# Buildables Week 3

**Github Repo:** https://github.com/afnank070/Data-Engineering-Buildables-Fellowship

## Exercise 1: Data Warehouse, Data Lake, Data Lakehouse

**Data Warehouse:** A data warehouse is a central system that stores and organizes data from different sources in one place. It helps businesses analyze past and current data to make better decisions.

**Data Lake:** A data lake is a central storage system that keeps large amounts of raw, structured, and unstructured data. It is flexible and cost-effective, allowing advanced analytics and machine learning on data from many sources.

**Data Lakehouse:** A data lakehouse is a modern storage system that combines features of data lakes and data warehouses. It stores all types of data and supports analytics, machine learning, and real-time processing in one place.

**Reference Website:** https://www.striim.com/blog/data-warehouse-vs-data-lake-vs-data-lakehouse-an-overview/

## Exercise 2: Star Schema

# Exercise 3: ETL vs ELT

| Aspect | ETL (Extract → Transform → Load) | ELT (Extract → Load → Transform) |
|---|---|---|
| **Pros** | Data is cleaned and shaped before going into the warehouse; reliable approach that has been used for many years. | Can deal with very large data; makes use of cloud warehouse power for faster processing; no need for a separate ETL server. |
| **Cons** | Slower when data is huge; needs extra systems for transformations; not great for unstructured data. | Can be expensive if the warehouse compute is high; depends fully on the cloud system; not suitable for old on-prem systems. |
| **Tools** | Talend, Informatica, AWS Glue, Microsoft SSIS. | dbt, Snowflake, BigQuery, Databricks, Azure Synapse. |
| **When to Use** | Good for on-prem or older data setups; when rules need data cleaned before storage; common in industries with strict regulations. | Best for cloud data platforms; when working with very large datasets; useful when it's easier to transform after loading. |

**Reference Website:** https://rivery.io/blog/etl-vs-elt/

# Exercise 4: Batch vs Streaming Processing

**Data Ingestion**

Batch processing gathers data over a period and processes it in large chunks, whereas stream processing handles data continuously in real time as soon as it is received.

**Processing Time**

Since batch processing deals with massive volumes of data in one go, it usually requires more time to complete. Stream processing, on the other hand, emphasizes immediate handling, ensuring that data is processed on the fly without building up in storage.

**Latency**

Batch processing naturally involves some delay, as the data is only processed at defined intervals. Stream processing avoids such intervals, enabling near-instant results with minimal latency.

**Speed**

Batch operations tend to prioritize efficiency and scalability over raw speed, making them suitable for large, high-throughput jobs. By contrast, stream processing is designed to maximize speed by ingesting, analyzing, and delivering results continuously.

**Complexity**

Batch systems are generally simpler to configure and maintain because the operational settings, such as processing intervals, do not need frequent adjustments. Stream processing systems are comparatively more complex since they demand constant data flow management and real-time analytics.

**Use Cases**

Batch processing is best suited for situations where immediate results are not necessary, such as generating daily sales reports, where the analysis is done after data collection. Stream processing, however, is ideal for scenarios that require real-time decision-making, such as fraud detection in banking, where transactions must be validated instantly to prevent potential losses.

**Reference Website:** https://www.astera.com/type/blog/batch-processing-vs-stream-processing/

## Exercise 5: Bronze, Silver, Gold Layer

## Exercise 6: Ingestion Methods

**1. Batch Ingestion**

Batch ingestion collects data over a fixed period of time and loads it into the system in bulk. This approach is widely used when real-time insights are not required, and efficiency in processing large datasets is the main priority. Because the data is processed in groups, it reduces the number of ingestion operations and optimizes system resources.

**Best use cases:** Generating daily sales reports, monthly payroll, or periodic backups.

**2. Streaming Ingestion**

Streaming ingestion captures and processes data continuously as it is produced. Instead of waiting for a scheduled time window, data flows in real time and becomes available for immediate analysis. This method is especially useful in scenarios where rapid decision-making or monitoring is critical.

**Best use cases:** Fraud detection in banking, live website analytics, or IoT sensor monitoring.

**3. Hybrid Ingestion**

Hybrid ingestion combines aspects of batch and streaming. Some data is processed in near real-time, while the rest is ingested in scheduled batches. This approach provides flexibility by balancing the need for real-time insights with the efficiency of batch loads. Organizations often adopt hybrid ingestion when they deal with multiple systems or workloads that require both immediate and periodic updates.

