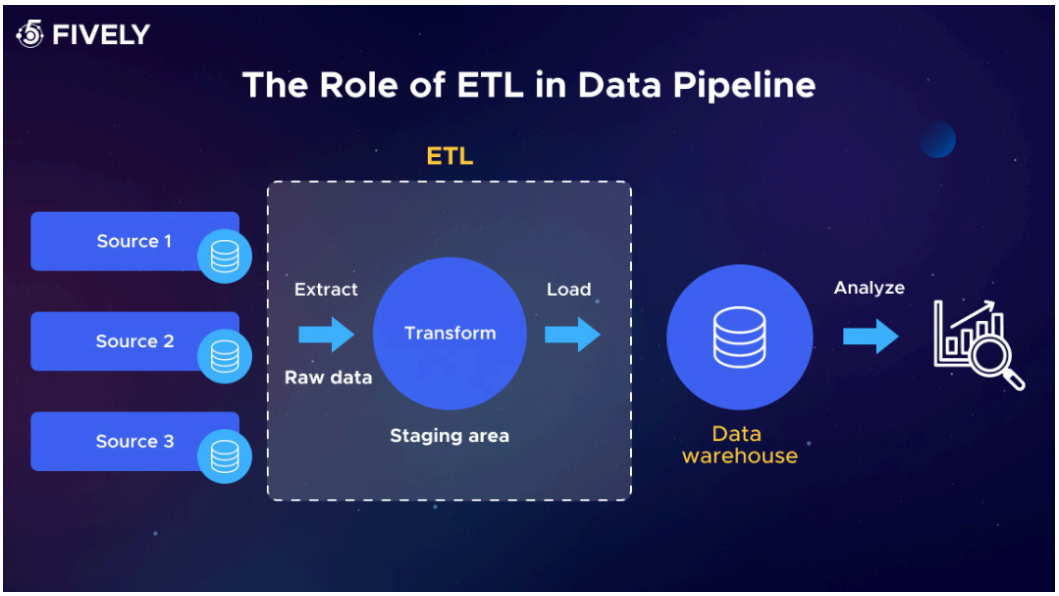
Data Pipeline

A **data pipeline** is a series of processes or tools that automate the movement and transformation of data from one system to another. It's essential for organizations that rely on data analytics, reporting, or machine learning.



What is ETL?

ETL stands for:

1. **Extract** – Pull data from source systems.
2. **Transform** – Clean, reformat, or enrich data.
3. **Load** – Store the transformed data into a destination (typically a **data warehouse**).

🔄 CI/CD for ETL

- Use Git to version transformation code (e.g., dbt models)
- Auto-run tests and deployments on pull requests
- Containerize jobs using Docker + Kubernetes

1️⃣ EXTRACT – Advanced Considerations

This step is about **accessing and collecting data** from source systems. While it seems simple, extraction is one of the **most complex and failure-prone** parts of ETL.

🔧 Key Concepts

- **Incremental Extraction**
  - Only pull data changed since the last run (via `last_updated` timestamp or change data capture).
  - Benefits: faster, less load on systems.

- **Change Data Capture (CDC)**
  - Tracks and captures changes in source data in real time or near real time.
  - Tools: Debezium, AWS DMS, Log-based CDC via Fivetran.

- **API Rate Limiting / Throttling**
  - Many APIs limit the number of requests per minute.
  - Strategy: implement retries, backoff algorithms, or use bulk extract endpoints.

- **Fault Tolerance**
  - Design retries, idempotency (same job run won't corrupt data), and backup strategies.

2️⃣ TRANSFORM – Where the Value Is Created

The transformation layer is where raw data becomes **useful**, **trustworthy**, and **analytics-ready**.

🔧 Transformation Techniques

- **Data Cleaning**
  - Removing duplicates
  - Handling missing/null values
  - Normalizing text, date/time formats

- **Data Enrichment**
  - Adding geolocation from IP addresses
  - Enhancing product data from external catalogs

- **Schema Mapping / Conformance**
  - Standardizing field names and data types across sources
  - Mapping JSON to relational tables (flattening nested structures)

- **Denormalization**

    - Joining multiple normalized tables (e.g., customers + orders) to create wide fact tables for faster BI querying

- **Slowly Changing Dimensions (SCD)**
    - Versioning historical records in dimension tables
    - Types:
        - Type 1: Overwrite
        - Type 2: Add row with timestamp/version
        - Type 3: Track limited history in same row

- **Surrogate Keys**
    - Generate internal keys (e.g., auto-increment ID) for dimension tables instead of using natural keys.

3️⃣ LOAD – Strategic and Performance-Sensitive

Loading is about **putting transformed data into its final destination**, often a **data warehouse** optimized for querying.

🔧 Load Strategies

- **Full Load**
    - Replace all data (common in small datasets or dev environments)
    - Downsides: inefficient, risks downtime

- **Incremental Load**
    - Append new rows or update existing rows
    - Requires good change tracking and deduplication logic

- **Partitioning**
    - Load data in partitions (e.g., by date) for easier management and querying
    - Example: Use `PARTITION BY` in BigQuery or `CLUSTER BY` in Snowflake

- **Upserts / MERGE**
    - Update existing records and insert new ones
    - Supported via `MERGE` SQL statements (in Snowflake, BigQuery, etc.)

🔥 Tools

- **Data Warehouses**: Snowflake, Redshift, BigQuery, Azure Synapse
- **Loading Tools**: Airbyte, Stitch, Fivetran (automated), custom scripts (manual)
- **Orchestration**: Apache Airflow, Prefect, Dagster