlines the process in four key stages:

1. **Analyzing**
   * **Interview Source System Experts** → Consulting source system experts to understand the data structure before extraction.
   * This step is crucial to avoid integration errors and assess data quality from the beginning.
2. **Coding**
   * **Data Ingestion** → Developing processes to ingest raw data into the storage system.
   * This may involve ETL (Extract, Transform, Load) or ELT (Extract, Load, Transform) tools.
3. **Validating**
   * **Data Completeness & Schema Checks** → Performing validation to ensure data completeness and schema compliance.
   * This includes record count checks, data type verification, and handling null or inconsistent values.
4. **Docs & Versioning**
   * **Data Documenting & Versioning in GIT** → Documenting data and ingestion processes while tracking changes using **GIT**.
   * This ensures traceability and reproducibility.

This image presents fundamental aspects to consider in a **data architecture project**, grouped into three main categories:

**1. Business Context & Ownership**

This section emphasizes the importance of understanding **who manages the data and what it is used for** within the company:

* **Who owns the data?** → Who is responsible for the data? It may be a specific department, such as Finance or Marketing.
* **What Business Process does it support?** → What business processes rely on this data? Examples include sales, billing, and inventory management.
* **System & Data Documentation** → Is there proper documentation for the system and data? Clear documentation prevents misuse and confusion.
* **Data Model & Data Catalog** → How are the data structured, and what catalogs exist? Understanding data models and catalogs helps with data relationships and usability.

**2. Architecture & Technology Stack**

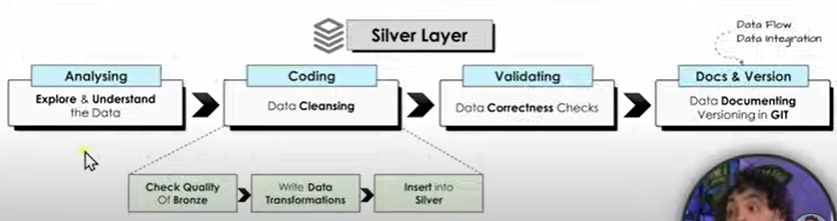
Defines how and where data is stored, as well as its integration capabilities with other systems:

* **How is data stored?** → Where is the data stored? (SQL Server, Oracle, AWS, Azure, etc.). The choice of platform affects performance, security, and scalability.
* **What are the integration capabilities?** → How does the data integrate with other systems? (APIs, Kafka, file extractions, direct database connections). This is crucial for ensuring smooth data flow between systems.

**3. Extract & Load (Data Extraction and Loading)**

This section defines how data is extracted from its sources and loaded into the storage or processing system:

* **Incremental vs. Full Loads** → Is data extracted in full batches or incrementally? Incremental loads optimize resources.
* **Data Scope & Historical Needs** → How much historical data is needed? Defining this prevents unnecessary data storage.
* **What is the expected size of the extracts?** → What is the estimated size of the extracted data? This impacts storage and performance.
* **Are there any data volume limitations?** → Are there restrictions on data volume? Some systems impose limits on query sizes or storage capacity.
* **How to avoid impacting the source system’s performance?** → How can we minimize the impact on the source system? Strategies include efficient queries, indexing, and limiting extraction frequency.
* **Authentication and authorization** → Security measures for data extraction (tokens, SSH keys, VPNs, IP whitelisting).



**Detailed Explanation of the Image**

The image represents the **Silver Layer** within a data pipeline, showing the key steps that take place in this stage. It focuses on transforming, validating, and documenting data from the **Bronze Layer** before passing it to higher layers (likely the **Gold Layer**).

The image is divided into four main sections:

1. **Analysing**
   * Data is explored and understood.
   * Patterns, anomalies, and issues from the Bronze Layer data are identified.
   * Necessary transformations are defined.
2. **Coding (Data Cleansing)**
   * Data cleaning and transformation processes are applied.
   * Inconsistent, duplicate, or null values are removed.
   * Data is restructured to make it more useful.
3. **Validating (Data Correctness Checks)**
   * Quality controls and validation processes ensure the data is correct.
   * Data is compared against predefined rules.
   * Consistency is verified before inserting it into Silver.
4. **Docs & Version (Documentation and Versioning in GIT)**
   * Changes and transformations applied to the data are documented.
   * Scripts and pipelines are versioned in **GIT**.
   * A change history is maintained for audit and traceability purposes.

