Delta-Exam-Academy-Questions

- Delta-Exam-Academy-Questions
  - Test Your Understanding
    - 1. What type of data is stored in a data warehouse?
    - 2. How is an operational system different from a Data Warehouse?
    - 3. What are the layers involved in the Data Warehouse architecture?
    - 4. What are the requirements to implement an EDW architecture?
    - 5. What is EDW Architecture?
      - Enterprise Data Warehouse (EDW) Architecture
  - Planning and Implementation
    - 1. Explain Data Mart?
    - 2. Explain Dependent Data Mart?
    - 3. Explain the difference between Data Mart & Data Warehouse?
    - 4. What are the two approaches to develop an EDW?
    - 5. How do we decide which approach to use while developing an EDW?
    - 6. List the pros and cons of each approach?
    - 7. Explain the top-down approach
    - 8. Explain bottom-up approach
  - Facts and Dimensions
    - 1. Explain Fact?
    - 2. Explain Dimensions?
    - 3. Give real-time examples of Fact and Dimensions?
  - Operational Data Store (ODS)
    - 1. Define an ODS?
    - 2. List the differences between ODS and Data Warehouse?
  - Indexing and Query Optimization
    - 1. Explain the Indexing mechanism?
    - 2. Explain types of Indexes and where they should be implemented?
      - Value-List Index (B+ Tree)
      - Suppose we have a Students table:

        - Bitmap indexes
        - Consider a Students table:
            - Advantages
            - When NOT to Use a Bitmap Index
        - Projection indexes
            - 🔹 A Better Example of Projection Index
        - Bit-Sliced indexes
    - 3. Explain Parallelism?
        - 🔹 1. Interquery Parallelism
        - 🔹 2. Intraquery Parallelism
        - 🔹 3. Mixed Parallelism
        - 🔹 4. Interoperation Parallelism
        - 🔹 5. Intraoperation Parallelism
        - 🔹 6. Pipeline Parallelism
        - 🔹 7. Independent Parallelism
        - 🔹 Summary Table
    - 4. Explain Partitioning types?
- ETL Process
    - 1. Define ETL?
    - 2. Explain ETL Transformations?
        - 1️⃣ Aggregation (Summarizing Data)
        - 2️⃣ Data Sorting (Ordering Data)
        - 3️⃣ Lookup (Fetching Related Data)
        - 4️⃣ Filter (Removing Unwanted Data)
    - 3. List the different ETL tools and their differences?
- Data Modeling and Aggregation
    - 1. Explain how ODS and ETL are interrelated?
    - 2. Explain the need for data modeling?
    - 3. Explain the need for real-time ETL and its challenges?
- Data Extraction and Loading
    - 1. Why is data staging required?
    - 2. What are the types of ETL loads?
        - 1️⃣ Initial Load (First-Time Load)
        - 2️⃣ Incremental Load (Delta Load)

Test Your Understanding

1. What type of data is stored in a data warehouse?

- A data warehouse stores historical, structured, and aggregated data for analytical and reporting purposes.

2. How is an operational system different from a Data Warehouse?

- An operational system handles real-time transactional data, whereas a data warehouse stores historical data optimized for analysis.

3. What are the layers involved in the Data Warehouse architecture?

- Common layers include Source Layer, Staging Layer, Data Warehouse Layer, and Presentation Layer.

4. What are the requirements to implement an EDW architecture?

- Data integration, data modeling, ETL processes, data storage, security, and reporting tools.

5. What is EDW Architecture?

Enterprise Data Warehouse (EDW) Architecture

EDW architecture is a framework for organizing and managing large-scale business data for analytics and decision-making. It typically consists of the following layers:

1. **Source Layer** – Collects data from various operational systems (ERP, CRM, databases, etc.).
2. **Staging Layer** – Temporarily stores raw data for cleansing, transformation, and validation.
3. **ETL Layer** – Extracts, transforms, and loads data into the warehouse.
4. **Data Warehouse Layer** – Stores integrated, historical, and structured data in schemas (Star, Snowflake).
5. **Data Mart Layer** – Subsets of the data warehouse optimized for specific business areas.
6. **Presentation Layer** – Provides access to data for reporting, BI tools, and dashboards.

Planning and Implementation

1. Explain Data Mart?

- A data mart is a subset of a data warehouse focused on a specific business area.

