Data Engineering 101 - Michaël Bettan

Definition: Data engineering enables data-driven decision making by collecting, transforming, and visualizing data. A data engineer designs, builds, maintains, and troubleshoots data processing systems with a particular emphasis on the security, reliability, fault-tolerance, scalability, fidelity, and efficiency of such systems.

Big Data's 3 Vs Describe the core characteristics that distinguish it from traditional data:

- **Volume** refers to the massive amount of data
- **Velocity** to the speed at which it's generated and processed
- **Variety** to the different formats it comes in (structured, semi-structured, unstructured).

Use Cases:

- Build/maintain data structures and databases
- Design data processing systems
- Analyze data and enable machine learning
- Design for reliability
- Visualize data and advocate policy
- Model business processes for analysis
- Design for security and compliance

Data engineers build data pipelines to enable data-driven decisions:

- Get the data to where it can be useful
- Get the data into a usable condition
- Add new value to the data
- Manage the data
- Productionize data processes
- Understand how to manipulate data formats, scale data systems, and enforce data quality and security.

Concepts

- **Durability**: Ensures that data remains intact and persistent, even in the event of system failures such as hardware crashes or power outages. It guarantees data integrity and the ability to recover information, safeguarding it from loss.
- **Availability**: Guarantees that data is always accessible when needed. Systems are built with redundancy and fault tolerance to minimize downtime, ensuring continuous and reliable access to information.
- **Low Latency**: minimal delay between a data request and its response. Low latency is critical for real-time or near real-time data processing, especially in time-sensitive applications like online gaming and financial transactions.

OLTP vs OLAP

- **OLTP (Online Transaction Processing)**: Focuses on high-volume, short transactions, prioritizing data integrity and consistency (e.g., banking transactions, e-commerce orders). ACID properties are crucial. Normalized schemas are common.
- **OLAP (Online Analytical Processing)**: Designed for complex queries, aggregations, and analysis over large datasets. Speed and flexibility for analytical queries are prioritized. Denormalized schemas (star/snowflake) are common.

Row-level vs Column-level

RDBMS: row-based storage, all the fields (columns) for a given row are stored together. However, in columnar storage, the data for each column is stored independently.

- **Efficient Querying:** Since data in each column is stored separately, queries that access only specific columns can be executed much faster because BigQuery doesn't need to read entire rows into memory—just the relevant columns. For example, if you have a table with 10 columns but your query only references 2 of them, BigQuery only reads and processes those 2 columns, which reduces the amount of data read from storage.
- **Better Compression:** Columns often have repeating or similar values (like IDs, dates, etc.), making them highly compressible. By storing each column separately, BigQuery can apply more efficient compression algorithms, further reducing storage costs.
- **Optimized for Analytics**: Columnar storage is particularly effective for analytical workloads where large datasets are scanned for a few fields (columns) to produce reports or insights.

Standard workflow & lifecycle

1. **Collect** raw data, such as streaming data from devices, on-premises batch data, application logs, or mobile-app user events and analytics
2. **Store** the data in a format that is durable and accessible into a data lake or data warehouse
3. **Process or enrich** to make the data useful for the downstream systems for analysis
4. **Analyze:** data exploration, ad-hoc querying and analysis
5. **Empower**: data visualization, sharing and activation to derive business value and insights from data

Collect mechanisms

- **Application**: data from application events, log files or user events, typically collected in a push model, where the application calls an API to send the data to storage (Cloud Logging, Pub/Sub, CloudSQL, Datastore, Bigtable, Spanner)
- **Streaming**: data consists of a continuous stream of small, asynchronous messages. Common uses include telemetry, or collecting data from geographically dispersed devices (IoT) and user events and analytics (Pub/Sub)
- **Batching**: large amounts of data are stored in a set of files that are transferred to storage in bulk. common use cases include scientific workloads, backups, migration (Storage, Transfer Service, Transfer Appliance, BQ Data Transfer Svc.)

