UNIT 1 — Detailed but Very Simple Explanations

1.1 Introduction
A DBMS is software used to store and manage data safely.
Instead of keeping many separate files, all data is kept in one organized place.
It helps us add, delete, update, and search data easily.
It protects data from loss and misuse.
The main goal is to store information properly and get it quickly whenever needed.

1.2 Advantages of DBMS
A DBMS avoids repeating the same data again and again.
It gives correct and consistent data because everything is stored in one place.
Information can be searched fast using queries.
Different users can work at the same time without disturbing each other.
It also protects data using permissions and rules.

1.3 Disadvantages of File System
Old file systems stored data in separate files, leading to confusion.
The same data was stored in many files, wasting storage.
Updating data everywhere was difficult and caused mismatches.
Searching across multiple files was slow and complicated.
Security was poor because files could not be protected easily.

1.4 Advantages of DBMS
DBMS keeps everything organized in tables.
We can run simple SQL queries to find information quickly.
Data remains correct with the help of rules like constraints.
Multiple people can work at the same time safely.
It ensures complete safety through authentication and access control.

1.5 DBMS vs File System
DBMS reduces repeated data; file system does not.
DBMS gives strong security; file system is weak.
DBMS stores data in a structured way; file system stores in scattered files.
DBMS supports fast and flexible retrieval; file system does not.
DBMS supports multiple users, while file system cannot handle concurrent access well.

1.6 Database System Applications
Databases are used in banks to store account data.
Airlines and railways use them for reservation systems.
Universities use them for student records.
Telecom companies use them for billing and customer details.
Almost every modern system with large data uses DBMS.

1.7 Data Abstraction
Data abstraction means showing only the needed information to users.
It hides complex storage details in the background.
There are 3 levels:
– Physical level: how data is stored.
– Logical level: what data is stored.
– View level: what users see.
This makes DBMS simple to use.

1.8 DBMS Users
DBA controls the entire database system.
Programmers write applications that access the database.
End users use the programs to get work done.

Specialized users work with scientific or special databases.
Each user has a different role and permission.

1.9 Data Models
A data model describes how data is arranged.
It helps in designing the database structure.
Examples: relational model (tables), ER model (diagrams), object model (objects).
Each model shows data, relationships, and rules.
It makes database design easier and more understandable.

1.10 Components of DBMS Architecture
The query processor understands SQL commands.
The storage manager handles how data is stored in memory and disk.
The system uses data dictionary to store metadata (information about structure).
It also uses transaction manager to maintain correctness.
These parts work together to run queries fast and safely.

1.11 Relational Model
Data is stored in tables with rows and columns.
Each row is a record; each column is a property.
Primary keys identify unique records.
Tables can be linked using keys.
It is the most simple and powerful way to store data.

1.12 Data Independence
Data independence means changing one level does not affect another.
You can change storage details without changing table structure.
You can change table structure without changing user views.
It saves a lot of time and cost.
It also makes database upgrades easier.

1.13 Schema and Instance
Schema is the framework or structure of the database.
It describes tables, columns, and relationships.
Instance is the actual data present in tables at a particular moment.
Schema is fixed, but instance keeps changing.
Together they define the full state of the database.

1.14 ER Model
ER model shows real-world data using diagrams.
Entities represent objects like Student or Employee.
Attributes describe properties like name or age.
Relationships show how entities are connected.
It makes database planning easy and clear.

1.15 Responsibilities of DBA
DBA creates and maintains the database structure.
They control user access and permissions.
They take backups and restore data if needed.
They monitor performance and fix issues.
They ensure database security and privacy.

1.16 Entity, Entity Set, Attributes
Entity is any real-world object stored in the database.
Entity set is a group of similar entities.
Attributes describe properties like name, age, or salary.

Some attributes are simple; some are composite.
Primary key uniquely identifies each entity.