2. Explain Dependent Data Mart?

- A dependent data mart is created from an existing enterprise data warehouse.

3. Explain the difference between Data Mart & Data Warehouse?

- A data warehouse stores enterprise-wide data, while a data mart focuses on departmental needs.

4. What are the two approaches to develop an EDW?

- Top-down approach and Bottom-up approach.

5. How do we decide which approach to use while developing an EDW?

- Based on organizational requirements, budget, and time constraints.

- **The approach to designing a data warehouse depends on:**
  The business objectives of an organization
  Nature of business
  Time and cost involved
  The level of dependencies between various functions

- Insurance: It is vital to get the overall picture with respect to individual clients, groups, history of claims, mortality rate tendencies, demography, profitability of each plan and agents, etc. All aspects are inter-related and therefore suited for the Inmon's approach.

  Marketing: This is a specialized division, which does not call for enterprise warehouse. Only data marts are required. Hence, Kimball's approach is suitable.

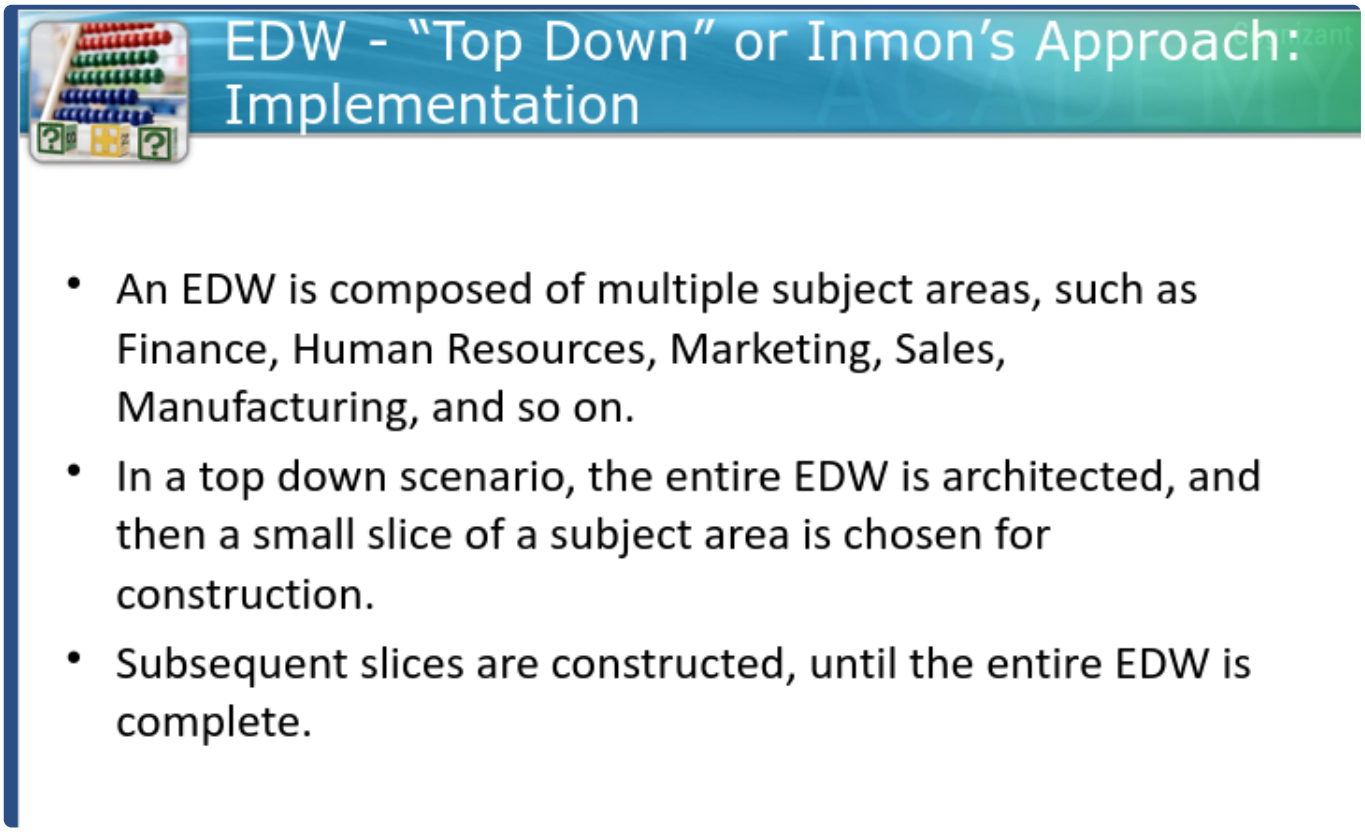
  CRM in banks: The focus is on parameters such as products sold, up-sell and cross-sell at a customer-level. It is not necessary to get an overall picture of the business. Since the scope is limited, you can go for Kimball's method. However, if the entire processes and divisions in the bank are to be linked, the obvious choice is Inmon's design.

