Week IV

Data Engineering Fundamentals

Intermediate Data Engineer in Azure

Trainer: Balazs Balogh

CUBIX
INSTITUTE OF TECHNOLOGY

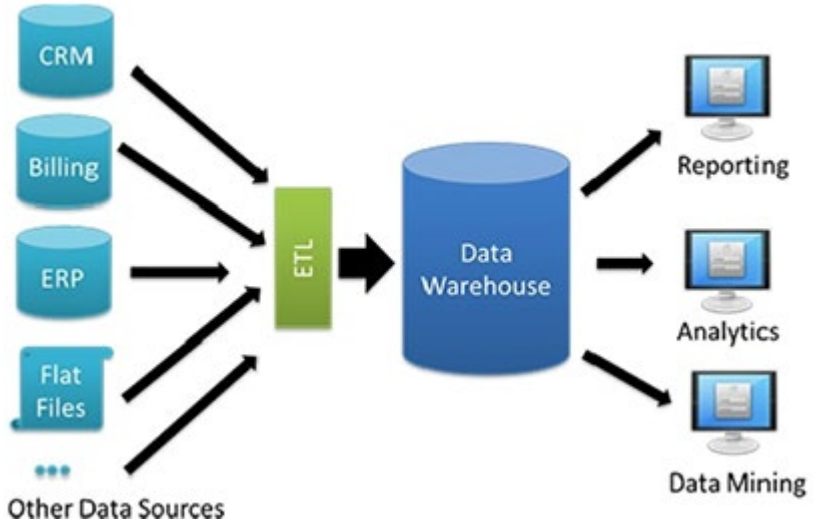
Data Warehouse

What is Data Warehouse?
Data warehousing is a process of **collecting**, **storing**, and **managing data** from various sources to support business intelligence activities.

A data warehouse is a **centralized repository** that stores data from various sources, such as transactional systems, applications, and external sources. It is designed to support business intelligence activities, such as reporting, analysis, and data science. Unlike transactional databases, which are designed for day-to-day operations, data warehouses are designed for complex queries and analysis.

Characteristics of a DWH:

- **Subject-Oriented:** Data in a data warehouse is organized around business functions or subjects, such as sales, finance, and marketing.
- **Integrated:** Data from various sources is integrated and transformed into a common format to ensure consistency and accuracy.
- **Time-Variant:** Data in a data warehouse is stored over time, enabling users to analyze trends and changes over time.
- **Non-Volatile:** Data in a data warehouse is not updated or deleted frequently. Instead, it is updated in batches to maintain data integrity.



Data Warehouse

Benefits:

- **Improved Decision Making:** By providing timely and relevant information, a data warehouse enables users to make informed decisions.
- **Increased Efficiency:** Data warehouses are designed to support complex queries and analysis, enabling users to retrieve data quickly and efficiently.
- **Enhanced Data Quality:** By integrating data from various sources, a data warehouse ensures data consistency and accuracy.
- **Competitive Advantage:** A data warehouse enables businesses to gain insights into customer behaviour, market trends, and competitor activities, providing a competitive advantage.

Cloud Data Warehouses:

- Azure Synapse Analytics
- Amazon Redshift
- Google BigQuery

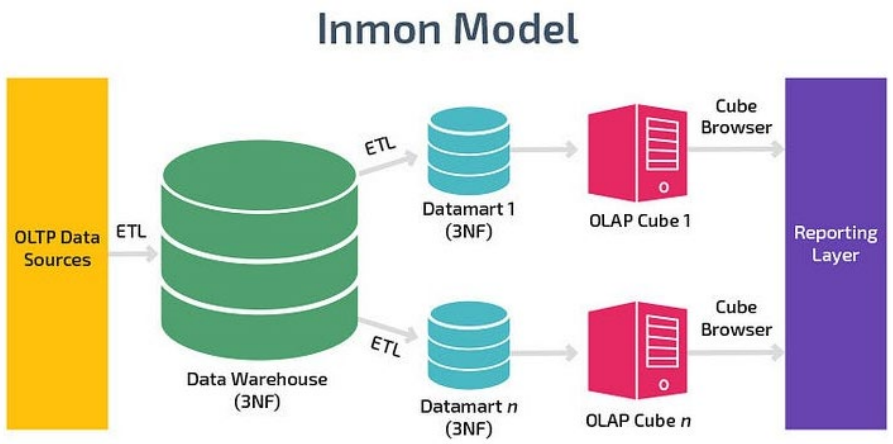
Data Warehouse: Inmon / Kimball

There are two famous techniques in architecture: from Bill Inmon and Ralph Kimball.

Inmon:
His approach involves creating a **centralized enterprise data warehouse** first, which serves as the foundation for any additional data marts.

Key Characteristics:

- **Enterprise-Wide Focus:** The data warehouse is a single, centralized repository for all organizational data, designed for scalability and integration across the enterprise.
- **Normalized Data Structure**: Data in the warehouse is stored in a **3rd normal form (3NF)*** schema to reduce redundancy and improve consistency.
- **Data Marts Derived from EDW**: Individual data marts are created later for specific business needs, pulling data from the central EDW.
- **Data Integration**: Focus on ensuring high-quality, consistent data by integrating it during the warehouse creation process.



***Normal forms in Database Management Systems (DBMS):** Normalizing is the process of minimising redundancy from a relation, or a set of relations. Redundancy can cause insertion, deletion and update anomalies. Normal forms are a series of guidelines to help that the design of the database is efficient and organized, and free from anomalies.