**Best use cases:** E-commerce platforms (real-time cart updates + daily transaction reports), telecom usage data.

| Method | Pros | Cons |
|---|---|---|

| | | |
|---|---|---|
| **Batch** | Efficient for bulk loads; simple to schedule and manage | Data availability is delayed; not suitable for real-time decisions |
| **Streaming** | Provides continuous updates; enables instant insights | More complex setup; higher operational and infrastructure costs |
| **Hybrid** | Offers flexibility by combining immediacy and efficiency | More difficult to design and coordinate; can increase system overhead |

**Reference Website:** https://www.fivetran.com/learn/data-ingestion

## Exercise 7: Quality Dimensions

1. **Accuracy**
   Data should be correct and reliable.
   ***Example:*** A customer's phone number stored without mistakes.
2. **Completeness**
   All required values should be present in the dataset.
   ***Example:*** An order record must include order ID, date, and customer details.
3. **Consistency**
   Data should not conflict across systems.
   ***Example:*** A product price recorded as $50 in one table should not appear as $55 in another.
4. **Timeliness**
   Data should be available and updated when needed.
   ***Example:*** Stock levels in an e-commerce site updated in real-time so customers don't order unavailable products.

**Reference Website:** https://www.collibra.com/blog/the-6-dimensions-of-data-quality

## Exercise 8: Schema Evolution

**Schema evolution** means updating a schema while keeping data usable across versions. Avro is a popular format because it supports both backward and forward compatibility.

**Backward Compatibility:**

- A new schema can still read data written with an older schema.
- **Example:** If you add a new column with a default value, old data can still be read because missing values are filled in with the default.
- Safe changes include:
    1. Adding fields with default values
    2. Removing optional fields
    3. Reordering fields

**Forward Compatibility**

- An older schema can still read data written with a newer schema.
- **Example:** If you add a new optional column, older apps just ignore it.
- Safe changes include:
    1. Adding optional fields
    2. Changing a field to a compatible type (e.g., int → long)
    3. Adding new e-num values

**Reference Website:** https://laso-coder.medium.com/avro-schema-evolution-demystified-backward-and-forward-compatibility-explained-561beeaadc6b

## Exercise 9: CSV vs Parquet

| Aspect | CSV (Comma-Separated Values) | Parquet (Columnar Format) |
|---|---|---|
| **Storage Type** | Row-based storage | Column-based storage |
| **Size** | Larger, no built-in compression | Smaller, supports compression → cost savings |
| **Performance** | Slower on large datasets, full scan required | Faster queries (can skip irrelevant columns/rows) |
| **Tools Support** | Easy to create with Excel, Google Sheets, text editors | Designed for big data tools like Hive, Athena, Spark |
| **Best Use Case** | Small data exchange, simple workloads | Big data, analytics, aggregations, cost-efficient |

**Reference Website:** https://learncsv.com/csv-vs-parquet

## Exercise 10: Partitioning

Partitioning means dividing a large dataset into smaller, more manageable parts based on a specific column (like date, region, or category). It is commonly used in data warehouses and data lakes because it reduces query cost and improves efficiency, since only the needed partitions are scanned instead of the whole dataset.

**Example:**
Suppose we have a sales dataset with millions of rows. If we partition the data by year and month, the folder structure may look like this:

/sales/year=2023/month=01/...

/sales/year=2023/month=02/...

/sales/year=2024/month=01/...

Now, if a query asks for sales in January 2024, only the folder /year=2024/month=01/ will be scanned, making the query much faster and cost-efficient.

## Exercise 11: Orchestration Tool

**Apache Airflow:**

Apache Airflow is an open-source workflow orchestration tool used to schedule and manage data pipelines. It allows developers to define workflows as DAGs (Directed Acyclic Graphs), where each task runs in order and dependencies are clear.

**Airflow is widely used because it:**

- Provides a web UI to monitor and manage workflows.
- Has good integration with cloud platforms and data tools.
- Supports retries, logging, and alerts for failed tasks.