Types of data structures

- **Structured data** has a fixed schema (such as RDBMS)
- **Semi-streaming data** has a flexible schema (can vary)
- **Unstructured data** does not have a structure used to determine how to store data → text, video, images, etc.

Data Lake vs Data Mesh

Data Mesh and Data Lake architectural approaches both aim to improve data accessibility and utilization within an organization, but they differ significantly in their philosophies and implementation strategies.

- Centralized vs Decentralized
- Schema-on-read vs Schema-on-write
- IT-centric vs Data products/Domain-centric

Data Lake: Suitable for organizations with a strong central IT team and a need for a cost-effective solution to store large volumes of raw data. Best when exploring data and use cases aren't clearly defined

Data Mesh: Ideal for large, complex organizations with diverse data domains and a need for greater agility and data ownership. Requires a significant cultural shift and investment in training and infrastructure

RDBMS

Relational Database Management Systems allows you to create, update, and manage a relational database**,** and ensure data integrity through ACID properties

- **Atomicity**: All changes in a transaction are treated as a single unit.
- **Consistency**: Data remains consistent before and after a transaction.
- **Isolation**: Multiple transactions don't interfere with each other.
- **Durability**: Once a transaction is committed, the changes are permanent.

Strong consistency: every operation (read or write) guarantees that all transactions are in the same order

Eventual consistency: parallel processes can see changes in different orders or temporarily inconsistent states, but over time, the system guarantees that all processes will converge on the same final state.

Table Format: Data Lake

- ***Data Lake*** *brings together data from across the enterprise into a single and consolidated location → Cloud Storage*
- ***Data Lake solve for these pain points:***
  - *Getting insights across multiple datasets is difficult*
  - *Data is often siloed in many upstream source systems.*
  - *Cleaning, formatting and getting the data ready for useful business insights in a data warehouse*
- ***Data Lake key considerations:***
  - *Can it handle all the types of data that you have ?*
  - *Can it scale to meet the demand ?*
  - *Does it support high-throughput ?*
  - *Is there fine-grained access control to objects ?*
  - *Can other tools connect easily ?*

Choosing the right file format

Data access patterns*: Are you primarily performing analytical queries or row-wise operations?*
Data size and storage costs*: How much data are you storing, and what are your storage budget constraints?*
Schema evolution requirements*: How often do you expect your data schema to change?*
Data quality needs: *How important is schema enforcement and data validation?*
Tooling and ecosystem*: What tools and technologies are you using to process and analyze your data?*

File Format

- ***CSV (Comma-Separated Values)****: A simple text format where each line represents a row, and values are separated by commas. Easy to generate and parse but lacks data type enforcement.*
- ***Parquet****: A columnar storage format optimized for analytical queries. Offers better compression and performance compared to row-based formats like CSV.*
- ***Avro****: A row-oriented format based on schemas. Supports schema evolution and offers better compression and performance compared to CSV.*
- ***ORC*** *(Optimized Row Columnar): Combines aspects of row and columnar storage. Offers good compression, efficient querying, and schema evolution.*
- ***JSON (JavaScript Object Notation):*** *A human-readable format based on key-value pairs. Widely used for web APIs and NoSQL databases. Offers flexibility but can be challenging to query efficiently at scale.*

Choosing the Right Format:

- ***Parquet****: Excellent choice for analytical workloads needing high performance, compression, and schema evolution.*
- ***Avro****: Best for data serialization and streaming applications where schema evolution is paramount.*
- ***ORC****: Good for Hive workloads requiring ACID properties and optimized for columnar access.*
- ***CSV****: Suitable for small datasets, simple data exchange, and human readability. Avoid for large analytical workloads.*
- ***JSON****: Good for web APIs and document databases where flexibility is important. Not ideal for large analytical queries.*

Table Format: Iceberg

Think of Iceberg as a ***high-level management layer*** *for your data lake. Provides essential features that traditional data lakes often lack:*

