Data Engineering Techniques (AID30010) — Unit I & Unit II

Supplementary Question Bank (Missing & Weak Topics)

PART B: LONG ANSWER QUESTIONS (5 marks)

> **How to use this document:** These are questions that were either missing from your existing notes or had insufficient depth. Read these alongside your existing Unit 1 and Unit 2 question banks. Everything here matches the same format and structure as your original notes.

UNIT I — Introduction to Data Engineering and Preprocessing

Q1: Explain the characteristics of Semi-Structured Data with appropriate examples.

Answer:

1. What is Semi-Structured Data?

- Semi-structured data sits between structured and unstructured data
- It has **some degree of organization** but is not as rigidly structured as a relational database table
- Organization is achieved through **tags, markers, or other elements** that define hierarchy within the data
- The order and amount of structuring tags may vary from record to record
- Requires **some pre-processing** before it can be analyzed

2. Key Characteristics

a) Flexible Schema

- Does not follow a fixed schema like relational databases
- Different records in the same dataset can have different fields
- Example: One JSON record may have 5 fields, another may have 8 — both are valid

b) Self-Describing

- The data itself carries metadata (tags or keys) that describe the structure
- You do not need a separate schema definition to understand the data
- Example: In XML, `<name>John</name>` — the tag `name` describes the value `John`

c) Hierarchical / Nested Structure

- Supports parent-child relationships and nesting
- Example: A JSON object for a customer can contain a nested array of orders

d) Partially Organized

- More organized than unstructured data (images, videos) but less rigid than structured data (Excel tables)
- Machines can parse it but require a parser (XML parser, JSON parser)

e) Variable and Sparse

- Not all records need to have the same attributes
- Some fields may be missing in some records without making the data invalid

3. Common Formats

Format	Description
JSON	Key-value pairs, widely used in web APIs
XML	Tag-based, used in web services and config files
HTML	Tag-based, used for web pages
YAML	Human-readable, used for configuration files
Email	Has headers (structured) and body (unstructured)

4. Real-World Examples

Example 1 — JSON (employee records):

```json
{ "emp_id": 101, "name": "Alice", "skills": ["Python", "SQL"] }
{ "emp_id": 102, "name": "Bob", "dept": "HR" }
```

Notice: First record has "skills", second has "dept" — different fields per record.

Example 2 — XML (product catalog):

```xml
```

```
<product>
  <id>P001</id>
  <name>Laptop</name>
  <price>55000</price>
</product>
```

Example 3 — Email:

- Headers (To, From, Date, Subject) = structured part
- Email body = unstructured part
- Together = semi-structured

5. Advantages and Disadvantages

Aspect	Detail
Advantage	Flexible — schema can evolve without breaking existing records
Advantage	Self-describing — easier to share across systems
Disadvantage	Harder to query than structured data
Disadvantage	Requires parsing before analysis

6. Semi-Structured vs Structured vs Unstructured (Quick Comparison)

Feature	Structured	Semi-Structured	Unstructured
Schema	Fixed	Flexible (self-describing)	None
Example	Excel, SQL table	JSON, XML, HTML	Image, Video, Audio
Pre-processing	Minimal	Some	Major

Q2: Explain Nominal and Ordinal Attributes with suitable examples. What are the key differences between them?

Answer:

1. Nominal Attributes

Definition:

- Nominal attributes are categorical attributes where values are **names or symbols representing categories or states**
- Also called **categorical attributes**
- There is **no order or ranking** among the values
- No mathematical operations (addition, subtraction, multiplication) can be performed on nominal data

Characteristics:

- Values represent distinct categories with no inherent sequence
- Each value is simply a label or name
- The only valid operation is checking equality (is A the same as B?)
- Mode is the only meaningful measure of central tendency for nominal data

Examples:

Attribute	Possible Values
Blood type	A, B, AB, O
Hair color	Black, Brown, Blonde, Red
Profession	Doctor, Engineer, Teacher
Marital status	Single, Married, Divorced
Country	India, USA, UK, Germany
Gender	Male, Female, Other

Key point: You cannot say "AB > A" for blood type. The values have no numeric meaning.

2. Ordinal Attributes

Definition:

- Ordinal attributes are categorical attributes where values have a **meaningful sequence or ranking**
- The order between values is known and meaningful
- However, the **exact magnitude of difference** between values is NOT known
- You know that A > B, but you do not know by how much

Characteristics:

- Values can be ranked or ordered

- Differences between ranks are not necessarily equal or quantifiable
- Mode and median are meaningful; mean is generally not meaningful
- Often arise from surveys, ratings, and rankings

Examples:

Attribute	Possible Values (in order)
Customer satisfaction	Low < Medium < High
Academic grade	F < D < C < B < A
Military rank	Private < Corporal < Sergeant < Captain
Spice level	Mild < Medium < Hot < Extra Hot
Movie rating	1 star < 2 stars < 3 stars < 4 stars < 5 stars
T-shirt size	XS < S < M < L < XL < XXL

Key point: We know "High" satisfaction > "Low" satisfaction, but we cannot say High is exactly twice as good as Low.

3. Key Differences — Nominal vs Ordinal

Feature	Nominal	Ordinal
Order among values	No	Yes
Magnitude of difference known	No	No
Mathematical operations	Not possible	Not possible (only ranking)
Example	Blood type, Hair color	Satisfaction rating, Grades
Valid central tendency	Mode only	Mode and Median
Typical use	Classification, labeling	Ranking, surveys

4. Memory Trick

- **Nominal** = Name only (just labels, no order)
- **Ordinal** = Order matters (ranked, but differences unknown)

Q3: Explain the types of Attribute Subset Selection in Data Reduction.