  Manufacturing: Multiple functions are involved here, irrespective of the budget involved. Thus, where there is a systemic dependency as in this case, an enterprise model is required. Hence Inmon's method is ideal.

6. List the pros and cons of each approach?

	Inmon	Kimball
Building Data warehouse	Time Consuming	Takes lesser time
Maintenance	Easy	Difficult, often redundant and subject to revisions
Cost	High initial cost; Subsequent project development costs will be much lower	Low initial cost; Each subsequent phase will cost almost the same
Time	Longer start-up time	Shorter time for initial set-up
Skill Requirement	Specialist team	Generalist team
Data Integration requirements	Enterprise-wide	Individual business areas

7. Explain the top-down approach



The pros of a "Top Down" approach are:

- Coordinated environment
- Single point of control & development

The cons of a "Top Down" approach are:

- "Cross everything" nature of enterprise project
- Analysis paralysis
- Scope control
- Time to market
- Risk and exposure

8. Explain bottom-up approach

The **Bottom-Up Approach** in EDW development starts by building **small, independent data marts** for specific business areas (like sales, marketing, or finance). These data marts store data relevant to their respective domains and are built **first**.

Once multiple data marts are developed, they are **combined and integrated** into a centralized **Enterprise Data Warehouse (EDW)**. This approach allows businesses to get **quick insights** without waiting for a massive EDW to be built first.

Think of it like building a **city**. Instead of constructing all the infrastructure at once (Top-Down), we start by developing **small neighborhoods** (Bottom-Up). Over time, roads, bridges, and utilities connect these neighborhoods to form a **well-planned city (EDW)**.

Facts and Dimensions

1. Explain Fact?

- Facts are measurable business metrics, like sales amount.

2. Explain Dimensions?

- Dimensions provide context to facts, such as product or location.

3. Give real-time examples of Fact and Dimensions?

- **Fact:** Number of sales.
- **Dimensions:** Product category, region, time period.

Operational Data Store (ODS)

1. Define an ODS?

> An ODS is usually designed to contain low-level or atomic (indivisible) data (such as transactions and prices) with limited history that is captured "real time" or "near real time" as opposed to the much greater volumes of data stored in the data warehouse generally on a less-frequent basis.

2. List the differences between ODS and Data Warehouse?

- ODS stores near real-time data, while a Data Warehouse stores historical data for analysis.

Indexing and Query Optimization

1. Explain the Indexing mechanism?

- Indexing improves query performance by reducing the time to locate data.

2. Explain types of Indexes and where they should be implemented?

Value-List Index (B+ Tree)

A **Value-List (B+ Tree) Index** is a type of index structure that stores **lists of values associated with a single key** in a **B+ Tree** format.

Suppose we have a `Students` table:

StudentID	Course
101	Math
102	Science
103	Math
104	English
105	Math

A **value-list B+ Tree index on `Course`** would store:

```
Math     → [101, 103, 105]
Science  → [102]
English  → [104]
```

Instead of repeating "Math" three times in the index, we store it **once** with a list of StudentIDs.

Bitmap indexes

A **Bitmap Index** is a specialized type of index used in databases to optimize queries, especially on **low-cardinality columns** (columns with a small number of distinct values). Instead of

storing row references, it **stores bitmaps** (binary arrays) representing the presence or absence of a value in each row.

Consider a `Students` table:

StudentID	Gender
1	Male
2	Female
3	Female
4	Male
5	Female

A **bitmap index on** `Gender` creates two bitmaps:

- **For "Male" (M)** → `1 0 0 1 0`
- **For "Female" (F)** → `0 1 1 0 1`

Each bit position corresponds to a row in the table.

Advantages

✅ **Uses less storage** than B+ Tree indexes for such cases.
✅ **Faster query performance** for analytical workloads (OLAP, data warehousing).