- ***ACID Properties****: Iceberg ensures atomicity, consistency, isolation, and durability for your operations. Changes are reliable, concurrent operations are handled safely, and your data remains consistent even in case of failures.*
- ***Schema Evolution****: handles schema changes gracefully. Add, remove, or modify columns without rewriting the entire dataset, and it maintains a history of schema versions.*
- ***Hidden Partitioning****: manages partitioning efficiently behind the scenes. You don't need to worry about complex directory structures, making data organization easier.*
- ***Time Travel:*** *tracks the history of your data, allowing you to query past snapshots. This is incredibly useful for auditing, debugging, or reproducing analyses.*

Data Storage*: stored in files using formats like Parquet, Avro, or ORC.*
Iceberg Metadata*: Iceberg adds a layer of metadata that tracks:*

- *The schema of the table*
- *The location of all data files*
- *Partitioning information*
- *The history of changes (snapshots)*

***Querying**: When you query an Iceberg table, it uses the metadata to locate the relevant data files and then reads the data in the underlying format (Parquet, etc.).*

Table Format (Hive)

Data warehouse system built on top of Hadoop for querying large datasets. Designed primarily for batch processing using SQL-like.

***Key Features**:*

- ***ACID Transactions**: offers limited ACID support, primarily through its integration with the ORC format (Hive ACID tables).*
- ***Schema Evolution**: Hive supports schema changes over time (e.g., adding or removing columns).*
- ***Partitioning**: Hive uses directory-based partitioning, which allows for efficient querying on partitioned data.*
- ***Optimized for Analytics**: Hive is optimized for large-scale analytics using batch processing.*
- ***File Format:** Hive works well with formats like ORC and Parquet (though ORC is more tightly integrated with Hive).*

***How it Fits**: with Hive, ORC is particularly optimized for columnar access and ACID properties.*

Table Format (Delta Lake)

Open-source storage layer that provides ACID transactions and scalable metadata handling for data lakes. It is often used with Apache Spark and is optimized for both batch and streaming data.

Key Features:

- ***ACID Transactions**: Delta Lake ensures data integrity by supporting full ACID transactions on top of data lakes.*
- ***Schema Evolution**: Supports schema enforcement and evolution to ensure data consistency.*
- ***Time Travel:** Delta Lake allows you to access historical versions of data (time travel).*
- ***Partitioning:** supports partitioning and automatically manages partitions for efficient querying.*

***File Format**: uses the Parquet file format but extends it with additional metadata for ACID and versioning.*

***How it Fits**: adds robust data management features like ACID transactions, schema evolution, and time travel to Parquet-based data lakes. Ensures data consistency and reliability, making it suitable for both batch and streaming workloads.*

Data Serialization

***Data serialization** is the process of transforming a complex data structure (like an object, array, or tree) into a format that can be easily stored or transmitted and then reconstructed later. Think of it like taking apart a Lego castle and putting the instructions in a box – you can ship it easily, and someone else can rebuild it using the instructions. Examples: JSON, XML, CSV, Protocol Buffer*

Protocol Buffer (protobuf)

Mechanism for serializing structured data in a language- and platform-neutral way. Think of it like a highly efficient and compact way to package information before sending it over a network or storing it in a file. You define how you want your data to be structured once, then use protobuf tools to generate code in various programming languages. This code makes it easy to write and read your structured data to and from a variety of data streams. It's often used for communication between microservices, storing data efficiently, and defining data contracts in APIs.