- **First Normal Form (1NF):** A table is in 1NF if all columns contain atomic (indivisible) values, and there are no repeating groups or arrays. One value per cell, unique column names.
- **Second Normal Form (2NF):** It is in 1NF and all non-key attributes are fully functionally dependent on the entire primary key. (Each column should be directly related to the primary key, and not to other columns.)
- **Third Normal Form (3NF):** It is in 2NF and all attributes are only dependent on the primary key, not on any other non-key attribute (no transitive dependency).

Besides these, there are the Boyce-Codd, Fourth, Fifth, and Sixth Normal Forms.

CUBIX
INSTITUTE OF TECHNOLOGY

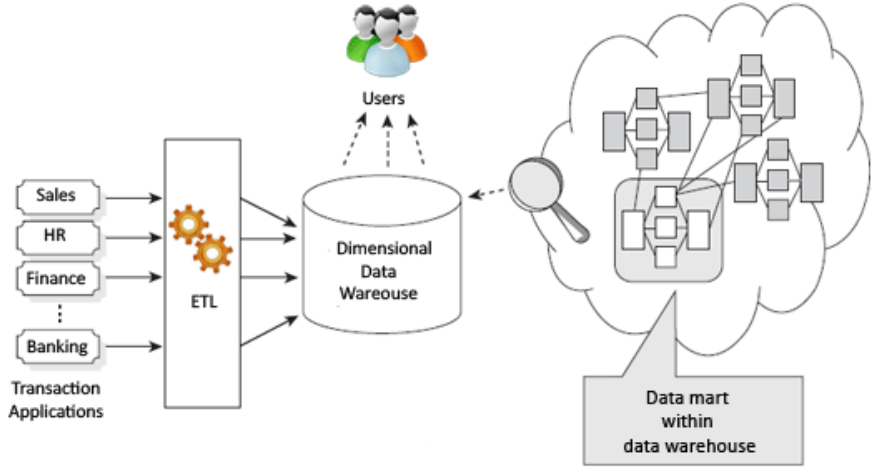
Data Warehouse: Inmon / Kimball

Kimball:
Advocates for a **bottom-up approach**, emphasizing building data marts tailored to specific business processes first, which can later be integrated into a cohesive data warehouse.

Key Characteristics:
- **Business-Process Focus**: Data marts are designed around key business processes (e.g., sales, finance) to meet immediate analytical needs.
- **Denormalized Data Structure**: Uses a **star schema** or **snowflake schema**, which is simpler and optimized for querying.
- **Dimensional Modeling**: Emphasizes the use of facts and dimensions for intuitive data organization.
- **Incremental Implementation**: Individual data marts are created quickly and independently, and over time, they collectively form the data warehouse.

Strengths:
- Faster time-to-value by addressing specific business needs early.
- Easier for business users to understand and query.
- Simpler design and implementation compared to Inmon's approach.

Weaknesses:
- Data integration can become challenging as more data marts are added.
- Risk of inconsistent data across different data marts if not carefully managed.
- Less suitable for complex, enterprise-wide analytics.



CUBIX
INSTITUTE OF TECHNOLOGY

DWH / Data Lake / Data Lakehouse

Data Lake:
A **data lake** is a centralized repository that can store vast amounts of unstructured, semi-structured, and structured data. Data lakes allow you to store raw data in its native format (e.g., JSON, CSV, Parquet, Avro) without needing to transform it first. Data lakes are more flexible than data warehouses, enabling the storage of data from a wide variety of sources, such as logs, social media data, sensors, or multimedia files.

Key Features:
- **Raw Data Storage**: Data lakes can store any type of data, including raw, unstructured, and semi-structured data.
- **Schema-on-Read**: Instead of applying a schema during the data ingestion (as in a data warehouse), a schema is applied when reading the data. This is called schema-on-read.
- **Scalability**: Data lakes are designed to scale to accommodate enormous volumes of data (petabytes or more) by leveraging distributed storage systems like Hadoop HDFS or cloud-based solutions like Amazon S3 or Azure Data Lake.
- **Flexibility**: They support various data types, including logs, video, audio, images, and social media posts, which can be difficult to fit into a structured format in a data warehouse.

Use Cases:
- Storing raw data for future processing.
- Advanced analytics and machine learning (ML) models.
- Storing semi-structured and unstructured data for analysis.
- Data discovery and exploration.

Cloud Data Lakes:
- Azure Data Lake Storage
- Amazon S3
- Google Cloud Storage

DWH / Data Lake / Data Lakehouse

Data Lakehouse:
A **data lakehouse** is a hybrid data architecture that combines elements of both data warehouses and data lakes. It provides the scalability and flexibility of a data lake while adding the structure and performance optimization features of a data warehouse. Lakehouses aim to enable data engineering, analytics, and machine learning within a single platform by allowing users to work with structured, semi-structured, and unstructured data seamlessly.