When NOT to Use a Bitmap Index

❌ When data is **highly unique** (e.g., `UserID`, `Email`).
❌ When the table **has frequent updates, inserts, or deletes** (bitmaps are costly to update).

Projection indexes

A **Projection Index** is a type of database index where **only the values of selected columns are stored**, without storing row references (like a traditional index). Instead of indexing **row locations**, it **stores the actual column values** in a compressed, optimized format.

- **A Better Example of Projection Index**

Consider a Traditional Row-Based Table:

SaleID	Product	Price	Customer	Date
1	Laptop	1000	Alice	2024-03-01
2	Phone	500	Bob	2024-03-02
3	Tablet	700	Charlie	2024-03-03

🔹 **Without a Projection Index**, a query like:

```
SELECT Product, Price FROM Sales;
```

Would require scanning the **entire table**, even though only `Product` and `Price` are needed.

🔹 **What Happens with a Projection Index?**

If we create a **projection index on (`Product`, `Price`)**, it would store:

```
Product  | Price
----------------
Laptop   | 1000
Phone    | 500
Tablet   | 700
```

Now, whenever we run:

```
SELECT Product, Price FROM Sales;
```

✅ The database **retrieves the data directly from the index** instead of scanning the entire `Sales` table.
✅ No need to load unnecessary columns (`Customer`, `Date`).
✅ **Faster query performance**, especially for **large datasets**.

🔹 **Where Are Projection Indexes Used?**

✔ **Columnar Databases** (Amazon Redshift, Snowflake, Vertica).
✔ **Data Warehouses** (Star Schema, Snowflake Schema).
✔ **Analytical Queries** (BI, Reporting).

🔹 **When NOT to Use a Projection Index**

❌ When you frequently **update or delete** rows.
❌ When queries involve **lots of joins** (projection indexes are better for simple aggregations).

Bit-Sliced indexes

A **Bit-Sliced Index (BSI)** is a specialized type of index that stores **numeric column values as bitwise slices** across multiple bitmaps. It allows for **fast mathematical operations** (like SUM, AVG, MIN, MAX) without scanning entire tables.

Consider a `Sales` Table:

SaleID	Price (Decimal)	Price (Binary)
1	5	101
2	3	011
3	7	111
4	2	010

Instead of storing full numbers, a **Bit-Sliced Index** stores **each bit separately**:

Bit Position	Slice 2 (MSB)	Slice 1	Slice 0 (LSB)
Row 1 (5 → `101`)	1	0	1
Row 2 (3 → `011`)	0	1	1
Row 3 (7 → `111`)	1	1	1
Row 4 (2 → `010`)	0	1	0

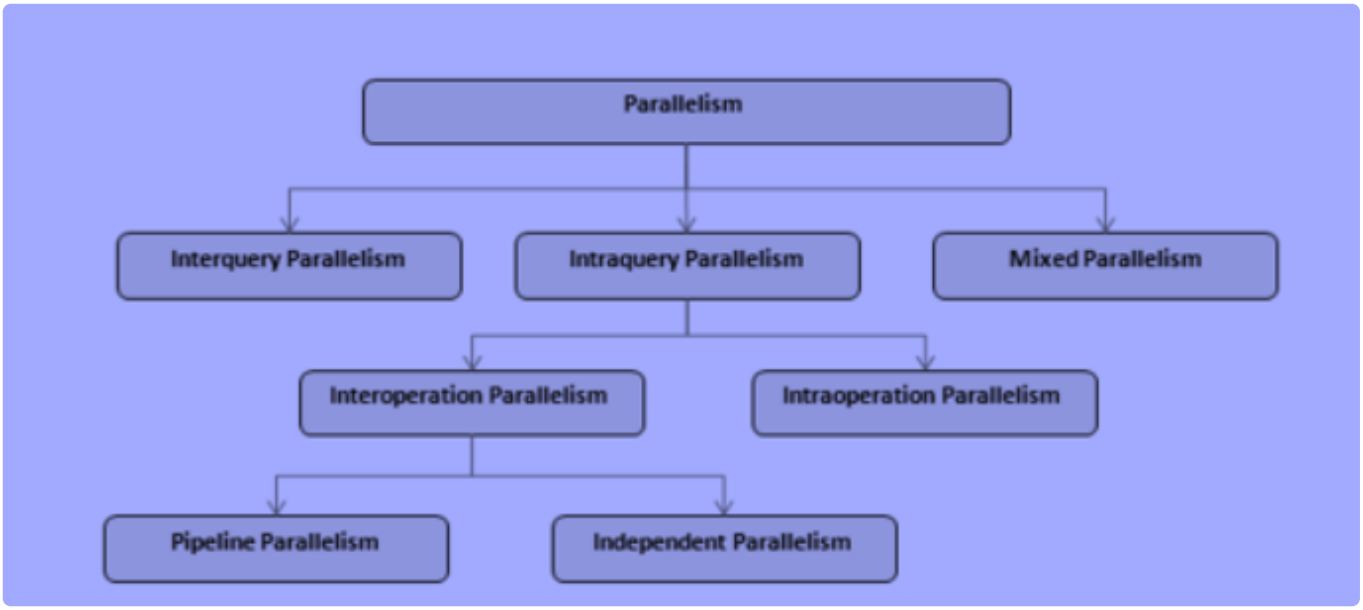
Here, **each column is a separate bitmap** representing a "slice" of the number.