Data Warehouse

- ***Data Warehouse** stores transformed data in a usable condition for business insights (e.g., BigQuery, Snowflake)*
- ***Data Warehouse key considerations:***
  - Can it serve as a sink for both batch and streaming data pipelines?
  - Can the data warehouse scale to meet my needs?
  - How is the data organized, cataloged, and access controlled?
  - Is the warehouse designed for performance?
  - What level of maintenance is required by our engineering team?
- RDBMS are optimized for **high throughput WRITES** to **RECORDS**. Cloud SQL databases are **RECORD-based storage** -- meaning the entire record must be opened on disk even if you just SELECTed a single column in your query.
- BigQuery is **COLUMN-based storage** that allows for really wide reporting schemas since you can simply read individual columns out from disk.
- **Upstream processes**: Data acquisition, integration, and preparation. Getting data into the data warehouse.
- **Downstream processes:** Data consumption, analysis, and reporting. Getting insights out of the data warehouse.
- Choosing cloud over on-premises:
  - **Cost reduction/Scalability**: Coffer potentially lower upfront costs and elastic scalability, allowing businesses to pay only for resources used and adjust capacity as needed, unlike fixed on-premises infrastructure.
  - **Reduced maintenance/Increased agility**: handle infrastructure maintenance, freeing up IT resources. This also allows for faster deployment and easier upgrades, increasing business agility.
  - **Enhanced accessibility/Collaboration**: enable broader data access and collaboration across geographical locations, supporting remote work and data sharing with partners more easily.

Data Pipeline patterns

- *What if your raw data needs additional processing? ⟶ data processing, usually Dataproc or Dataflow*
- *What if your data arrives continuously and endlessly? ⟶ data streaming, usually Pub/Sub + Dataflow*
- ***Extract, Transformation, Load (ETL)** processes begin with extracting data from one or more data sources. When multiple data sources are used, the extraction processes need to be coordinated. This is because extractions are often time based, so it is important that extracts from*

different sources cover the same time period.

- ***Extract, Load, Transformation (ELT)*** *processes data is loaded into a database before transforming the data*
- ***Extraction, Load (EL)*** *processes do not transform data → when data does not require changes from the source format.*
- ***Change Data Capture (CDC)*** *approach, each change is a source system that is captured and recorded in a data store. This is helpful in cases where it is important to know all changes over time and not just the state of the database at the time of data extraction.*

Batch vs Streaming

Batch streaming *is for bounded, finite datasets, with periodic updates, and delayed processing.*
Choose batch if:

- *High throughput is more important than low latency.*
- *Data is finite and can be processed in large chunks.*
- *Complex data transformations are required.*
- *Periodic processing is acceptable.*

Stream processing *is for unbounded datasets, with continuous updates, and immediate processing. Stream data and stream processing must be decoupled via a message queue. Can group streaming data (windows) using tumbling (non-overlapping time), sliding (overlapping time), or session (session gap) windows.*
Choose streaming if:

- *Low latency is crucial (e.g., fraud detection, real-time analytics).*
- *Data is continuous and unbounded.*
- *Immediate feedback is required.*

Main challenge: out of order in streams

Types of pipelines

- ***ETL =*** *Extract, Transform and then Load*
- ***ELT =*** *Extract, Load then Transform*
- ***ETL Pipelines (Extract, Load, Transform)*** *are usually necessary to ensure data accuracy and quality.*
- ***Data Pipelines transformations and processing*** *at scale, bring system to life with fresh new data available for scalable*
- ***High-level Orchestration workflows*** *Entire workflows, another abstraction layer, coordinate effort at regular or event driven cadence. Responsible for kicking off the data pipeline in the first place with a new file in of the sources.*
- *Data Pipeline may process data from Lake to Warehouse*