Answer:

1. What is Attribute Subset Selection?

- Also called **Dimensionality Reduction**
- The process of removing **irrelevant or redundant attributes** from a dataset
- Goal: Find the **minimum set of attributes** such that the probability distribution of data classes is as close as possible to the original using all attributes
- Reduces the number of attributes → patterns become easier to understand → algorithms run faster

2. Why is it Needed?

- Many real-world datasets have hundreds or thousands of attributes
- Not all attributes are useful for a given mining task
- Irrelevant attributes add noise and slow down the algorithm
- Redundant attributes contain information already present in another attribute

3. Difference: Relevant vs Redundant Attributes

Type	Definition	Example
Irrelevant attribute	Not useful for the mining task at all	Customer's favourite color when predicting loan default
Redundant attribute	Its information is already present in another attribute	"Age" and "Date of Birth" together

4. Heuristic (Greedy) Methods for Attribute Subset Selection

Method 1 — Stepwise Forward Selection

- Start with an **empty set** of attributes
- At each iteration, add the **single best attribute** (the one that improves the model the most)
- Continue until adding more attributes does not improve the result
- Direction: builds up from nothing

Method 2 — Stepwise Backward Elimination

- Start with the **full set** of all attributes
- At each iteration, **remove the single worst attribute** (the one that contributes least)

- Continue until removing more attributes degrades the result
- Direction: prunes down from everything

Method 3 — Combination of Forward + Backward (Best of Both)

- At each step: add the best attribute AND remove the worst attribute simultaneously
- More thorough but computationally more expensive

Method 4 — Decision Tree Induction

- Build a decision tree on the data
- Attributes that are **NOT used in any branch of the tree** are considered irrelevant
- Only the attributes appearing in the tree are selected
- This is a supervised method (uses class labels)

5. Summary Table

Method	Starting Point	Direction	Type
Stepwise Forward Selection	Empty set	Add best one by one	Greedy
Stepwise Backward Elimination	Full set	Remove worst one by one	Greedy
Combination	Full set	Add best + Remove worst	Greedy
Decision Tree Induction	Full set	Tree-based pruning	Supervised

6. Benefits of Attribute Subset Selection

- Faster data mining algorithms
- Better model performance (less noise)
- Simpler, more understandable models
- Reduced storage requirements

Q4: Explain Data Discretization with a real-world example. Why is it needed?

Answer:

1. What is Data Discretization?

- The process of converting **continuous numerical data** into a **finite set of discrete intervals or categories**

- Instead of working with exact values, data is grouped into ranges called **bins** or **intervals**
- Simplifies large datasets for easier analysis and better model performance

2. Why is Discretization Needed?

- Some classification algorithms (like Naïve Bayes) only work with categorical data
- Continuous data has infinite possible values — discretization makes it manageable
- Reduces the effect of small variations (noise) in data
- Interval labels replace exact values → simpler representation
- Improves efficiency of data mining algorithms

3. Real-World Example — Age Groups

Before Discretization (continuous values): Ages in a dataset: 5, 12, 17, 23, 35, 42, 55, 68, 74, 82

After Discretization (discrete intervals):

Range	Label
0 – 12	Child
13 – 17	Teenager
18 – 35	Young Adult
36 – 60	Adult
61+	Senior

So 5 → "Child", 35 → "Young Adult", 74 → "Senior"

Why this helps: A machine learning model can now treat "Young Adult" and "Senior" as categories rather than computing on raw numbers.

4. Another Real-World Example — Temperature

Before: 18°C, 22°C, 25°C, 31°C, 38°C, 42°C

After:

Range	Label
Below 20°C	Cold
20 – 30°C	Comfortable
Above 30°C	Hot

5. Another Real-World Example — Salary

A bank wants to classify loan applicants. Instead of using exact salary:

Range	Category
Below ₹20,000/month	Low Income
₹20,000 – ₹60,000/month	Middle Income
Above ₹60,000/month	High Income

6. Types of Discretization Methods

Method	Type	Approach
Binning	Unsupervised	Divide into equal-width or equal-frequency bins
Histogram Analysis	Unsupervised	Use frequency histograms to determine breakpoints
Clustering Analysis	Unsupervised	Use clusters as intervals
Decision Tree Analysis	Supervised	Use entropy to find best split points
Chi-Square (χ^2) Analysis	Unsupervised	Merge adjacent intervals with similar distributions

7. Supervised vs Unsupervised Discretization

Type	Uses class labels?	Example
Supervised	Yes	Decision Tree Analysis
Unsupervised	No	Binning, Histogram, Clustering

8. Top-Down vs Bottom-Up

Approach	Method	What it does
Top-down (split)	Binning, Histogram, Decision Tree	Starts with all data; splits into intervals
Bottom-up (merge)	Chi-Square	Starts with many small intervals; merges similar ones

UNIT II — Data Warehousing

Q5: Explain the ETL Process in a Data Warehouse in detail.