**General Process in the Silver Layer**

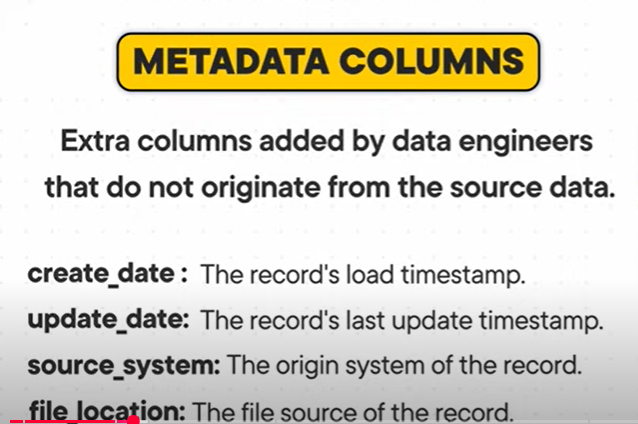
The bottom part of the image describes the general workflow in this layer:

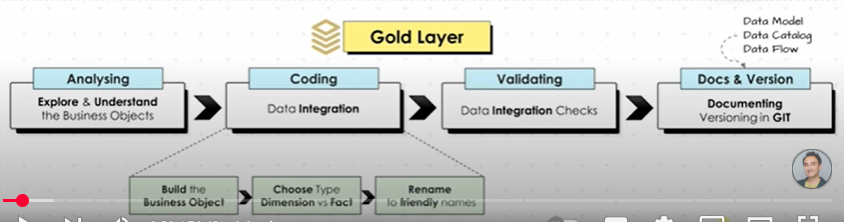
1. **Check Quality of Bronze**
   * The quality of the data in the Bronze Layer is assessed.
   * Necessary transformations are identified.
2. **Write Data Transformations**
   * Data transformations are applied based on the previous step.
3. **Insert into Silver**
   * The transformed and validated data is stored in the Silver Layer.

**Overall Context**

The **Silver Layer** is a crucial stage in a data pipeline because it converts **raw (Bronze) data** into **cleaner, structured data**, ready for advanced analysis or further processing in the **Gold Layer**.

The image also mentions **Data Flow and Data Integration**, suggesting that this process is part of a larger pipeline where different systems and data sources are connected.





La imagen representa un flujo de trabajo dentro de la **"Gold Layer"** en un proceso de modelado y gestión de datos. Se centra en la integración, validación y documentación de datos para análisis empresarial.

**Estructura del flujo de trabajo**

Se compone de cuatro etapas principales, organizadas en un flujo secuencial de izquierda a derecha:

1. **Analysing (Análisis)**
   * **Propósito:** Explorar y comprender los objetos de negocio.
   * **Tareas clave:**
     + Construcción del objeto de negocio.
     + Elección del tipo de dato (**Dimensión vs. Hecho**).
     + Asignación de nombres amigables para facilitar la interpretación.
2. **Coding (Codificación)**
   * **Propósito:** Integración de los datos.
   * **Tarea clave:** Implementación de la integración de datos, asegurando que la estructura y relaciones sean correctas.
3. **Validating (Validación)**
   * **Propósito:** Verificación de la integración de datos.
   * **Tareas clave:**
     + Revisar que los datos sean consistentes y estén correctamente relacionados.
     + Validar que la integración respete los estándares definidos.
4. **Docs & Version (Documentación y Versionado)**
   * **Propósito:** Documentar el modelo de datos y gestionar versiones mediante **GIT**.
   * **Tareas clave:**
     + Creación del **modelo de datos**.
     + Registro en el **catálogo de datos**.
     + Definición del **flujo de datos**.
     + Versionado del código y documentación en **GIT**.

**Aspectos adicionales**

* La **Gold Layer** hace referencia a una capa de datos refinados y listos para análisis.
* Se destaca el uso de **GIT** para versionado y documentación del flujo de datos.

Este proceso permite garantizar calidad, estructura y trazabilidad en la gestión de datos empresariales.

La **construcción del objeto de negocio** en este contexto se refiere a la definición y estructuración de los datos clave que representan entidades dentro de un modelo de datos para análisis empresarial.

**¿Qué es un Objeto de Negocio?**

Un **objeto de negocio** es una representación de una entidad real dentro de una organización, como **clientes, productos, ventas o transacciones**. Estos objetos contienen atributos y relaciones con otros elementos del negocio.

**¿Cómo se construye un Objeto de Negocio?**

El proceso de construcción de un objeto de negocio implica los siguientes pasos:

1. **Identificación del objeto**
   * Se define qué entidad representa el objeto (ejemplo: "Cliente", "Pedido", "Producto").
2. **Definición de atributos**
   * Se seleccionan las características clave del objeto.
   * Ejemplo para un **Cliente**: ID, Nombre, Email, País.
3. **Clasificación del tipo de dato**
   * Se determina si el objeto representa una **dimensión** o un **hecho**:
     + **Dimensión:** Son descripciones (Ejemplo: Cliente, Producto, Región).
     + **Hecho:** Representan eventos medibles (Ejemplo: Ventas, Ingresos, Transacciones).
4. **Relaciones con otros objetos**
   * Se establecen conexiones entre entidades (Ejemplo: Un Cliente puede estar asociado a múltiples Pedidos).
5. **Normalización y Nombres Amigables**
   * Se renombra atributos para mejorar la comprensión del modelo.
   * Ejemplo: cust\_id → Customer ID.

Este proceso es clave para que los datos sean **estructurados, comprensibles y reutilizables** en análisis y reportes.