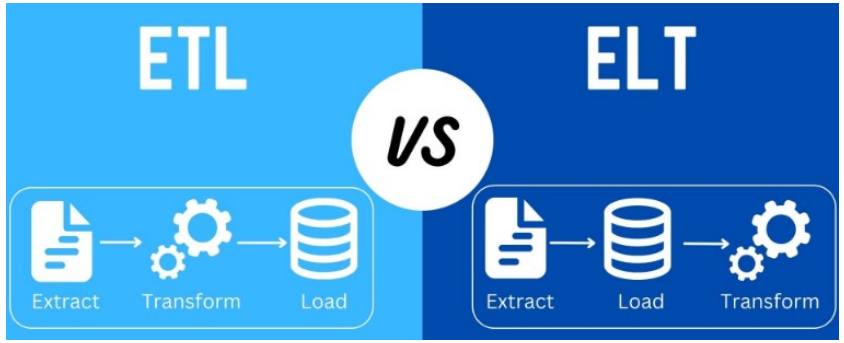
Key Features:
- **Unified Storage**: It can store raw, structured, and semi-structured data, allowing users to run BI queries and ML models on the same dataset.
- **Transactional Support**: Unlike traditional data lakes, lakehouses provide transactional capabilities (ACID transactions), which ensure consistency and reliability.
- **Schema-on-Write and Schema-on-Read**: Lakehouses support both schema-on-write (like in data warehouses) and schema-on-read (like in data lakes).
- **Open File Formats**: Data lakehouses typically use open file formats like Parquet or Delta Lake for storing data, which makes it easier to integrate with various analytics tools.
- **Performance Optimization**: Like data warehouses, lakehouses support features like indexing, caching, and query optimization to improve performance for both structured and unstructured data.

Use Cases:
- Business intelligence and machine learning from both structured and unstructured data.
- Real-time analytics combined with historical reporting.
- Storing raw data while also enabling fast querying for business insights.
- Unified data storage for multiple types of users (data engineers, analysts, ML practitioners).

Cloud Data Lakehouses:
- Azure Synapse / Databricks (Delta Lake)
- AWS Glue
- Google BigQuery

ETL / ELT

Both **ETL** (Extract, Transform, Load) and **ELT** (Extract, Load, Transform) are data integration processes used to transfer data from various sources into a centralized system, such as a data warehouse or data lake. The difference lies in the sequence of operations and the tools used.



Extract, Transform, Load (ETL) involves:
1. Extract: Data is extracted from different source systems (databases, files, APIs, etc.).
2. Transform: Data is cleaned, enriched, and transformed into the desired format or structure for analysis.
3. Load: The transformed data is loaded into a data warehouse or database for querying and reporting.

Use Case: ETL is ideal for traditional data warehouses with structured data, where transformations occur before loading.
Example Tools: Informatica, Talend, IBM DataStage.

Extract, Load, Transform (ELT) involves:
1. Extract: Data is extracted from various source systems.
2. Load: The raw data is loaded directly into a data lake or cloud-based data warehouse.
3. Transform: Transformations are applied within the target system (e.g., SQL queries or Python scripts).

Use Case: ELT works well for modern cloud-based data warehouses (e.g., Snowflake, BigQuery) or data lakehouses, which can handle large volumes of raw data and support in-place transformations.
Example Tools: dbt, Apache Spark, AWS Glue.

OLTP / OLAP

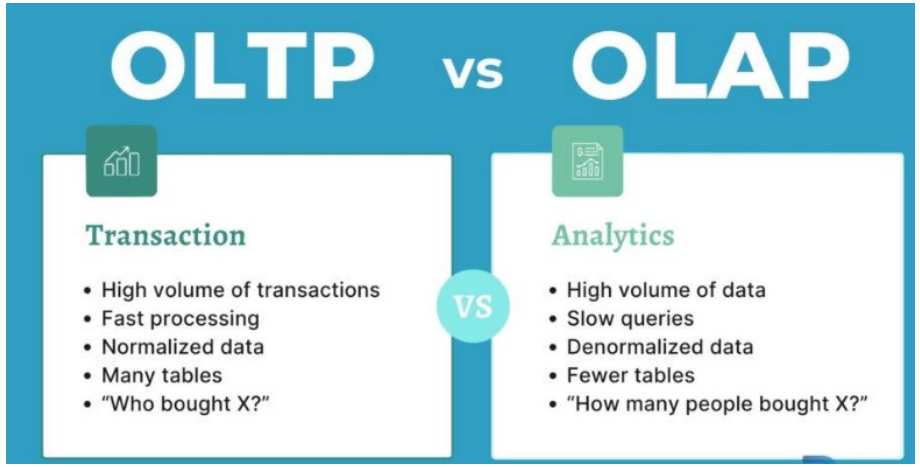
OLTP (Online Transaction Processing) and **OLAP (Online Analytical Processing)** are data processing systems that help you store and analyze business data.

You can collect and store data from multiple sources—such as websites, applications, smart meters, and internal systems. **OLAP** combines and groups the data so you can analyze it from different points of view.
Conversely, **OLTP** stores and updates transactional data reliably and efficiently in high volumes. OLTP databases can be one among several data sources for an OLAP system.

OLTP:
OLTP systems are designed for managing and processing high volumes of short, atomic transactions in real-time. They are used for day-to-day operations in businesses.

Key Characteristics of OLTP:
- **Purpose**: Handles frequent, small, and simple transactions like orders, payments, or inventory updates.
- **Data Structure**: Normalized to reduce redundancy and improve data integrity.
- **Response Time**: Fast, often in milliseconds, to support real-time operations.
- **Users**: Primarily used by front-line employees or automated systems.
- **Data Volume**: Smaller datasets; focused on current data.

Example Use Cases:
- E-commerce systems (e.g., processing orders, payments).
- ATM transactions in banks.
- Inventory management systems.



OLAP:
OLAP systems are designed for complex analytical queries and decision-making processes. They process large volumes of data, often historical, to generate insights.

Key Characteristics of OLAP:
- **Purpose**: Enables reporting, trend analysis, and data visualization.
- **Data Structure**: Denormalized to improve query performance and simplify analysis.
- **Response Time**: Slower than OLTP, as queries are more complex.
- **Users**: Data analysts, business executives, and decision-makers.
- **Data Volume**: Large datasets; focused on historical data.