Answer:

1. What is ETL?

- ETL stands for **Extract, Transform, Load**
- It is the process by which data is moved from operational source systems into the data warehouse
- ETL is the backbone of data warehousing — without it, raw data from heterogeneous sources cannot be made available for analysis
- ETL runs periodically (daily, weekly) to refresh the warehouse with new data

2. Why ETL is Needed

- Operational databases use different formats, naming conventions, and encodings
- Data from multiple sources must be cleaned and standardized before loading
- The warehouse must contain high-quality, consistent, historical data — ETL ensures this
- Direct querying of source systems for analytics would degrade operational performance

3. Step 1 — Extract

What it does:

- Pulls raw data from multiple **heterogeneous source systems**
- Sources include: relational databases, flat files, ERP systems, CRM systems, online transaction records, external APIs, legacy systems

Key challenges:

- Sources may use different database systems (Oracle, MySQL, PostgreSQL)
- Data may be in different formats (CSV, XML, JSON, binary)
- Extraction must not slow down or disrupt the operational source system
- Only changed or new data may need to be extracted (incremental extraction) vs full extraction

Example: Pulling today's sales transactions from the POS (Point of Sale) system, customer data from the CRM, and product data from the inventory system

4. Step 2 — Transform

What it does:

- The largest and most complex step
- Converts extracted raw data into the format required by the data warehouse
- Applies business rules, cleans data, resolves inconsistencies

Key transformations performed:

Transformation	Description	Example
Data Cleaning	Fix errors, handle missing values	Replace NULL salary with average salary
Format Standardization	Unify formats across sources	Convert all dates to DD-MM-YYYY
Data Type Conversion	Match data types	Convert "1"/"0" strings to TRUE/FALSE
Deduplication	Remove duplicate records	Same customer appearing twice
Aggregation	Summarize data	Calculate monthly totals from daily sales
Encoding Standardization	Unify codes	Male=1/M/Male → standardize to "M"
Currency Conversion	Unify monetary values	Convert USD, EUR, GBP to INR
Key Generation	Create surrogate keys	Replace source system IDs with warehouse-specific keys

Staging Area:

- Transformation happens in a **staging area** — a temporary storage space separate from the warehouse
- Raw extracted data is loaded into staging first, transformed there, then moved to the warehouse
- This protects the warehouse from incomplete or dirty data

5. Step 3 — Load

What it does:

- Loads the cleaned and transformed data into the **data warehouse**
- Populates fact tables and dimension tables

Two types of loading:

Type	Description	When Used
Initial Load (Full Load)	Load all historical data for the first time	First-time warehouse setup
Incremental Load (Delta Load)	Load only new/changed records since the last load	Regular refresh (daily/weekly)

What happens during load:

- Data is sorted, summarized, and consolidated
- Integrity checks are performed
- Indexes are built for fast query performance
- Materialized views are computed and stored
- Partitions are created

6. ETL Architecture Diagram (Simple)

```
Source Systems → [Extract] → Staging Area → [Transform] → [Load] → Data Warehouse → BI Tools
```

7. Key Properties of a Good ETL Process

Property	Description
Accuracy	Data must be correct after transformation
Completeness	No records should be lost during the process
Consistency	Same data must look the same across all tables
Timeliness	Warehouse must be refreshed within an acceptable time window
Auditability	Every transformation step must be logged for traceability

8. ETL vs ELT

Feature	ETL	ELT
Transform step	Before loading	After loading
Where transformation happens	Staging area / ETL tool	Inside the data warehouse
Best for	Traditional warehouses	Cloud-based warehouses (BigQuery, Snowflake)
Speed	Slower for large data	Faster with cloud compute

Q6: Explain Dimensional Modelling. How is it different from ER Modelling?

Answer:

1. What is Dimensional Modelling?

- A **design technique specifically created for data warehouses**
- Developed by Ralph Kimball
- Organizes data into **facts** (what is measured) and **dimensions** (the context of measurement)
- Optimized for **analytical queries and reporting** — not for transaction processing
- The output of dimensional modelling is a **Star Schema, Snowflake Schema, or Fact Constellation Schema**

2. The Two Core Elements

a) Fact Table

- Contains the **measurements or metrics** of the business process
- Each row in the fact table represents one business event (a sale, a patient visit, a flight)
- Contains:
  - **Measures** (the numeric values being tracked): e.g., sales_amount, quantity_sold, patient_count
  - **Foreign keys** to each dimension table

b) Dimension Tables

- Contain the **descriptive context** around the facts
- Answer the "who, what, where, when, how" questions about the fact

- Examples: Time dimension, Product dimension, Customer dimension, Location dimension
- Contain attributes used for filtering, grouping, and labeling in reports

3. The Four Steps of Dimensional Modelling (Kimball's Process)

Step 1 — Choose the Business Process

- Identify what process you want to model
- Examples: Sales transactions, hospital admissions, flight bookings, university course enrollments
- Each process becomes one fact table

Step 2 — Declare the Grain

- The **grain** is the most atomic level of data that will be stored in the fact table
- This is the most critical decision — everything else follows from it
- Example: "One row per individual sales transaction" (not per day, not per customer)
- More atomic grain = more flexibility for analysis but more storage

Step 3 — Identify the Dimensions

- Ask: "For each fact row, what descriptive context is needed?"
- Common dimensions: Time, Product, Customer, Location, Store, Employee
- Each dimension becomes a separate table with a primary key

Step 4 — Identify the Facts (Measures)

- Ask: "What is being measured at the grain level?"
- Facts should be numeric and additive where possible
- Examples: Sales amount, Quantity, Profit, Count of patients

4. Example — Sales Dimensional Model

Business process: Retail Sales **Grain:** One row per individual item sold in a single transaction

Fact Table — Sales_Fact:

Column	Type
time_key (FK)	Integer
product_key (FK)	Integer
customer_key (FK)	Integer
store_key (FK)	Integer
quantity_sold	Measure
sales_amount	Measure
discount_amount	Measure