1.17 Structural Constraints (Cardinality)
Defines how many entities can be related.
Types include 1:1, 1:N, N:1, and M:N.
Example: One teacher teaches many students (1:N).
Used to design correct relationships.
Makes sure the model follows real-life rules.

UNIT 2 — DETAILED BUT VERY SIMPLE EXPLANATIONS

✅ 2.1 SQL Introduction
SQL is a language used to talk to the database.
It helps us create tables, store data, update data, and delete data.
It is very simple and uses English-like commands.
All modern databases like MySQL, Oracle, SQL Server use SQL.
SQL is the standard language to manage structured data.

✅ 2.2 SQL Data Types
Data types define what kind of values can be stored in a column.
Common types include NUMBER, VARCHAR, DATE, FLOAT, CHAR.
Choosing correct data type prevents mistakes in data.
It ensures data remains consistent and valid.
Every column in a table must have a data type.

✅ 2.3 DDL (Data Definition Language)
DDL is used to create and modify table structure.
Commands include CREATE, ALTER, DROP, TRUNCATE.
CREATE makes new tables, ALTER changes them, DROP deletes them.
DDL changes the structure, not the actual data.
These commands affect the whole table at once.

✅ 2.4 DML (Data Manipulation Language)
DML is used to change data inside tables.
Commands: INSERT, UPDATE, DELETE.
INSERT adds new rows, UPDATE modifies existing rows, DELETE removes rows.
DML updates content but does not change table structure.
It is used in daily operations of applications.

✅ 2.5 DCL (Data Control Language)
DCL controls who can access the data.
Commands: GRANT and REVOKE.
GRANT gives permission to a user; REVOKE removes it.
Used by DBA to maintain security.
Prevents unauthorized access to sensitive data.

✅ 2.6 TCL (Transaction Control Language)
TCL manages transactions in the database.
Commands: COMMIT, ROLLBACK, SAVEPOINT.
COMMIT saves all changes permanently.
ROLLBACK cancels changes if any error happens.
Helps maintain accuracy and consistency in multi-step updates.

✅ 2.7 Constraints

Constraints are rules applied on columns to protect data.
Examples: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK.
They prevent invalid or incomplete data.
They maintain accuracy and consistency of tables.
Constraints act like “guards” for data quality.

✅ 2.8 Primary Key
Primary key uniquely identifies every row in a table.
It cannot contain duplicate or NULL values.
A table can have only one primary key.
Examples: RollNo, EmployeeID, AccountNo.
It helps in searching and linking tables.

✅ 2.9 Foreign Key
Foreign key links one table to another.
It ensures relationship between tables remains correct.
It must match a value in the primary key of another table.
Prevents deletion of important linked data.
Maintains referential integrity.

✅ 2.10 Unique Constraint
Unique ensures that no two rows have the same value in a column.
Unlike primary key, UNIQUE can allow a NULL value.
Used for columns like email, mobile number.
Helps maintain uniqueness in special fields.
Prevents duplication of sensitive data.

✅ 2.11 NOT NULL Constraint
NOT NULL means the column must have a value.
It cannot be left empty.
Used when data is mandatory like name, age, roll number.
Prevents missing or incomplete information.
Improves data reliability.

✅ 2.12 CHECK Constraint
CHECK ensures that values follow a certain rule.
Example: age > 18, salary > 0.
It prevents incorrect values from entering the table.
Acts like a filter during data entry.
Helps maintain legal or logical correctness.

✅ 2.13 Default Constraint
DEFAULT gives a value automatically when the user does not enter one.
Used for fields like country = ‘India’, status = ‘Active’.
Helps reduce input effort.
Prevents NULL values where not required.
Improves consistency.

✅ 2.14 SELECT Query
SELECT is used to fetch data from tables.
We can select specific columns or entire table.
Conditions can be added using WHERE.
Sorting done using ORDER BY.
It is the most commonly used SQL command.

✅ 2.15 WHERE Clause

WHERE filters rows based on a condition.
Used to select only matching data.
Example: WHERE city = ‘Pune’.
Supports operators like <, >, =, LIKE, BETWEEN.
Makes searching fast and accurate.