Miscellaneous

- **SQL databases** are used for structured data
  - 1 type → relational
- **NoSQL databases** are used for semi-structured data.
  - 4 types: key-value, document, wide-column, graph
- **OLAP workloads** → analytical data
- **OLTP workloads** → transactional data, where schema is highly normalized and joins are performed extensively to get the results
- **Data marts** — siloed copies of some of the data stored in a data warehouse to offload analytic activity and preserve performance. Fulfills the request of a specific division or business function
- **Time-series data** (time-stamped data) is a sequence of data points indexed in time order. Time-stamped is data collected at different points in time. These data points typically consist of successive measurements made from the same source over a time interval and are used to track change over time. Examples:
  - CPU utilization over time
  - Temperature at each minute
  - Stock and commodity pricing during trading day
- **Data orchestration tools**, such as Apache Airflow, focus on orchestration only and structure the flow of data as a series of events. This workflow management tool keeps data flowing through the pipeline as defined by sequential tasks.
- **DataOps**, short for data operations, brings together data engineers, data scientists, business analysts, and other data stakeholders to apply agile best practices to the data lifecycle, from data preparation to reporting with data.
- **OLAP cube** is a data structure used in business intelligence to allow fast analysis of data according to multiple dimensions. It is called a "cube" because it conceptually organizes data in a multi-dimensional space, even though the number of dimensions may be more than three.
- **Data Freshness**: How up-to-date the data is. The required freshness depends on the use case. Real-time analytics needs very fresh data, while historical trends might tolerate older data. Staleness of data is the opposite – how old or out-of-date the data is.
- **Report Performance**: How quickly a report loads and responds to user interaction. Caching, efficient queries, and optimized data structures improve report performance.
- **Serialized**: Data stored as a single string (e.g., JSON, XML). Flexible but requires parsing to access individual values.
- **Unserialized**: Data stored in individual columns with specific data types. Faster for structured queries but less flexible for storing varying data structures.
- **Data Wrangling**: The process of cleaning, transforming, and preparing data for analysis. This often involves handling missing values, converting data types, and restructuring data.
- **Self-Join**: Joining a table to itself. Useful for comparing rows within the same table (e.g., finding employees who report to the same manager).
- **Hot Data**: Frequently accessed data, often kept in fast storage (e.g., in-memory or SSD).
- **Cold Data**: Infrequently accessed data, often archived to cheaper, slower storage.
- **In-Flight Data**: Data that is currently being processed or transferred. Not yet stored in a persistent location.
- **Shard Tables (Sharding)**: Distributing a large table across multiple smaller tables (shards) based on a key. Improves performance for very large datasets.
- **Data Skew**: Uneven distribution of data across partitions or shards. Can lead to performance bottlenecks in distributed systems.

- **Hot Spotting**: A specific partition or shard receiving a disproportionately high volume of requests. Similar to data skew but focuses on access patterns rather than data distribution.
- **Nested and Repeated Fields:** Data structures where fields can contain other fields (nested) or multiple values (repeated). Common in semi-structured data formats like JSON.
- **Shuffling**: A data redistribution process in distributed computing where data is regrouped across nodes based on a key. Necessary for operations like aggregations and joins in systems like Spark and Hadoop. It's a network-intensive operation.
- **Data lineage** is the process of tracing and visualizing the flow of data from its origin to its destination. It documents how data is transformed, combined, and moved throughout a system. Think of it like a family tree, but for data.
- **Data tiering** is a strategy used to optimize storage costs and performance in data warehouses and data lakes. It involves classifying data based on how frequently it is accessed and then placing it into different storage tiers with corresponding costs and performance characteristics.
- **Idempotence** operation can be applied multiple times without changing the result beyond the initial application. Ensuring data consistency and fault tolerance in distributed systems where failures and retries are common.

Data Mesh

Data Mesh promotes **distributed data ownership** while **centralizing governance** and **data discoverability**. It shifts the responsibility for managing, producing, and consuming data to **domain-specific teams** that have the most context for that data.