Dimension Tables: Time_Dim, Product_Dim, Customer_Dim, Store_Dim

5. Dimensional Modelling vs ER Modelling

Feature	ER Modelling	Dimensional Modelling
Purpose	Transactional (OLTP) databases	Analytical (OLAP) data warehouses
Structure	Highly normalized (3NF)	Denormalized (star shape)
Number of joins	Many (complex)	Few (star: one join per dimension)
Query performance	Optimized for writes/updates	Optimized for reads/analytics
User-friendliness	Complex — hard for business users	Simple — business users can understand
Redundancy	Minimized	Allowed (for performance)
Designed for	Clerks, IT professionals	Analysts, managers, executives
Example schema	Normalized ER diagram	Star schema

6. Why ER Models Fail for Data Warehouses

- End users cannot understand complex normalized ER diagrams
- Many data warehouse projects failed historically due to complex ER-based designs
- Too many joins required = slow query performance
- Browsing and ad-hoc analysis become difficult
- Not optimized for aggregation and summarization

Q7: Design a Star Schema for a Healthcare Provider's Data Warehouse. The fact table should contain patient count. Dimensions include patients, doctors, diagnoses, procedures, and time. Justify the type of measure for patient count.

Answer:

1. Business Context

- A healthcare provider wants to analyze patient data across multiple dimensions
- Goal: Track patient counts by doctor, diagnosis, procedure, and time period
- This enables analysis like: "How many patients were treated by Dr. Sharma for diabetes in Q3 2024?"

2. Dimensional Modelling Steps Applied

Step 1 — Business Process: Hospital patient visits / admissions

Step 2 — Grain: One row per patient visit (one patient, one doctor, one diagnosis, one procedure, on one date)

Step 3 — Dimensions: Patient, Doctor, Diagnosis, Procedure, Time

Step 4 — Fact (Measure): patient_count (count of patient visits)

3. Star Schema Design

FACT TABLE — Patient_Visits_Fact

Column Name	Data Type	Description
visit_id (PK)	Integer	Surrogate key for the visit
patient_key (FK)	Integer	Foreign key → Patient_Dim
doctor_key (FK)	Integer	Foreign key → Doctor_Dim
diagnosis_key (FK)	Integer	Foreign key → Diagnosis_Dim
procedure_key (FK)	Integer	Foreign key → Procedure_Dim
time_key (FK)	Integer	Foreign key → Time_Dim
patient_count	Integer	Number of patients (measure)
visit_duration_hours	Decimal	Duration of visit (optional measure)

DIMENSION TABLE 1 — Patient_Dim

Column	Description
patient_key (PK)	Surrogate key
patient_id	Natural key from source system
patient_name	Full name
age	Age of patient
gender	M / F / Other
blood_group	A, B, AB, O
city	City of residence
insurance_type	Government / Private / None

DIMENSION TABLE 2 — Doctor_Dim

Column	Description
doctor_key (PK)	Surrogate key
doctor_id	Natural key
doctor_name	Full name
specialization	Cardiology, Neurology, etc.
department	Department name
qualification	MBBS, MD, etc.
hospital_name	Hospital where doctor works

DIMENSION TABLE 3 — Diagnosis_Dim

Column	Description
diagnosis_key (PK)	Surrogate key
diagnosis_code	ICD-10 code (e.g., E11 for Type 2 Diabetes)
diagnosis_name	Diabetes, Hypertension, etc.
diagnosis_category	Chronic / Acute / Infectious
body_system	Cardiovascular, Neurological, etc.

DIMENSION TABLE 4 — Procedure_Dim

Column	Description
procedure_key (PK)	Surrogate key
procedure_code	Procedure code
procedure_name	Blood test, MRI, Surgery, etc.
procedure_type	Diagnostic / Therapeutic / Preventive
cost_category	Low / Medium / High

DIMENSION TABLE 5 — Time_Dim

Column	Description
time_key (PK)	Surrogate key
full_date	14-03-2026
day	14
month	3
quarter	Q1
year	2026
day_of_week	Saturday
is_weekend	Yes / No
is_holiday	Yes / No

4. Justification of Measure Type for Patient Count

patient_count = count() = DISTRIBUTIVE MEASURE

Definition of Distributive Measure: A measure is distributive if it can be computed on partitions of data and the partial results can be combined to get the final result — without needing to go back to the original raw data.

Why patient_count is Distributive:

- count() satisfies this condition perfectly
- Example: If we count patients in January = 500, February = 600, March = 700
- The total for Q1 = 500 + 600 + 700 = 1800 ← this is correct without re-scanning raw data
- We can partition by hospital, sum partial counts, and get the total hospital count
- count() computed at any level can be rolled up to a higher level by simple addition

Formula check:

```
count(D) = count(D1) + count(D2) + ... + count(Dn)
where D is partitioned into D1, D2, ..., Dn
```

This holds true for count() → it is distributive.

Contrast with other measure types:

Measure Type	Example	Why patient_count is NOT these
Algebraic	avg() = sum()/count()	avg() needs both sum and count — two sub-aggregates. count() alone is simpler.
Holistic	median(), mode()	These cannot be computed from partial results — you need ALL data. count() does not have this limitation.

Conclusion: patient_count uses the count() function which is a distributive measure. It can be efficiently pre-computed at all levels of the data cube and stored in cuboids, making it the most computationally efficient type of measure.

Q8: Explain the Role of Metadata in a Data Warehouse. What are the different types of metadata?