✅ 2.16 ORDER BY Clause
ORDER BY sorts data in ascending or descending order.
ASC = smallest to biggest; DESC = biggest to smallest.
Example: ORDER BY salary DESC.
Helps analyze and present data neatly.
Often used in reports and lists.

✅ 2.17 Aggregate Functions
Aggregate functions calculate values from multiple rows.
Common functions: SUM, AVG, COUNT, MAX, MIN.
Used in total marks, average salary, number of employees.
They summarize large data.
Useful for reports and analytics.

✅ 2.18 GROUP BY
GROUP BY groups rows that have same values in a column.
Used with aggregate functions.
Example: total salary per department.
Helps create category-wise summaries.
Commonly used in financial and business reports.

✅ 2.19 HAVING Clause
HAVING filters grouped data.
It works only with GROUP BY.
Example: show departments with total salary > 50,000.
Used for grouped conditions.
WHERE cannot be used after grouping, but HAVING can.

✅ 2.20 Joins
Joins combine rows from two or more tables.
Types: INNER, LEFT, RIGHT, FULL.
INNER shows only matching rows.
LEFT shows all left rows, even if not matched.
Used to link related tables like student and marks.

✅ 2.21 Index
Index improves the speed of searches in a table.
Works like an index of a book for fast lookup.
Takes extra storage space.
Speeds up SELECT but slows INSERT/UPDATE.
Useful for large tables with many queries.

✅ 2.22 View
View is a virtual table created using SELECT query.
It does not store data physically.
Shows only limited or required data to users.
Improves security by hiding sensitive columns.
Easy to use and simplifies complex queries.

✅ 2.23 Updating / Dropping Views

Views based on one table can be updated.
Views with joins or aggregate functions cannot be updated.
DROP VIEW removes the view definition.
Changes in the base table reflect automatically in the view.

✅ 2.24 PL/SQL Basics
PL/SQL is Oracle's programming extension to SQL.
It supports variables, loops, and conditions.
Used to write powerful programs inside the database.
Helps automate tasks.
Improves performance through stored blocks.

✅ 2.25 Stored Procedure
Stored procedure is a saved program in the database.
It runs faster because it is precompiled.
Reduces repeat coding.
Provides security by restricting direct table access.
Called using EXEC procedureName.

✅ 2.26 Trigger
Trigger runs automatically when an insert/update/delete happens.
Used for auditing, validation, or automatic logs.
Types: BEFORE, AFTER, INSTEAD OF.
Cannot be called manually.
Ensures rules are always followed.

✅ 2.27 BEFORE UPDATE Trigger (Example Question)
Used to save old data before updating a table.
Old values are stored in a log table.
Helps track changes for security or auditing.
Useful in fee records, salary changes, attendance updates.

✅ 2.28 SQL Queries on Movie Database
Based on JOIN operations between tables.
Find movies by director name.
Sort movies by stars or title.
Update ratings for specific directors.
Practices real SQL query writing.

UNIT 3 — DETAILED BUT VERY SIMPLE EXPLANATIONS

✅ 3.1 Attribute and Domain
An attribute is a column in a table that stores one type of information (like Name, Age, RollNo).
A domain is the set of allowed values for an attribute (Age must be numbers, Name must be text).
Attributes describe the structure of a table.
Domains ensure that only correct values are stored.
Together, they help maintain clean and meaningful data.
Example: Age cannot be negative because the domain prevents it.

✅ 3.2 Codd's 12 (13) Rules
Codd created rules to define how a true relational DBMS should behave.
Rules include storing all data in tables, supporting SQL, handling NULLs properly, and ensuring data independence.
They also require integrity constraints, view updates, and distribution transparency.
These rules ensure the database remains consistent, reliable, and easy to use.

No system follows all rules fully, but they guide DBMS design.
They form the foundation of relational databases.