- **Centralized vs. Decentralized Data Management**:
  - **Centralization** often leads to bottlenecks with IT teams and stale data.
  - **Full decentralization** can result in data silos, duplication, and inconsistency.
- **Data Mesh Principles**:
  - **Domain-Oriented Data Ownership**: Each domain or business unit owns and manages its data, treating data as a product.
  - **Data as a Product**: Data producers ensure the data they create is useful, documented, and accessible via standardized APIs or interfaces.
  - **Federated Governance**: Centralized governance rules ensure compliance, data quality, and discoverability through tools like Dataplex, BigQuery, and Cloud Storage.
  - **Self-Service Infrastructure**: Teams are empowered with tools to manage and consume data, reducing dependencies on central IT teams.
- **Google Cloud's Role**:
  - **Dataplex** provides centralized data governance while enabling distributed ownership through "virtual lakes."
  - **BigQuery**, **Dataproc**, and **Cloud Storage** are core components for scalable, serverless data storage and processing.
  - **Analytics Hub** is mentioned as an enabler for **data sharing** and creating internal and external **data exchanges**.
- **Data locality:** all data and metadata stay in one region

Need to know

- Difference between a data lake and data warehouse.
- Google Cloud Storage as a data lake solution.
- Getting your data to Google Cloud.
- BigQuery as a data warehouse solution.
- Differences between ETL, ELT and EL.
- Google Cloud reference architectures for ETL, ELT and EL.
- Strong transactional consistency (ACID compliance)
- Hot Spotting
- Sharding
- Data engineers build data pipelines.
- The customers of a data engineer are all the people who make decisions with data.
- T
- ***Vertical** = more compute on single instance (CPU/RAM)*
- ***Horizontal** = more instances (nodes) sharing the load*
- *An INNER JOIN command will limit the scope of a query to improve efficiency.*
- *streaming data: use Pub/Sub and Dataflow*

Data transformation

- **Denormalization** is the process of adding precomputed redundant data to an otherwise normalized relational database to improve read performance of the database.
- **Normalizing a database** involves removing redundancy so only a single copy exists of each piece of information
- **Standardizing**: converting all data types to the same format
- **Cleansing**: resolving inconsistencies and inaccuracies
- **Mapping:** combining data elements from two or more data models
- **Augmenting**: pulling in data from other sources
- **Data deduplication** is a technique for eliminating duplicate copies of repeating data.

Key Concepts

- ***Denormalize data**: Combine data from multiple tables into a single denormalized table to reduce joins.*
- ***Denormalization** often **improves query performance,** but it can **increase storage costs** and complicate updates.*
- ***Normalization** is better for **data integrity and management**, but it can lead to **slower queries***
- ***Deduplication:** process identifies and removes duplicate data entries within a dataset. This improves data quality and storage efficiency, especially when dealing with large volumes of data from various sources.*
- ***Dimension tables**: These contain descriptive attributes that categorize data, like customer demographics, product categories, or time periods. They typically have unique primary keys and are smaller in size.*
- ***Fact tables**: These contain quantitative data points or measurements associated with dimensions, like sales figures, website clicks, or sensor readings. They have foreign keys referencing dimension tables for contextualization.*
- ***Data Marts**: Subject-specific subsets of the main data warehouse tailored for specific user groups or analytical needs.*

- ***ETL (Extract, Transform, Load)****: The process of extracting data from source systems, transforming it into the warehouse format, and loading it into the warehouse.*
- ***Star Schema****: A common data warehouse model using a central fact table with multiple dimension tables.*
- ***Snowflake Schema****: A variation of the star schema with normalized dimension tables.*
- ***Data Modeling****: designing the structure of a data warehouse to meet specific analytical needs.*
- Retries with **Exponential backoff** is a technique that retries an operation, with an exponentially increasing wait time, up to a maximum retry count has been reached

Type of databases

Feature	*Relational*	*Document*	*Wide-Column*
Data Model	*Table*	*JSON/BSON document*	*Sparse multidimensional array*
Schema	*Fixed*	*Flexible*	*Flexible within column families*
Scalability	*Vertical*	*Horizontal*	*Horizontal*
Querying	*SQL*	*Specific query languages*	*Specific query languages*
Transactions	*ACID*	*Limited*	*Limited*
Use Cases	*Transactional systems, structured data*	*Content management, e-commerce, user profiles*	*Time-series data, analytics, large datasets*