🔹 **Why Use Bit-Sliced Indexes?**

✅ **Super-efficient for numeric aggregations** (SUM, AVG, MIN, MAX).
✅ **Bitwise operations are much faster** than traditional row-based summation.
✅ **Useful in OLAP & data warehousing** (columnar storage).

🔹 **When NOT to Use Bit-Sliced Indexes?**
❌ When the column is **highly unique** (e.g., large primary keys).
❌ When data **changes frequently** (updates can be expensive).
❌ For non-numeric values (works best for **integers**).

3. Explain Parallelism?



Parallelism in databases is a technique used to execute multiple tasks **simultaneously** to improve query performance. It is crucial for **handling large datasets efficiently** and is widely used in **parallel databases, distributed systems, and data warehouses**.

There are different levels of parallelism, which can be classified as:

🔹 **1. Interquery Parallelism**

- **Definition**: Multiple independent queries run **at the same time**.
- **Goal**: Maximizes overall system throughput.
- **Example**:
  - User A runs `SELECT * FROM Orders`
  - User B runs `UPDATE Customers SET status = 'Active' WHERE last_order > '2024-01-01'`
  - Both queries execute in **parallel**, without interfering with each other.

✅ **Best for**: Multi-user databases with heavy workloads.
❌ **Not useful** if a single query is slow.

🔹 **2. Intraquery Parallelism**

- **Definition**: A **single query** is broken into **multiple tasks** and executed **simultaneously**.

- **Goal**: Speeds up execution of a single complex query.
- **Example**:

```
SELECT SUM(sales) FROM Orders WHERE order_date BETWEEN '2023-01-01' AND
'2023-12-31';
```

  - The query is split into **smaller subqueries** (e.g., process January-March, April-June, etc.).
  - Each subquery runs **in parallel** on multiple processors.
  - The final result is **combined** at the end.

✅ **Best for**: Analytical queries in **data warehouses**.
❌ **Overhead** of managing parallel execution.

🔹 3. Mixed Parallelism

- **Definition**: Combines **Interquery** and **Intraquery Parallelism**.
- **Goal**: Improves system-wide performance by **executing multiple queries and parallelizing individual queries**.
- **Example**:
  - Query A (`SELECT AVG(price) FROM Products`) is parallelized (Intraquery).
  - Query B (`UPDATE Customers SET active = TRUE WHERE last_login > '2024-01-01'`) runs simultaneously with Query A (Interquery).

✅ **Best for**: High-performance **data warehouses and OLAP systems**.

🔹 4. Interoperation Parallelism

- **Definition**: Different operations in a query execute in **parallel**.
- **Goal**: Speeds up query execution by running **multiple relational operations** (e.g., `JOIN`, `GROUP BY`, `SORT`) **at the same time**.
- **Example**:

```
SELECT * FROM Orders JOIN Customers ON Orders.customer_id = Customers.id
WHERE Orders.amount > 500;
```

- The **JOIN operation** runs **in parallel** with the **filtering operation (`WHERE`)**.

✅ **Best for**: Queries with **multiple complex operations**.

🔹 5. Intraoperation Parallelism

- **Definition**: A **single operation** (e.g., `JOIN`, `AGGREGATE`, `SORT`) is divided into smaller tasks and run in parallel.
- **Goal**: Improves performance of **resource-heavy operations**.
- **Example**:
  - A **large `JOIN` operation** is **divided across multiple CPUs**.
  - Instead of one process handling all rows, **each processor handles a portion of the data**.