✅ 3.3 Keys in DBMS
Keys uniquely identify rows and maintain relationships.
Super key → Any set of attributes that uniquely identifies a row.
Candidate key → Minimal super key (smallest unique set).
Primary key → One chosen candidate key that uniquely identifies every row.
Alternate key → Other candidate keys not chosen as primary.
Foreign key → Refers to primary key of another table, linking data.
Keys keep data connected and prevent duplicates.

✅ 3.4 Super Key
A super key is any combination of attributes that uniquely identifies a row.
It may contain extra unnecessary attributes.
Example: (RollNo), (RollNo + Name), (RollNo + Phone) — all can be super keys.
Not all super keys are minimal.
Used to identify possible unique combinations in a table.

✅ 3.5 Candidate Key
A candidate key is a minimal super key.
It has no extra attributes — only the required ones.
A table can have many candidate keys.
Example: RegNo, and (RollNo + Phone).
One of these becomes the primary key.

✅ 3.6 Primary Key
Primary key uniquely identifies each row.
It cannot have duplicates or NULL values.
A table can have only one primary key.
Used for indexing and fast searching.
Ensures correctness and uniqueness of data.

✅ 3.7 Alternate Key
All candidate keys except the primary key are alternate keys.
They are still unique but not used as the main key.
Example: If RegNo is primary key, RollNo+Phone is alternate key.
Alternate keys help maintain additional uniqueness.
Used when another unique attribute is needed.

✅ 3.8 Foreign Key
Foreign key maintains relationships between tables.
It refers to the primary key of another table.
Ensures linked data remains consistent.
Prevents deletion of referenced data.
Used to connect tables like Student and Department.

✅ 3.9 Composite Key
A composite key is made using two or more attributes.
Used when a single field cannot uniquely identify rows.
Example: (StudentID + CourseID) in Enrollment table.
Ensures uniqueness in many-to-many relationships.
Common in link tables.

✅ 3.10 ER Model
ER model helps design the database structure using diagrams.
It has entities (objects), attributes (properties), and relationships (connections).

Entities shown as rectangles, attributes as ovals, relationships as diamonds.
Helps visualize how data is connected.
Used in planning before creating actual tables.

✅ 3.11 Entity Types
Two types: Strong and Weak.
Strong entity has its own primary key (e.g., Student).
Weak entity depends on another entity and cannot exist alone (e.g., OrderItem depends on Order).
Weak entity uses partial key + foreign key.
Used to show real-world dependency.

✅ 3.12 Relationship Types
Shows how entities are related.
Types: 1:1, 1:N, N:1, M:N.
Defines how many entities participate.
Example: One teacher teaches many students (1:N).
Helps design correct ER diagrams.

✅ 3.13 Mapping Cardinality
Specifies minimum and maximum relationships.
Shows how many entities in one set relate to the other.
Helps create correct relational structure.
Ensures database reflects real-world rules.

✅ 3.14 Participation Constraints
Defines whether participation is total or partial.
Total participation → entity must participate (e.g., every employee must belong to a department).
Partial → entity may or may not participate.
Used to design accurate ER diagrams.

✅ 3.15 Generalization
Combines multiple lower-level entities into a higher-level entity.
Example: Car, Bus → Vehicle.
Used to remove redundancy.
Shows common properties in one parent entity.

✅ 3.16 Specialization
Opposite of generalization.
Breaks a higher-level entity into smaller subtypes.
Example: Employee → Teacher, Clerk, Manager.
Used when subtypes have different attributes.
Improves clarity in design.

✅ 3.17 Aggregation
Represents a relationship as an entity itself.
Used when relationships need to participate in more relationships.
Example: Student teaches Course using Instructor — group as one object.
Helps in complex real-world modeling.

✅ 3.18 Relational Algebra
A mathematical language to query the database.
Basic operations: SELECT, PROJECT, JOIN, UNION, INTERSECT, MINUS.
Used to process and filter data.
Forms the basis of SQL operations.
Helps in theoretical understanding of DBMS.