Example Use Cases:
- Sales trend analysis over multiple years.
- Market segmentation and customer behaviour analysis.
- Financial forecasting and reporting.

CUBIX
INSTITUTE OF TECHNOLOGY

OLTP / OLAP

Key Differences Between OLTP and OLAP:

- **Purpose:** OLTP systems are optimized for processing real-time transactions, whereas OLAP systems focus on analyzing large datasets for insights.
- **Performance:** OLTP systems prioritize high-speed processing for frequent, simple transactions, while OLAP systems are designed to handle complex analytical queries efficiently.
- **Usage:** OLTP systems support routine business operations and daily activities, while OLAP systems enable strategic decision-making and long-term planning.
- **Data Structure:** OLTP systems utilize a highly normalized schema to maintain data consistency, whereas OLAP systems employ a denormalized structure to improve query performance.
- **Data Volume:** OLTP systems manage smaller, granular datasets, while OLAP systems handle significantly larger datasets, often spanning historical data.

Layers of a Data Warehouse

A Data Warehouse typically has several logical and physical layers, each serving a specific purpose in the data processing and storage pipeline.

First layer: Source System Layer
The source system layer is the starting point for a data warehouse, where all the data originates. **This includes various systems** and applications like transactional databases, CRM platforms, and point-of-sale systems.
The primary function of this layer is to **extract data** from different sources.

Second layer: Data Integration Layers
In this layer, raw data from multiple sources is cleaned, validated, and transformed into a consistent format. Here, data is **standardized**, duplicate records are removed, and inconsistencies are resolved.

Third layer: Data Presentation Layer
The data presentation layer is where the **data is made accessible to end-users** in a meaningful format. A **Data Model** is a typical outcome which is a combination of a Fact table and multiple Dimension tables (stars chema / snowflake schema). It supports tools like dashboards, scorecards, and ad-hoc reports for analysis and decision-making.
This layer organizes and structures the data for easy understanding and analysis while ensuring that access is secure, granting users only the data they need.

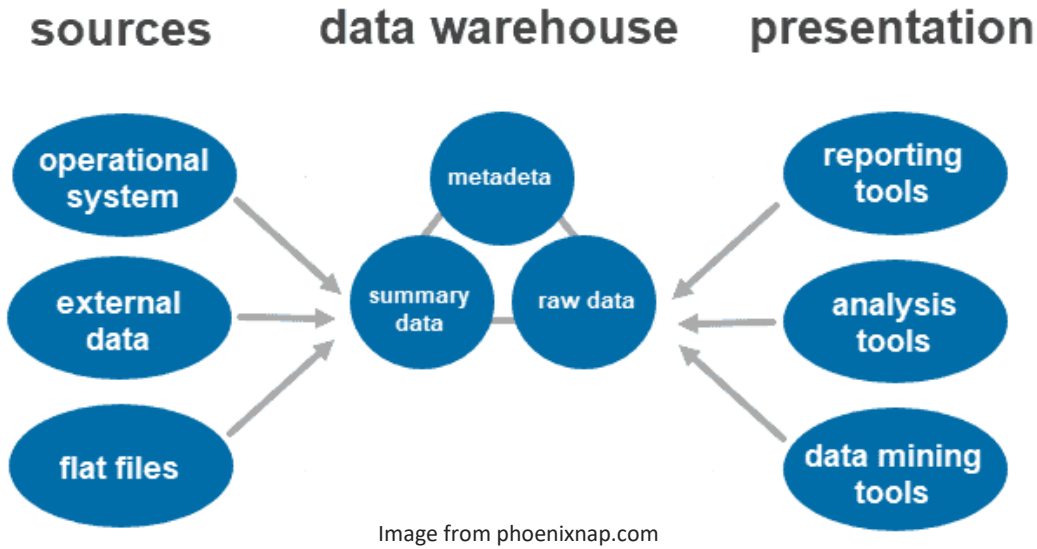


Image from phoenixnap.com

Conclusion
In essence, a data warehouse is built on **three key layers**: the source system layer, the data integration layer, and the data presentation layer. Each layer plays a distinct role in the data warehousing process, working together to enable effective business intelligence and analytics. By using these layers, organizations can design robust solutions to derive insights and drive strategic decisions.

Layers of a Data Warehouse - Databricks

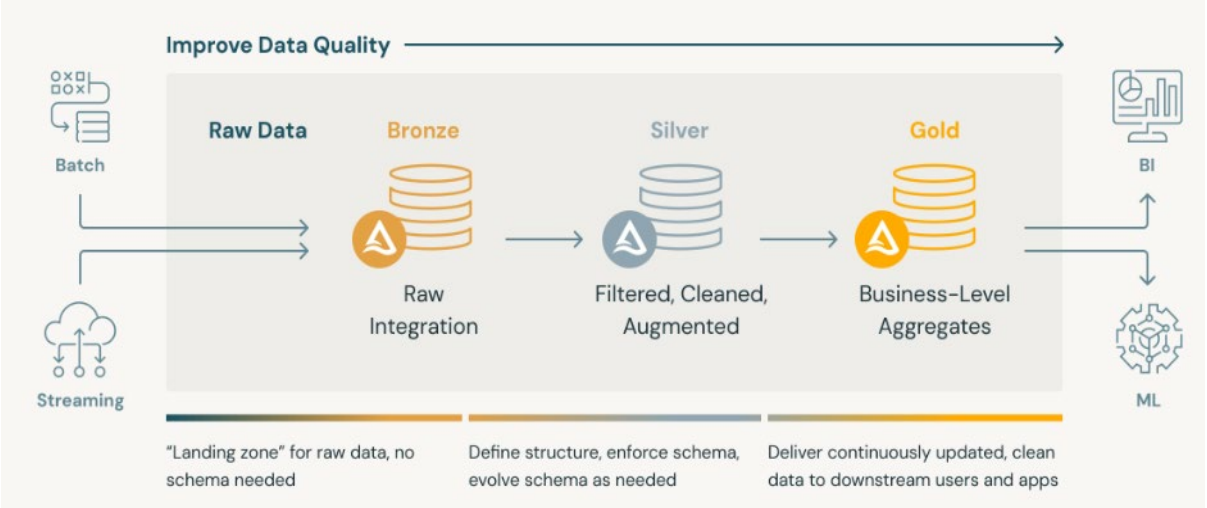
Databricks' **Medallion Architecture** also organizes data into three layers.