**Reference Website:** https://en.wikipedia.org/wiki/Apache_Airflow

## Exercise 12: Horizontal vs Vertical Scaling

**Distributed computing** is the practice of dividing a workload across multiple computers or systems so that tasks can be completed faster, at a larger scale, and with better reliability. Scaling plays a central role in distributed systems because as data and traffic grow, systems must adapt to handle the increased load.

**Why do we need scaling?**

- To handle more users and traffic.
- To maintain performance and response time.
- To ensure availability and reduce downtime.
- To support increasing data storage and processing needs.

**Vertical Scaling (Scaling Up):**
Vertical scaling means upgrading a single server or machine by adding more CPU, RAM, or storage. It is simple to implement and works well for smaller or monolithic applications.

**For example:**

- Upgrading a MySQL server from 16 GB RAM to 64 GB.
- Moving a website from a 2-core VM to an 8-core VM.

***Advantages:*** Easy to manage and increases system power quickly.
***Disadvantages:*** Limited by physical hardware, can cause downtime during upgrades, and the whole system still depends on a single server.

**Horizontal Scaling (Scaling Out):**
Horizontal scaling means adding more servers or nodes and distributing the load between them. This is widely used in cloud and large-scale applications.

**For example:**

- Websites like Netflix and Amazon run multiple servers across regions.
- Auto-scaling in AWS spins up more instances during peak hours.

***Advantages:*** Better performance, high availability, and fault tolerance since failure of one server doesn’t stop the system.

***Disadvantages:*** More complex architecture, harder to maintain data consistency, requires load balancers and orchestration tools.

**Reference Website:** https://www.geeksforgeeks.org/system-design/system-design-horizontal-and-vertical-scaling/

## Exercise 13: Indexing & Caching in Databases

**Indexing:**

Indexing is a technique used in databases to make data retrieval faster. An index works like the index of a book, instead of scanning every page, you can jump directly to the location of the needed data.

- **Example:** Adding an index on the customer_id column helps quickly find all orders for a customer without scanning the whole table.
- **Advantage:** Faster queries.
- **Disadvantage:** Indexes take extra storage and can slow down insert/update operations because the index also needs to be updated.

**Caching:**

Caching stores frequently accessed data in a temporary storage (memory) so that future requests can be served faster.

- **Example:** A website caching user profiles in Redis or Memcached so the database isn't queried every time.

- **Advantage:** Improves performance and reduces load on the database.
- **Disadvantage:** Cached data can become stale if not updated properly.

**Reference Website:** https://www.geeksforgeeks.org/dbms/difference-between-database-index-and-cache/

## Exercise 14: Compression (Snappy, Gzip, etc.)

Compression is the process of reducing the size of data files so they take up less storage and move faster over the network. Different algorithms are used depending on whether we care more about speed or how small the file gets.

- **Gzip**: Focuses on high compression (makes files much smaller), but it's slower. Good when saving space is more important than speed.
- **Snappy**: Focuses on speed, not maximum compression. Often used in big data systems (like Hadoop, Spark) where reading/writing quickly matters more than saving every bit of space.
- **LZ4**: Very fast compression and decompression, even faster than Snappy in many cases, but results in slightly larger files. Good for real-time systems.
- **Zstd (Zstandard)**: Balances both: it can compress nearly as well as Gzip but much faster, and sometimes even smaller. It's modern and widely adopted in new systems.

**Reference Website:** https://dev.to/konstantinas_mamonas/which-compression-saves-the-most-storage-gzip-snappy-lz4-zstd-1898

## Exercise 15: Data Observability Tool

**Amazon CloudWatch** is a monitoring and observability service for AWS resources and applications. It helps you collect and analyze data (metrics, logs, events, traces) across your infrastructure in near real-time, all from a single platform.

**Purpose:**
The main goal of CloudWatch is to give visibility into the health and performance of AWS resources so that issues can be detected early and fixed quickly, reducing downtime and improving reliability.

**Key Capabilities:**

- **Monitoring & Alarms:** Create alarms that notify you (or trigger automated actions) when thresholds are crossed, e.g., CPU utilization above 70%.
- **Log Analysis:** Collect, explore, and visualize application and system logs to troubleshoot faster.
- **Anomaly Detection:** Use machine learning models to automatically detect unusual patterns.
- **Dashboards & Visualization:** Build dashboards to track trends and performance over time.
- **Automation:** Automate actions based on metrics or events, improving efficiency and response times.