Avoid anti-SQL patterns

Self-joins *are used to compute row-dependent relationships. The result is that it potentially squares the number of output rows. This increase in output data can cause poor performance.*
Cross joins *are queries where each row from the first table is joined to every row in the second table, with non-unique keys on both sides.*
Single-Row DML *to treat EDW like an OLTP system. EDW focuses on OLAP system by using table scans and not point lookups.*
Lack of Foreign Keys*: Not enforcing relationships between tables with foreign keys can lead to data integrity issues.*
Using ***Generic Primary Keys*** *(e.g., just an auto-incrementing integer) can make it harder to understand the data and relationships.*

Star Schema

EDW schema that ***organizes data into fact and dimension tables***

- ***Fact Table****: Contains* ***quantitative data (facts****) like sales, revenue, etc., usually representing business transactions or measurements. The fact table holds* ***foreign keys*** *pointing to dimension tables.*
- ***Dimension Tables****: These are smaller tables that* ***describe the attributes*** *of the facts. For instance, a "Sales" fact table might link to dimension tables like "Date", "Customer", "Product", and "Store", which contain descriptive information about those categories (e.g., product names, customer demographics, store locations).*

Key Features:

- ***Denormalized****: Dimension tables are not normalized, leading to some data redundancy.*
- ***Simple Joins****: Easier to understand and query since each dimension table is directly related to the fact table.*
- ***Faster Query Performance****: Ideal for read-heavy operations like reporting or data analysis.*

Snowflake Schema

Extension of the Star Schema, where dimension tables are further normalized. More complex structure that resembles a snowflake.

- ***Fact Table****: Similar to the star schema, it contains numerical measures and foreign keys.*
- ***Normalized Dimension Tables****: The main difference is that dimension tables in a Snowflake Schema are* ***normalized into multiple related tables.*** *For example, instead of a single "Product" dimension table, it might be split into multiple related tables like "Product Category", "Product Subcategory", and "Product Details".*

The structure is more complex:

- ***Central Fact Table*** *at the center.*
- **Dimension Tables** are further broken down into sub-tables

Key Features:

- **Normalized**: Dimension tables are split into smaller, normalized tables to reduce data redundancy.
- **Complex Joins**: Queries require more joins as data is spread across more tables.
- **Storage Efficiency:** Less data redundancy, which saves storage space, but at the cost of more complex queries.

Streaming semantics

Maintaining strong streaming semantics in a distributed stream processing system means ensuring that data processing is accurate, consistent, and reliable even in the face of failures like network partitions, machine crashes, or software bugs.

- ***Exactly-Once Semantics****: guarantees that each record in the input stream is processed exactly one time by the application. This is incredibly difficult to achieve in a distributed system, as it requires complex coordination between various components. Approximating exactly-once is often the practical approach.*
- ***At-Least-Once Semantics:*** *guarantees that each record in the input stream is processed at least once by the application. While easier to implement than exactly-once, it can lead to duplicate processing of some records if failures occur during processing. This requires applications to be idempotent—meaning they can handle repeated processing of the same data without producing incorrect results.*
- ***At-Most-Once Semantics:*** *guarantees that each record is processed at most once. This is the weakest guarantee, as records may be lost during processing. Typically unacceptable for most applications that require accurate results.*

Data Build Tool (DBT)

SQL-first transformation workflow that lets teams collaboratively deploy analytics code following software engineering best practices like modularity, portability, CI/CD, and documentation.

***Pros**:*

- *Strong community & extensive documentation.*
- *Transformation-focused, excels at complex logic.*
- *Great for modularity and code reuse.*
- *Wider range of supported data warehouses.*
- *Open-source core with paid cloud offering.*

***Cons**:*

- *Steeper learning curve.*
- *Less focus on orchestration and scheduling.*
- *Limited built-in testing capabilities*