✅ **Best for**: Queries with **large dataset operations**.

🔹 6. Pipeline Parallelism

- **Definition**: The **output of one operation** is passed **directly to the next operation without waiting** for the entire operation to complete.
- **Goal**: **Reduces waiting time** and improves performance.
- **Example**:

```
SELECT SUM(price) FROM (SELECT * FROM Orders WHERE amount > 500);
```

  - The `WHERE` filter executes **while** the `SUM(price)` operation is already processing data.
  - No need to wait for filtering to **fully complete** before aggregation starts.

✅ **Best for**: **Streaming processing** and real-time analytics.

🔹 **7. Independent Parallelism**

- **Definition**: Multiple operations run in **parallel without dependency on each other**.
- **Goal**: Maximizes resource utilization by running **completely independent operations simultaneously**.
- **Example**:
  - `SELECT COUNT(*) FROM Orders` runs in one thread.
  - `SELECT MAX(price) FROM Products` runs in another thread.
  - Since these queries **don't depend on each other**, they can execute in parallel.

✅ **Best for**: Databases handling **multiple independent queries**.

🔹 **Summary Table**

Type of Parallelism	Description	Best Use Case
Interquery Parallelism	Multiple queries run **simultaneously**	Multi-user environments, OLTP
Intraquery Parallelism	A **single query** runs **in parallel**	Large analytical queries, OLAP
Mixed Parallelism	Combines **Interquery & Intraquery** parallelism	High-performance databases
Interoperation Parallelism	**Multiple query operations** run in parallel	Complex queries with `JOIN`, `GROUP BY`
Intraoperation Parallelism	A **single operation** (e.g., `JOIN`, `SORT`) is parallelized	Large dataset processing
Pipeline Parallelism	Output of one operation **immediately feeds into** the next	Streaming analytics, real-time processing
Independent Parallelism	**Unrelated operations** run at the same time	Multi-query workloads

4. Explain Partitioning types?

- Horizontal
  - Range, List, Hash, and Composite Partitioning.
- Vertical

- Normalization
- Code splitting

ETL Process

1. Define ETL?

- ETL stands for Extract, Transform, Load – the process of moving data from source to destination.

2. Explain ETL Transformations?

1️⃣ Aggregation (Summarizing Data)

🔹 **What it does**: Aggregates multiple rows into a **single summary value** (e.g., SUM, AVG, COUNT, MAX, MIN).
🔹 **Why?** Reduces data volume for analytics and reporting.
🔹 **Example**:
We have **Sales Data**:

Product	Region	Sales
Laptop	US	1000
Laptop	US	1500
Phone	EU	500
Phone	EU	700

✅ **ETL Aggregation** (Total Sales per Product):

Product	Total Sales
Laptop	2500
Phone	1200

2️⃣ Data Sorting (Ordering Data)

🔹 **What it does**: Sorts data based on one or more columns.

🔹 **Why?** Required for **better indexing, faster querying, or preprocessing for next steps**.

🔹 **Example**:

Raw **Customer Data**:

CustomerID	Name	SignupDate
102	Alice	2024-01-10
101	Bob	2023-12-15
103	Charlie	2024-02-01

✅ **ETL Sorted Data (by Signup Date)**:

CustomerID	Name	SignupDate
101	Bob	2023-12-15
102	Alice	2024-01-10
103	Charlie	2024-02-01

3️⃣ Lookup (Fetching Related Data)

🔹 **What it does**: Matches values from one dataset with another to **fetch additional details**.

🔹 **Why?** Useful for **enriching data** (e.g., replacing `IDs` with names, fetching customer details).

🔹 **Example**:

We have **Orders Table**:

OrderID	CustomerID	Amount
1	101	500
2	102	700

✅ **ETL Lookup (Get Customer Names from Customers Table)**

OrderID	CustomerID	CustomerName	Amount
1	101	Bob	500
2	102	Alice	700

4️⃣ Filter (Removing Unwanted Data)

- 🔹 **What it does**: **Removes rows** that don't meet certain criteria.
- 🔹 **Why?** Improves **data quality** and **reduces storage and processing costs**.
- 🔹 **Example**:

We have **Raw Sales Data**:

OrderID	Customer	Amount
1	Alice	500
2	Bob	200
3	Charlie	50

✅ ETL Filter (Only Orders Above ₹300)

OrderID	Customer	Amount
1	Alice	500
2	Bob	200

👉 **ETL Tool Example**: Filtering in **Talend, SSIS, Pentaho**

```
SELECT * FROM Sales WHERE Amount > 200;
```

3. List the different ETL tools and their differences?

- Informatica, Talend, SSIS, DataStage – differing in cost, ease of use, and scalability.