Answer:

1. What is Metadata?

- Metadata literally means **"data about data"**
- In a data warehouse context, it describes and contextualizes the data stored in the warehouse
- Metadata is considered the **backbone of effective data warehouse management**
- Without metadata, users and systems cannot understand what data is stored, where it came from, or how it was transformed

2. Why Metadata is Important

- Helps users **understand** what data is available and what it means
- Enables the ETL process to know how to map source data to warehouse tables
- Supports **data governance** — enforcing business rules and ensuring data quality
- Allows **data lineage tracking** — knowing where each piece of data came from
- Improves **data accessibility** — users can search for and find the data they need
- Required for **query optimization** — the query engine uses metadata to choose the best cuboid or index

3. Types of Metadata

Type 1 — Technical / Structural Metadata

- Describes the physical structure of the data warehouse
- Stored in the **metadata repository** (also called the data dictionary or data catalog)
- Contents:
  - Schema definitions (tables, columns, data types, constraints)
  - View definitions
  - Dimension hierarchies
  - Derived data definitions
  - Data mart locations and contents
  - Indexes and partitions

Type 2 — Operational Metadata

- Describes how the data warehouse operates and is managed
- Contents:
  - **Data lineage:** History of how data was migrated and what transformations were applied
  - **Currency of data:** Whether a record is active, archived, or purged
  - **Monitoring information:** Usage statistics, error reports, audit trails, load logs

    - ETL job schedules and execution history
    - Data refresh timestamps

Type 3 — Business Metadata

- Describes data in business terms that non-technical users can understand
- Contents:
    - Business terms and definitions (data glossary)
    - Ownership of data (who is responsible for each dataset)
    - Data quality rules and thresholds
    - Charging policies (in some organizations, data access is charged internally)
    - KPI definitions and business rules

Type 4 — Process / ETL Metadata

- Describes the algorithms and mappings used during ETL
- Contents:
    - Mapping from source system fields to warehouse fields
    - Transformation rules applied
    - Algorithms used for aggregation and summarization

4. The Metadata Repository

- A dedicated store within the data warehouse architecture that holds all metadata
- Acts as a central catalog for the entire warehouse
- Accessed by: ETL tools, query engines, BI tools, data administrators, and end users
- Modern implementations: Data Catalog tools (AWS Glue Data Catalog, Apache Atlas, Alation)

5. Metadata and the Data Catalog

- A **data catalog** combines metadata with search tools
- Analysts can search for datasets by keyword, owner, or business term
- Helps reduce time spent looking for the right data
- Improves self-service analytics

6. Summary Table

Type	What it describes	Example
Technical/Structural	Physical schema	Column names, data types
Operational	How warehouse is managed	Load timestamps, error logs
Business	Business meaning of data	Definition of "active customer"
Process/ETL	How data was transformed	Mapping from source field to warehouse field

---

Q9: Explain the Architectural Components of a Data Warehouse in detail.

Answer:

1. Overview

A data warehouse has several distinct architectural components that work together to move data from source systems to end users. Understanding each component is essential.

2. Component 1 — Data Sources (Operational Layer)

- The **bottom-most layer** containing all systems that generate raw data
- Types of data sources:
  - **Internal sources:** OLTP databases (sales, HR, finance, inventory), ERP systems, CRM systems
  - **External sources:** Market data feeds, social media data, government databases, third-party data providers
  - **Legacy systems:** Old mainframe systems, flat files
- Data from these sources is heterogeneous — different formats, systems, and conventions

3. Component 2 — Staging Area

- A **temporary storage area** between the data sources and the warehouse
- Raw data is first extracted from sources and landed in the staging area
- Data is then cleaned, transformed, and validated here before being loaded
- The staging area is **not available to end users** — it is purely for internal ETL processing
- Protects the warehouse from incomplete or dirty data
- Usually cleared after each successful ETL run

4. Component 3 — ETL Engine (Back-End Tools)

- The engine that performs **Extract, Transform, Load** operations

- Sub-components:
  - **Data Extraction tools:** Pull data from heterogeneous sources
  - **Data Cleaning tools:** Detect and fix errors, handle missing values
  - **Data Transformation tools:** Convert formats, apply business rules, generate surrogate keys
  - **Data Loading tools:** Bulk-load data into warehouse tables efficiently
  - **Refresh tools:** Propagate updates from sources to the warehouse (incremental loads)

5. Component 4 — Data Warehouse Storage (Central Repository)

- The **core of the architecture** — where integrated, cleaned, historical data is permanently stored
- Organized using **dimensional models** (star schema, snowflake schema)
- Contains fact tables and dimension tables
- Optimized for **read-heavy, complex analytical queries**
- May be implemented on-premises (traditional) or in the cloud (AWS Redshift, Google BigQuery, Snowflake)

6. Component 5 — Data Marts

- **Smaller, subject-specific subsets** of the main warehouse
- Created for specific departments or business areas
- Examples: Sales data mart, Marketing data mart, Finance data mart, HR data mart
- Two types:
  - **Dependent data mart:** Fed directly from the enterprise data warehouse (preferred — more consistent)
  - **Independent data mart:** Built directly from source data (may cause inconsistencies)
- Faster query response for department-specific queries

7. Component 6 — OLAP Server

- Sits between the warehouse storage and the front-end tools
- Processes analytical queries efficiently
- Types: ROLAP (Relational), MOLAP (Multidimensional), HOLAP (Hybrid)
- Provides operations: Roll-up, Drill-down, Slice, Dice, Pivot

8. Component 7 — Metadata Repository

- Stores all metadata about the warehouse

- Acts as the data dictionary — describes schemas, transformations, lineage, business definitions
- Used by ETL tools, query engines, and end users
- Critical for governance and understanding

9. Component 8 — Front-End / User Access Tools

- The **top-most layer** that end users interact with
- Types:
  - **Query and Reporting tools:** SQL-based queries, standard reports (Tableau, Power BI)
  - **OLAP tools:** Multidimensional analysis, drill-down exploration
  - **Data Mining tools:** Pattern discovery, predictive analytics (Python, R, Weka)
  - **Dashboards:** Executive-level summary views with KPIs

10. Three-Tier Architecture Summary

Tier	Components	Function
Bottom (Data Sources)	Operational DBs, External Data	Generate and supply raw data
Middle (Data Storage)	Staging Area, ETL Engine, Warehouse, Data Marts, OLAP Server	Store, clean, integrate, serve
Top (Front-End)	BI Tools, Query Tools, Data Mining Tools, Dashboards	User access and analysis

---

Q10: Explain the Measures in a Data Cube — Distributive, Algebraic, and Holistic. How does categorization help in efficient computation?