The **Bronze layer** stores raw data from source systems "as-is," with metadata for tracking lineage and supporting Change Data Capture, serving as a historical archive for reprocessing and auditability.

The **Silver layer** cleanses, conforms, and integrates data from the Bronze layer into an enterprise-wide view of business entities (e.g., master customers or transactions). It enables self-service analytics, advanced analytics, and ML while prioritizing speed and agility with minimal transformations.

The **Gold layer** contains curated, project-specific datasets optimized for reporting and analytics. This layer uses denormalized, read-optimized models (e.g., star schemas) and applies the final business rules and transformations for actionable insights in areas like customer segmentation, product recommendations, and sales analytics.

The architecture emphasizes flexibility and scalability, supporting advanced analytics and ML across integrated enterprise datasets, which traditional systems often struggle to achieve cost-effectively.



CUBIX
INSTITUTE OF TECHNOLOGY

Star schema / Snowflake schema

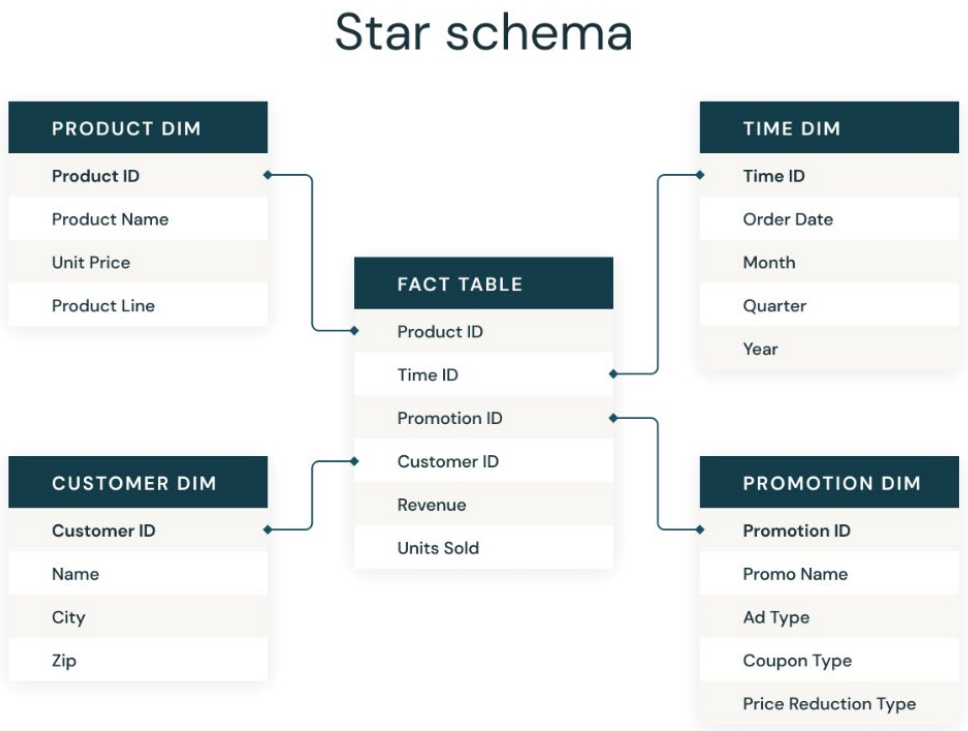
A **star schema** is the simplest form of dimensional modeling. Introduced by **Ralph Kimball** in the 1990s, star schemas are efficient at storing data, maintaining history, and updating data by reducing the duplication of repetitive business definitions, making it fast to aggregate and filter data in the data warehouse.

It consists of:

- **A single fact table** at the center.
- **Multiple dimension tables** radiating outward, resembling a star, directly linked to the Fact table.

Key Characteristics:

- Dimension tables are denormalized (contain redundant data for simplicity – **Product Line = Mobile Phone** in multiple rows in **Product DIM**, this redundancy is normal).
- Queries are straightforward and perform well for analytics.



Star Schema example:

Fact Table:

- Sales

Dimension Tables:

- Product_DIM
- Customer_DIM
- Time_DIM
- Promotion_DIM

Advantages:

- Simple to understand and query.
- Optimized for fast retrieval and aggregation.

Disadvantages:

- Can lead to data redundancy in dimension tables.