Data Modeling and Aggregation

1. Explain how ODS and ETL are interrelated?

- ETL processes extract data from ODS before loading it into the data warehouse.

2. Explain the need for data modeling?

- Data modeling structures data for efficient retrieval and reporting.

3. Explain the need for real-time ETL and its challenges?

- Real-time ETL enables up-to-date insights but requires high processing power.

Data Extraction and Loading

1. Why is data staging required?

- To clean and transform data before loading it into the data warehouse.

2. What are the types of ETL loads?

- Initial Load, Incremental Load, and Full Refresh.

1️⃣ Initial Load (First-Time Load)

- 🔹 **What it does**: Loads **all historical data** into the target system for the first time.
- 🔹 **When to use?**

- When setting up a **new data warehouse**.
- When migrating data from **legacy systems**.
  Impact:
  ✔ Large data movement (**can take hours/days**).
  ✔ Happens **only once**, followed by incremental loads.

2️⃣ Incremental Load (Delta Load)

- 🔹 **What it does**: Loads **only new or changed data** since the last load.
- 🔹 **Why? Faster and more efficient** than full loads.
- 🔹 **When to use?**

- **Daily, hourly, or real-time updates**.
- **Data warehouses that must stay up-to-date**.

Impact:

✔ **Minimizes load time and resource usage**.

✔ **Most commonly used method** for **real-time or frequent data updates**.

3️⃣ Full Refresh (Truncate & Reload)

🔹 **What it does**: Deletes **all existing data** from the target system and **reloads everything from scratch**.

🔹 **When to use?**

- When data **quality issues** occur.
- When a **major data model change** happens.

Impact:

❌ **Risky** (loses all previous data).

❌ **Takes time and uses more resources**.

✔ **Ensures complete data accuracy**.

3. What is data sourcing?

- Data sourcing involves collecting data from multiple systems.

4. Why do we need Change Data Capture (CDC) in ETL?

- To track and process only the modified data instead of the entire dataset.

Data Integrity Issues

1. What are common data integrity problems?

- Inconsistent spelling, duplicate records, missing values, and incorrect manual entries.

1️⃣ Inconsistent Spelling

- **Problem:** Different spellings for the same entity cause confusion.
- **Example:**

CustomerID	Name	Country
101	John Smith	USA
102	Jon Smith	U.S.A
103	J. Smith	United States

👉 **Issue:** The same customer appears with different spellings.

2️⃣ Duplicate Records

- **Problem:** Duplicate data **wastes storage and distorts reports**.
- **Example:**

OrderID	Customer	Amount
201	Alice	$500
202	Bob	$700
201	Alice	$500

👉 **Issue:** Order `201` appears **twice**, leading to **overcounting revenue**.

3️⃣ Missing Values (Null Data)

- **Problem:** Missing critical information makes data unreliable.
- **Example:**

CustomerID	Name	Email
101	Alice	alice@email.com
102	Bob	(NULL)
103	(NULL)	charlie@email.com

👉 **Issue:**

- **Bob's email is missing** → Unable to send promotions.
- **Customer 103 has no name** → Can't track orders properly.

4️⃣ Incorrect Manual Entries

- 🔹 **Problem:** Typos or human errors lead to incorrect data.
- 🔹 **Example:**

EmployeeID	Name	Salary
301	Kevin	$80,000
302	Sarah	$95,000
303	John	8,000,000 (Error!)

👉 **Issue:** John's salary was **entered incorrectly**. This can distort **payroll and reporting**.

2. How can data integrity issues be resolved?

- Data cleansing, standardization, and implementing data validation rules.

Solution	How It Helps	Example Fix
Data Cleansing	Removes duplicates, fills missing values, corrects errors	Remove duplicate orders and fix typos
Standardization	Ensures consistent formats	Convert all country names to "USA"
Data Validation Rules	Prevents incorrect data at entry	Enforce salary range (e.g., $10K - $500K)