**Reference Website:** https://atlan.com/know/data-observability-tools/#amazon-cloudwatch

# Exercise 16: Compare APIs – REST vs GraphQL

| Aspect | REST | GraphQL |
|---|---|---|
| **Endpoints** | Multiple endpoints, each for a resource | Single endpoint |
| **Data Fetching** | Fixed structure; may over-fetch or under-fetch | Clients request exactly the data they need |
| **Methods** | Uses standard HTTP methods (GET, POST, etc.) | Queries for reading, mutations for updating |
| **Flexibility** | Less flexible; adding fields can break clients | **GraphQL** |
| **Best Use Case** | Simple, stable APIs, easy caching | Single endpoint |

**Reference Website:** https://aws.amazon.com/compare/the-difference-between-graphql-and-rest/

# Exercise 17: A Messaging Pattern

**A messaging pattern** is a way for different parts of a system to communicate asynchronously. Instead of one service waiting for another to finish, messages are sent to a queue or topic and processed independently, which improves scalability and reliability.

**Kafka:**

- Kafka works like a log of events where messages are stored in order.
- Producers send messages to topics, and consumers read them at their own pace.

- It’s commonly used for real-time data streams, like tracking user activity or logs.

**RabbitMQ:**

- RabbitMQ is a traditional message broker where messages are placed in queues.
- Consumers take messages from queues for processing.
- It’s used when task distribution is important, like sending emails or background jobs.

**Pub/Sub (Publish/Subscribe):**

- In Pub/Sub, a publisher sends messages to a topic, and multiple subscribers receive them.
- It decouples senders and receivers, allowing systems to scale easily.
- Useful for broadcasting events to multiple services, e.g., notifications or updates.

**Reference Website:** https://medium.com/career-drill/message-queues-kafka-rabbitmq-pub-sub-system-design-3dcfde74d805

## Exercise 18: Data Contracts

**Data Contracts** are formal agreements between data producers (systems or teams creating data) and data consumers (systems or teams using that data). They define **what data is produced, its structure, format, quality expectations, and frequency**.

### Importance:

**Consistency:** Ensures that all consumers receive data in a predictable format, reducing errors.

**Reliability:** Helps prevent breaking downstream systems when producers change data.

**Collaboration:** Makes responsibilities clear between teams producing and consuming data.

**Governance:** Supports compliance and auditing by documenting expected data behavior.

**Reference Website:** https://atlan.com/data-contracts/

## Exercise 19: Pipeline Diagram

## Exercise 20: Glossary

**Data Warehouse** – Central system storing structured data from multiple sources for analytics.

**Data Lake** – Storage system for raw, structured, and unstructured data, supporting advanced analytics.

**Data Lakehouse** – Modern storage combining features of data lakes and warehouses, supporting analytics and ML.

**Star Schema** – Database schema with a central fact table connected to dimension tables.

**ETL (Extract → Transform → Load)** – Data processing pipeline where data is transformed before loading into the warehouse.

**ELT (Extract → Load → Transform)** – Data processing pipeline where data is loaded first and transformed afterward.

**Batch Processing** – Processing large volumes of data at scheduled intervals.

**Streaming Processing** – Processing data continuously in real-time as it arrives.

**Bronze Layer** – Raw, ingested data in a lake or warehouse.

**Silver Layer** – Cleaned and enriched data ready for analysis.

**Gold Layer** – Aggregated and business-ready data for reporting.

**Batch Ingestion** – Collecting data over a fixed period and loading it in bulk.

**Streaming Ingestion** – Capturing and processing data continuously as it is produced.

**Hybrid Ingestion** – Combination of batch and streaming ingestion for flexible data processing.

**Data Quality Dimensions** – Measures such as accuracy, completeness, consistency, and timeliness to ensure reliable data.

**Schema Evolution** – Updating a schema while maintaining backward and forward compatibility.

**Partitioning** – Dividing a dataset into smaller parts, e.g., by year or month, for efficiency.

**Compression** – Reducing data file sizes using algorithms like Gzip, Snappy, LZ4, or Zstd.

**Data Observability** – Monitoring data pipelines and infrastructure to detect issues, e.g., with Amazon CloudWatch.

**Data Contract** – Formal agreement defining the structure, format, and quality of data shared between producers and consumers.