CUBIX
INSTITUTE OF TECHNOLOGY

Stars Schema / Snowflake schema

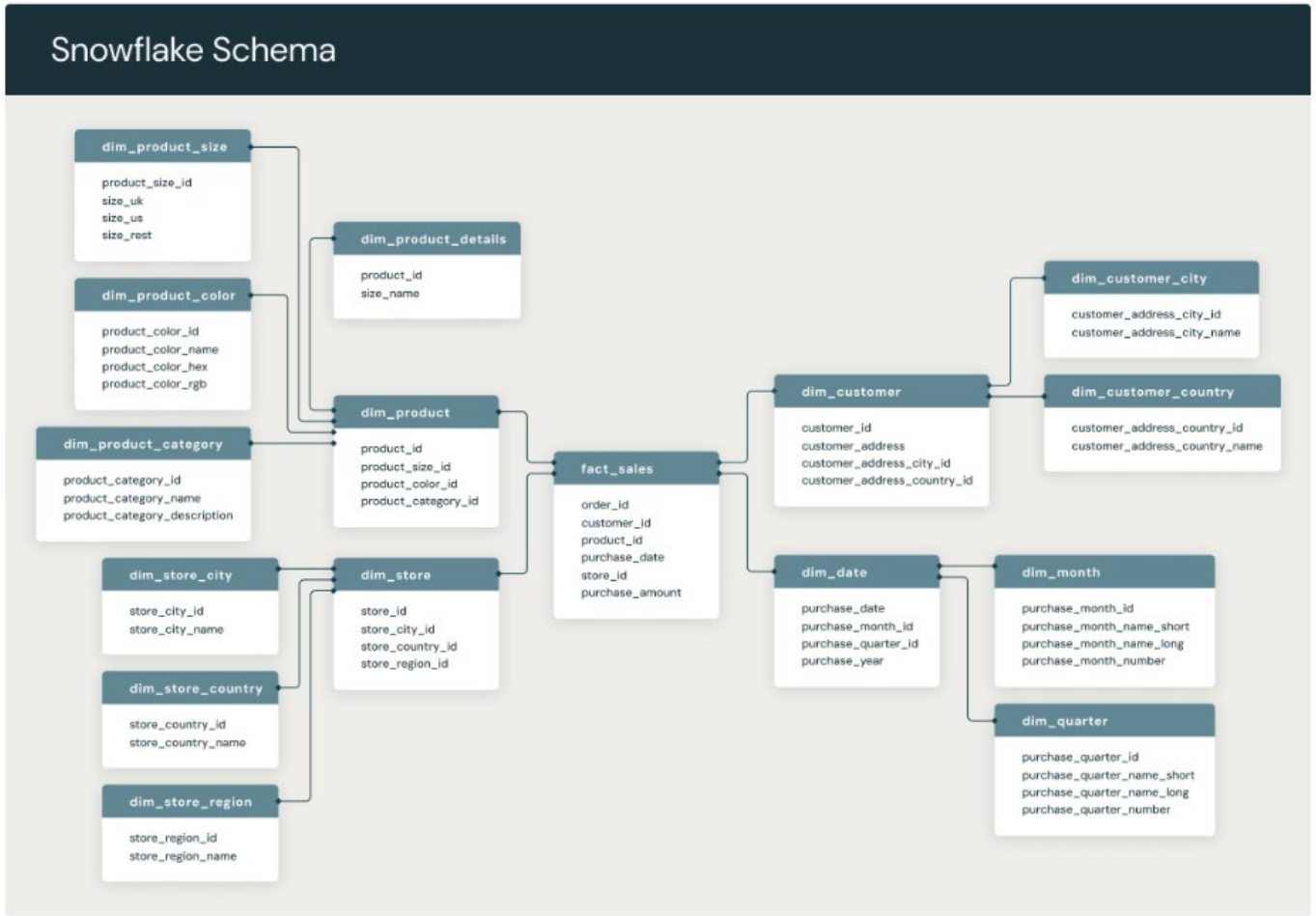
A **snowflake schema** is a more normalized version of a star schema.
In a snowflake schema, engineers break down individual dimension tables into logical subdimensions.
This makes the data model more complex, but it can be easier for analysts to work with, especially for certain data types.

Key Characteristics:
- Dimension tables are normalized to reduce redundancy.
- More complex queries, but saves storage space.

Snowflake Schema example:
Fact table: fact_sales
Dimension tables: dim_*

Advantages:
- Reduces redundancy in dimension tables.
- Saves storage space.

Disadvantages:
- Queries are more complex and can be slower.



CUBIX
INSTITUTE OF TECHNOLOGY

Fact / Dimensional tables

Fact Tables
Fact tables are the central tables in a star schema or snowflake schema that store **quantifiable, numerical data**. These tables represent **measurable business processes** or events, such as sales, orders, or transactions.

Characteristics of Fact Tables

1. Measures/Quantities:
- Contain numeric values (e.g., sales revenue, quantity sold, profit).
- These are the key data points for analysis.

2. Foreign Keys:
- Include foreign keys that link to dimension tables, providing context for the measures.

3. Granularity:
- Define the level of detail in the table (e.g., one row per transaction, per day, or per store).
- A sales fact table might store data at the level of "per transaction."

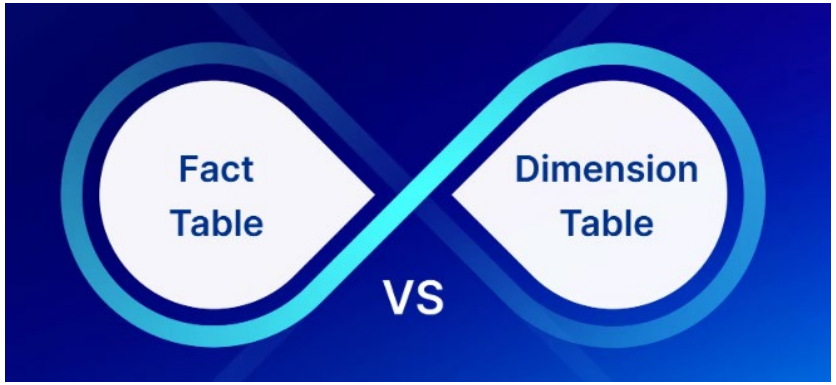


Image from acceldata.io

Fact / Dimensional tables

Dimension tables provide the **context** for the measures in fact tables. They contain **descriptive attributes** that help users analyze the facts.