Answer:

1. What are Measures?

- Measures are the **numeric values** stored in the fact table of a data warehouse
- They represent what is being tracked or analyzed in the business process
- Examples: total_sales, count_of_orders, average_salary, maximum_temperature
- Measures are categorized based on how they behave when aggregated across multiple levels of a data cube

2. Category 1 — Distributive Measures

Definition: A function F is distributive if it can be computed by:

1. Partitioning the data into n subsets
2. Computing F on each subset (partial results)
3. Combining the partial results using the same function F

The final result equals what you would get by computing F on the entire data without partitioning.

Mathematical condition:

```
F(D) = F(F(D1), F(D2), ..., F(Dn))
where D is partitioned into D1, D2, ..., Dn
```

Examples:

Function	Why Distributive
count()	count(all) = count(part1) + count(part2) + ...
sum()	sum(all) = sum(part1) + sum(part2) + ...
min()	min(all) = min(min(part1), min(part2), ...)
max()	max(all) = max(max(part1), max(part2), ...)

Computation efficiency: HIGHEST — can be pre-computed for all cuboids in the data cube. Partial results are stored and combined when needed.

3. Category 2 — Algebraic Measures

Definition: A function is algebraic if it can be computed by an algebraic function that takes **M arguments** (M is a bounded integer), where each argument is itself obtained by applying a distributive aggregate function.

In simple terms: Algebraic measures are computed from a fixed number of distributive sub-aggregates.

Examples:

Function	Why Algebraic	Sub-aggregates needed
avg()	avg = sum() / count()	2 sub-aggregates: sum and count
standard_deviation()	Computed from sum and sum of squares	2 sub-aggregates
min_N() (top N minimum values)	Needs partial sorted lists	Bounded extra storage

Computation efficiency: MEDIUM — cannot be computed as simply as distributive, but the number of extra values that must be stored per partition is bounded (e.g., for avg(), store both sum and count).

4. Category 3 — Holistic Measures

Definition: A function is holistic if there is **no constant bound** on the storage needed to describe a sub-aggregate. Partial results cannot be meaningfully combined to give the final result.

In simple terms: You MUST have all the data together to compute these — you cannot split and merge.

Examples:

Function	Why Holistic
median()	To find the true median, you must sort ALL values — partial medians cannot be combined
mode()	The mode of a partition is not necessarily the mode of the whole dataset
rank()	Ranking requires knowing all values simultaneously

Example to illustrate why median is holistic:

- Partition 1: {1, 3, 5} → median = 3
- Partition 2: {2, 4, 6} → median = 4
- Combined: {1, 2, 3, 4, 5, 6} → median = (3+4)/2 = 3.5
- 3.5 ≠ any combination of 3 and 4 using a simple formula → holistic!

Computation efficiency: LOWEST — cannot be efficiently pre-computed across cuboids. Computed on-demand from raw data.

5. How Categorization Helps in Efficient Cube Computation

Category	Pre-computable?	Storage needed	Query speed
Distributive	Yes — all cuboids	Minimal (just the result)	Fastest
Algebraic	Partially — store sub-aggregates	Small and bounded	Medium
Holistic	No — compute on-demand	Unbounded	Slowest

Practical impact:

- When designing a data cube, prioritize **distributive measures** — they can be materialized at every cuboid level for instant query response
- **Algebraic measures** can be partially pre-computed — store the sub-aggregates (e.g., sum + count for avg) at each cuboid level and compute the final value at query time
- **Holistic measures** should be used sparingly — only compute them when specifically requested, never pre-materialize them

---

Q11: Explain the concept of Materialized Views and Efficient Data Cube Computation.

Answer:

1. What is a Materialized View?

- A **materialized view** is a pre-computed query result that is stored physically in the database
- Unlike a regular (virtual) view, which is computed every time it is queried, a materialized view stores the result once and reuses it
- When a query arrives, the system checks if a materialized view already contains the answer
- If yes → return the pre-stored result instantly (very fast)
- If no → compute the result from scratch

2. Materialized Views in the Context of Data Cubes

A data cube for n dimensions generates a **lattice of cuboids:**

- Base cuboid (most detailed — all dimensions)
- Apex cuboid (least detailed — no dimensions, just one total value)
- All possible combinations of dimensions in between

Formula for number of cuboids:

```
Total cuboids = Π(Li + 1)  for i = 1 to n
```

Where n = number of dimensions, Li = number of levels in dimension i (including "all")

Example:

- 4 dimensions, each with 4 levels (+ "all" = 5 levels)
- Total cuboids = 5 × 5 × 5 × 5 = 625 cuboids

3. Three Approaches to Materialization

Approach 1 — Full Materialization

- Pre-compute and store **every possible cuboid** in the lattice
- Advantage: Fastest query response — any query can be answered from a pre-stored cuboid
- Disadvantage: Requires enormous storage (625+ cuboids can be very large)