Characteristics of Dimension Tables

1. Attributes:
- Store textual or categorical data, like product names, customer demographics, or dates.
- Attributes help slice and dice the data in fact tables.

2. Denormalization*:
- Dimension tables are often denormalized for simplicity and performance in queries.

3. Primary Keys:
- Have a unique identifier (primary key) that links to the foreign key in the fact table.

4. Hierarchies:
- Organize data into levels, enabling drill-down analysis (e.g., Year → Month → Day in a Date dimension table).

***Denormalization** in dimension tables refers to intentionally introducing some redundancy (duplicating data) to make the schema simpler and improve query performance. This design choice is particularly important in data warehousing, where the primary goal is to support analytical queries efficiently rather than optimize for storage space or transactional consistency.

Normalized Product data:

ProductKey	ProductName	SubCategoryKey
101	SmartPhone	11

Subcategory table:

SubCategoryKey	SubCategoryName	CategoryKey
11	Mobile Phones	1

Category table:

CategoryKey	CategoryName
1	Electronics

Denormalized Product Dimension:

ProductKey	ProductName	SubCategoryName	CategoryName
101	SmartPhone	Mobile Phones	Electornics

Slowly Changing Dimensions (SCD)

Slowly Changing Dimensions (SCD) are techniques used in data warehousing to manage changes in dimension data over time. These methods ensure historical data accuracy and consistency while reflecting current and past values as needed.
Mostly SCD 1-3 are used.

SCD Type 1: Overwrite

- **Definition**: Updates the existing record with new data, overwriting the old value. No history of previous data is retained.
- **Use Case**: When historical data is not required, and only the latest value is necessary.

Before update:

CustomerKey	Name	City
1	Alice	New York

After update:

CustomerKey	Name	City
1	Alice	London

SCD Type 2: Add historical row

- **Definition**: Retains full history by creating a new row for each change. Includes metadata like StartDate, EndDate, and Current flags to track active records.
- **Use Case**: When tracking historical data is critical for reporting or analysis.

Before update:

CustomerKey	Name	City	StartDate	EndDate	Current
1	Alice	New York	2024-10-01	NULL	1

After update:

CustomerKey	Name	City	StartDate	EndDate	Current
1	Alice	New York	2024-10-01	2025-02-01	0
1	Alice	London	2025-02-01	NULL	1

CUBIX
INSTITUTE OF TECHNOLOGY

Slowly Changing Dimensions (SCD)

SCD Type 3: Add new attribute

- **Definition**: Tracks limited history by adding additional columns to the table for the previous value.
- **Use Case**: When only the previous value of an attribute needs to be stored.

Before update:

CustomerKey	Name	CurrentCity	PrevCity
1	Alice	New York	NULL

After update:

CustomerKey	Name	CurrentCity	PrevCity
1	Alice	New York	NULL
1	Alice	London	New York

SCD Type 4: Add historical table

- **Definition**: Maintains historical data in a separate table while keeping only the current data in the main dimension table.
- **Use Case**: When space optimization is necessary, and historical data is infrequently accessed.

Main table (current data):

CustomerKey	Name	City
1	Alice	London

Historical table:

CustomerKey	Name	City	EffectiveDate
1	Alice	New York	2024-10-01

Slowly Changing Dimensions (SCD)

SCD Type 5: Hybrid of SCD Type 1 and Type 4

- **Definition**: Combines Type 1 for the main dimension table (overwriting current data) and Type 4 for maintaining history in a separate table. It allows the currently-assigned mini-dimension (historical table) attribute values to be accessed along with the base dimension's others without linking through a fact table.
- **Use Case**: When both current and historical data are needed but stored separately for performance.

SCD Type 6: Hybrid of SCD Type 1, 2 and 3

Definition: Combines features of Types 1, 2, and 3 in a single table. Tracks the current value, previous value, and full history in the same table.

- **Use Case**: When both current data, limited historical changes, and full historical changes are needed for analysis.

Table design:

CustomerKey	Name	CurrentCity	PrevCity	StartDate	EndDate	Current
1	Alice	New York	NULL	2024-10-01	2025-02-01	0
1	Alice	London	New York	2025-02-01	NULL	1

Primary / Foreign Key

Primary Keys (PK):
- A **primary key** is a unique identifier for each row in a table.
- Ensures **uniqueness** and is often required for database integrity.
- Example:
  In the Employees table (first table), the **employee_id** column could be the primary key.

Characteristics of Primary Keys:
1. **Unique**: No two rows can have the same primary key value.
2. **Non-nullable**: Cannot contain NULL values.
3. **Immutable**: Should not change once assigned.

Foreign Keys (FK):
- A **foreign key** establishes a relationship between two tables by referencing the primary key in another table.
- Helps maintain **referential integrity** by ensuring that relationships between tables are valid.