Approach 2 — No Materialization

- Store only the raw data (base cuboid)
- Compute every query from scratch at query time
- Advantage: Minimum storage
- Disadvantage: Slowest query response — every query requires full computation

Approach 3 — Partial Materialization (Recommended)

- Select and pre-compute only the **most frequently accessed or most useful cuboids**
- Balances storage cost vs query performance
- This is the standard approach in practice

4. Which Cuboids to Materialize? (Selection Criteria)

Criterion	Explanation
Access frequency	Cuboids queried most often should be materialized
Size of cuboid	Smaller cuboids cost less to store — prefer smaller ones when access is similar
Sharing potential	A cuboid that can serve multiple different queries is more valuable to materialize
Query response time requirement	If a query must respond in under 1 second, its cuboid must be materialized

5. Efficient OLAP Query Processing Using Cuboids

When an OLAP query arrives:

1. Determine which OLAP operations are needed (roll-up, drill-down, slice, dice)
2. Find the **most specific materialized cuboid** that can answer the query
3. Apply any remaining operations (e.g., roll-up) on top of that cuboid
4. Always use the smallest materialized cuboid that satisfies the query to minimize computation

Example:

- Query: "Total sales by product and year"
- Available cuboids: {product, year, city} and {product, year}
- Best choice: Use {product, year} cuboid directly — no further aggregation needed

6. Refreshing Materialized Views

- When source data changes (after an ETL load), materialized views must be updated
- Two strategies:
  - **Full refresh:** Recompute the entire view from scratch — simple but expensive
  - **Incremental refresh:** Update only the portions affected by new data — efficient but complex

---

Q12: Compare Data Lake, Data Warehouse, and Data Lakehouse. What problems does a Data Lakehouse solve?

Answer:

1. Data Lake — Overview

Definition: A centralized repository that stores raw data in its **original/native format** at any scale.

Key characteristics:

- Stores ALL types of data: structured, semi-structured, and unstructured
- Schema is defined **on-read** (not at the time of storage — also called schema-on-read)
- Very low storage cost (uses cloud object storage like AWS S3, Azure ADLS, GCS)
- Best suited for: Big data analytics, machine learning, data science

Problems with Data Lakes:

Problem	Description
No ACID transactions	Cannot guarantee data consistency — concurrent writes may corrupt data
Data quality issues	Raw, unprocessed data may be unreliable or incorrect
"Data Swamp" risk	Too much unorganized data with no governance → becomes unusable
Poor BI performance	Not optimized for fast SQL queries used by business analysts
No schema enforcement	Bad data can be written in any format, breaking downstream pipelines

2. Data Warehouse — Overview

Definition: A subject-oriented, integrated, time-variant, nonvolatile collection of structured data for analytical decision-making.

Key characteristics:

- Stores only **structured, processed, clean** data
- Schema is defined **on-write** (data must conform to the schema before loading)
- High performance for SQL queries and BI tools
- Best suited for: Business intelligence, standard reporting, executive dashboards

Problems with Data Warehouses:

Problem	Description
Cannot store unstructured data	Images, videos, logs, PDFs cannot be stored
Expensive	High licensing costs for commercial warehouses
Rigid schema	Changing the schema is difficult and time-consuming
Limited ML support	Not designed for machine learning workloads
Long ETL cycles	Data takes time to go through the ETL pipeline before being available

3. Data Lakehouse — Overview

Definition: A hybrid architecture that combines the **flexibility and low cost of a Data Lake** with the **performance, reliability, and structure of a Data Warehouse**.

Formula: Data Lake + Data Warehouse = Data Lakehouse

Key features:

Feature	Description
ACID Transactions	Guarantees data consistency and reliability (via Delta Lake, Apache Iceberg, Apache Hudi)
Schema enforcement	Validates data at write time, but can also evolve flexibly
Schema evolution	Can change schema without breaking existing queries
Time travel	Query data as it existed at a past point in time (data versioning)
High performance	Optimized query engines (Spark, Trino, Presto) run SQL fast on lake data
Single source of truth	Both BI and ML teams use the same data — no duplication
Open formats	Uses open file formats like Parquet and ORC — not vendor-locked

4. What Problems Does the Data Lakehouse Solve?

Problem 1 — The Two-Copy Problem:

- Before Lakehouses, companies had to maintain both a Data Lake (for ML) and a Data Warehouse (for BI)
- This meant two copies of the same data → expensive, inconsistent, hard to maintain
- Lakehouse gives a single unified store for both use cases

Problem 2 — Data Lake Quality Issues:

- Lakehouses add ACID transactions and schema enforcement on top of the lake
- Prevents the "data swamp" problem
- Data quality is enforced at write time

Problem 3 — Data Warehouse Rigidity:

- Lakehouses use flexible, open formats (Parquet, ORC) with optional schema enforcement
- Schema can evolve without rebuilding the entire warehouse

5. Full Comparison Table

Feature	Data Lake	Data Warehouse	Data Lakehouse
Data Types Supported	All (structured, semi, unstructured)	Structured only	All
Schema	On-read (flexible)	On-write (rigid)	Both (enforced + flexible)
ACID Transactions	No	Yes	Yes
Storage Cost	Low	High	Low-Medium
Query Performance	Low (for SQL)	High	High
BI Support	Limited	Excellent	Excellent
ML/AI Support	Excellent	Limited	Excellent
Data Quality	Low (raw data)	High (processed)	High (enforced)
Time Travel	No	No	Yes
Example Technologies	AWS S3, Azure ADLS	Redshift, BigQuery, Snowflake (traditional)	Delta Lake, Apache Iceberg, Apache Hudi

6. Lakehouse Architecture Layers

Layer	Technology Examples
Storage Layer	AWS S3, Azure ADLS, Google GCS (open formats: Parquet, ORC)
Table Format Layer	Delta Lake, Apache Iceberg, Apache Hudi
Processing Layer	Apache Spark, Flink, Trino, Presto
Analytics & BI	Power BI, Tableau, Looker
ML & AI	MLflow, TensorFlow, PyTorch
Governance & Security	Access control, versioning, auditing

Q13: Explain the Three Kinds of Data Warehouse Applications with examples.

Answer:

1. Overview

Data warehouse applications are typically classified into three categories based on the type of analysis they perform. Together, they form a complete Business Intelligence (BI) ecosystem.

2. Application 1 — Information Processing

What it does:

- Supports **basic querying, statistical analysis, and reporting**
- Answers the question: **"What happened?"**
- Uses standard reporting formats: tables, crosstabs, charts, graphs

Tools and techniques:

- SQL queries on the warehouse
- Standard reports (monthly, quarterly, annual)
- Pivot tables and cross-tabulations
- Basic statistical analysis (averages, totals, percentages)

Examples:

- Monthly sales report by region
- Product inventory summary for the current quarter
- Number of customer complaints received this week
- Annual revenue by product category

Characteristics:

- Relatively simple analysis
- Pre-defined report formats
- Results are presented as tables or charts
- Primarily used by operational managers and analysts

3. Application 2 — Analytical Processing (OLAP)

What it does:

- Supports **multidimensional analysis** of warehouse data
- Answers the question: **"Why did it happen?"** and **"What if?"**
- Uses OLAP operations: slice-dice, roll-up, drill-down, pivot

Tools and techniques:

- OLAP servers (ROLAP, MOLAP, HOLAP)
- Multidimensional data cube exploration
- What-if analysis and scenario modeling
- Comparative analysis across dimensions

Examples:

- Analyzing sales by product, region, and time simultaneously
- Rolling up from weekly to quarterly to annual figures to identify trends
- Drilling down from national sales to state to city to store level
- Comparing sales performance across different product categories in Q3 vs Q4
- Slicing the sales cube to view only "Electronics" products

Characteristics:

- More complex than information processing
- Ad-hoc and exploratory analysis
- Interactive — users navigate the cube in real time
- Used by business analysts and managers

4. Application 3 — Data Mining

What it does:

- Supports **knowledge discovery from hidden patterns** in data
- Answers the question: **"What will happen?"** and **"What patterns are hidden in the data?"**
- Automated discovery of previously unknown patterns and relationships

Techniques used:

Technique	What it finds	Example
Association Rules	Items that frequently occur together	"80% of customers who buy diapers also buy baby formula"
Classification	Assign new records to predefined categories	Classify loan applicants as high-risk or low-risk
Clustering	Group similar records together	Segment customers into behavioral groups
Regression	Predict a numeric value	Predict next quarter's revenue
Anomaly Detection	Find unusual records	Detect fraudulent credit card transactions
Sequential Patterns	Find patterns over time	Identify buying behavior sequences

Examples:

- Predicting which customers are likely to leave (churn prediction)
- Discovering that customers who buy Product A also tend to buy Product B (market basket analysis)
- Identifying patient groups with similar treatment outcomes in a hospital
- Finding that sales of umbrellas spike 3 days before monsoon arrives

Characteristics:

- Most advanced and complex application
- Results are presented using visualization tools
- Requires specialized algorithms and statistical knowledge
- Used by data scientists and advanced analysts

5. Why All Three Are Important Together

Application	Question Answered	User	Output
Information Processing	What happened?	Operational manager	Reports, tables
Analytical Processing	Why? What if?	Business analyst	OLAP cube analysis
Data Mining	What will happen? What patterns?	Data scientist	Models, patterns, predictions

Together, these three layers form a complete decision support system:

- Information Processing gives the baseline facts
- Analytical Processing gives understanding and context
- Data Mining gives predictive power and hidden insights

Summary — Topics Added in This Supplementary Document

#	Topic	Unit	Why Added
Q1	Semi-structured data characteristics in depth	Unit I	Exam Q1A — needed more detail than existing notes
Q2	Nominal vs Ordinal with full comparison	Unit I	Exam Q1B — needed side-by-side comparison table
Q3	Attribute subset selection (all 4 methods)	Unit I	Exam Q2A — needed all methods with clear table
Q4	Data discretization with real-world examples	Unit I	Exam Q2B — needed applied examples (age, salary, temperature)
Q5	ETL Process in detail	Unit II	Syllabus topic — missing from existing notes
Q6	Dimensional Modelling vs ER Modelling	Unit II	Exam Q3A context — missing from existing notes
Q7	Star schema design (Healthcare) + measure justification	Unit II	Exam Q3A — measure type justification was missing
Q8	Role of Metadata — all types in detail	Unit II	Syllabus topic — existing notes had partial coverage
Q9	Architectural components of a data warehouse	Unit II	Syllabus topic — needed dedicated treatment
Q10	Measures: computation efficiency and categorization	Unit II	Syllabus topic "Computation" — missing from existing notes
Q11	Materialized Views and efficient cube computation	Unit II	Syllabus topic — existing notes were brief

#	Topic	Unit	Why Added
Q12	Data Lake vs Warehouse vs Lakehouse (full comparison)	Unit II	Syllabus topic — existing notes needed more depth
Q13	Three kinds of data warehouse applications	Unit II	Syllabus topic — complete treatment with examples