Example of Relationship:
Employees table has a **department_id** column (foreign key) that references the **department_id** in the **Departments** table (primary key)

```
+-----------+-------+-------------+
|employee_id|   name|department_id|
+-----------+-------+-------------+
|          1|  Alice|          101|
|          2|    Bob|          102|
|          3|Charlie|          103|
|          4|  David|          101|
|          5|    Eve|          104|
|          6|  Frank|          105|
|          7|  Grace|          102|
|          8|  Helen|          106|
|          9|    Ian|          103|
|         10|   Jack|          104|
+-----------+-------+-------------+

None
+-------------+---------------+
|department_id|department_name|
+-------------+---------------+
|          101|             HR|
|          102|        Finance|
|          103|             IT|
|          104|      Marketing|
|          105|          Sales|
|          107|     Operations|
+-------------+---------------+
```

CUBIX
INSTITUTE OF TECHNOLOGY

Natural / Surrogate key

The **natural key** is a unique key in the table that we choose that best identifies a record in the table.

In the **employees** table, **SSN** serves as the natural key

```
CREATE TABLE employees (
    SSN int PRIMARY KEY,
    FirstName varchar(50),
    LastName varchar(50),
    Department varchar(50),
    DateHired date
);
```

In cases where a single column is insufficient, a **composite primary key** (multiple columns combined) can be used, as long as the combination remains unique and meaningful.

Natural / Surrogate key

A surrogate key is a key that is artificially generated, often as an **auto-incrementing integer**.

These keys are particularly useful when records lack a natural key.
For example, in a **users** table, two individuals could share the same name, birthdate and deparment, or in a log table, two events might occur with identical timestamps.

Unlike primary keys, not all tables require surrogate keys. For instance, in a table listing U.S. states, you likely don't need an ID number. The state abbreviation could function as the natural key.

The **primary advantage** of a surrogate key is its guaranteed uniqueness, which makes it reliable for identifying records. However, the main drawback is its lack of inherent meaning. For example, "30" might represent California, but it carries no context by itself.
In contrast, using "CA" in the State column of an Address table makes it immediately clear which state is being referenced, without needing a lookup in a separate table.

```sql
CREATE TABLE users (
    ID int IDENTITY(1,1) PRIMARY KEY,
    FirstName varchar(50),
    LastName varchar(50),
    Department varchar(50),
    BirthDate date
);
```

Batch and Stream processing

Batch Processing

Batch processing involves processing large datasets at once. The data is collected over time, stored, and then processed in batches.

Key Characteristics:
- **Latency:** Higher latency; results are available after processing a complete batch.
- **Data Size:** Works on large datasets collected over a period.
- **Use Cases:** Suitable for historical analysis, reporting, or scenarios where real-time insights are not required.
- **Processing Model:** Data is processed in discrete chunks (batches).

Examples of Batch Processing:
- **ETL Workflows:** Extracting, transforming, and loading data into a data warehouse.
- **Periodic Reports:** Generating weekly or monthly business reports.
- **Data Cleaning:** Cleaning and aggregating raw data collected over time.
- **Machine Learning Training:** Training models on historical data.

Cloud tools for Batch Processing:
Azure: Data Factory, Synapse
AWS: Glue
Google: BigQuery

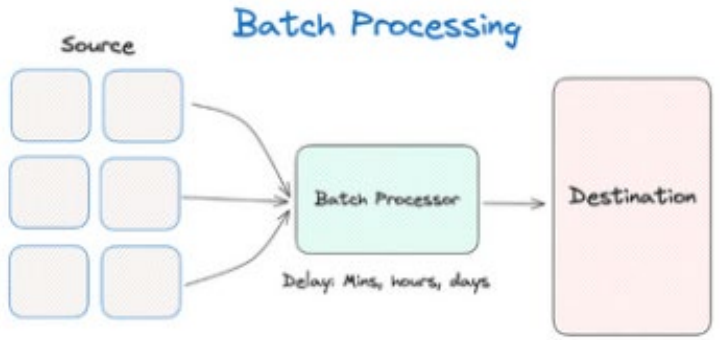


Image from decube.io

Batch and Stream processing

Stream Processing

Stream processing involves processing data in real-time or near real-time as it arrives.

Key Characteristics:
- **Latency:** Low latency; results are available almost immediately.
- **Data Size:** Processes data continuously, often in smaller sizes.
- **Use Cases:** Suitable for scenarios requiring real-time decision-making or immediate actions.
- **Processing Model:** Processes data in a continuous flow or in small micro-batches.

Examples of Stream Processing:
- **Real-Time Analytics:** Monitoring website activity or user behavior.
- **Fraud Detection:** Identifying suspicious transactions as they happen.
- **IoT Applications:** Processing sensor data from connected devices.
- **Log Monitoring:** Analyzing server or application logs in real-time.

Cloud tools for Stream Processing:
Azure: Azure Stream Analytics
AWS: Amazon Kinesis
GCP: Google Cloud Dataflow (stream mode)

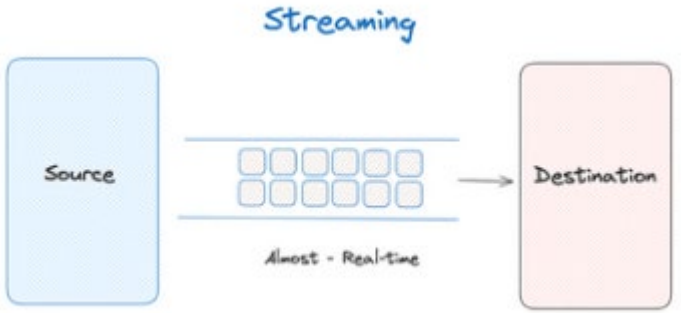


Image from decube.io