✅ 3.19 SELECT Operation (σ)

Used to filter rows based on condition.
Example: σ(age > 20).
Works like SQL WHERE clause.
Gives only matching records.
Used for row selection.

✅ 3.20 PROJECT Operation (π)
Used to select specific columns.
Example: π(Name, Salary)(Employee).
Removes duplicate values.
Used for column selection.
Similar to SELECT column names in SQL.

✅ 3.21 JOIN Operation
Combines two tables based on a condition.
Shows related data together.
Used in almost all real queries.
Example: Employee ⋈ Department.
Provides complete information.

✅ 3.22 UNION Operation
Combines rows of two tables.
Both tables must have same columns.
Removes duplicate rows.
Used to merge similar datasets.
Example: Data from two branches.

✅ 3.23 SET DIFFERENCE (MINUS)
Gives rows present in one table but not in another.
Used for comparing datasets.
Example: students who paid fees – students who submitted form.
Useful in reporting and analysis.

✅ 3.24 INTERSECTION
Gives rows common to both tables.
Used to find matching records.
Example: students who paid fees AND submitted form.
Helps identify overlap.

✅ 3.25 CARTESIAN PRODUCT
Combines every row of one table with every row of another.
Produces very large output.
Basis of JOIN operations.
Rarely used directly.

✅ 3.26 Tuple Relational Calculus
Non-procedural query language.
You specify what you want, not how to get it.
Uses simple logical expressions.
Basis for SQL.
Easier to express complex queries.

✅ 3.27 Domain Relational Calculus
Similar to tuple calculus but uses domain variables.
Each variable refers to a column value.
More flexible for column-level conditions.

Forms theoretical foundation of SQL.
Used in academic understanding.

UNIT 4 — DETAILED BUT VERY SIMPLE EXPLANATIONS

✅ 4.1 Transaction — Definition & Example
A transaction is a small unit of work in a database, such as inserting or updating data.
It must finish completely or not run at all.
Example: Transferring money from one bank account to another.
If any step fails, the entire operation must go back to the original state.
Transactions ensure the database remains correct even during errors.

✅ 4.2 ACID Properties
ACID ensures every transaction behaves safely and correctly.
A — Atomicity: All steps must complete, or none will.
C — Consistency: Database must stay valid before and after transaction.
I — Isolation: Transactions don’t disturb each other even if run at the same time.
D — Durability: Once committed, changes are permanent even after power failure.
These properties keep data reliable.

✅ 4.3 States of a Transaction
A transaction goes through multiple states.
Active: When steps are running.
Partially committed: After last statement runs but before final save.
Committed: Changes permanently saved.
Failed: Something went wrong.
Aborted: Changes rolled back to old state.
These states help DBMS track progress safely.

✅ 4.4 Need of Concurrency Control
Multiple users may access the database at the same time.
Without concurrency control, they can overwrite each other’s data.
This can cause wrong results, lost updates, or inconsistent data.
Concurrency control prevents such problems.
It ensures each user gets correct and isolated results.
Helps maintain data accuracy in multi-user systems.

✅ 4.5 Problems Without Concurrency Control
If no proper control exists, the database becomes unreliable.
Lost Update: One user’s update overwrites another’s update.
Dirty Read: Reading data written by an incomplete (uncommitted) transaction.
Unrepeatable Read: Same query gives different results in one transaction.
Phantom Read: New rows appear during a transaction.
These problems make data inconsistent.

✅ 4.6 Serializability
Serializability ensures transactions run in a safe order.
It means parallel transactions behave as if they ran one after another.
DBMS checks for conflicting operations and arranges them properly.
This prevents incorrect results.
Serializability is the main goal of concurrency control.

✅ 4.7 Conflict Serializability
Looks at conflicting operations like read-write or write-write.
Two transactions are conflict serializable if their conflicting actions can be reordered safely.
If conflicts can be removed by swapping non-conflicting actions, schedule is safe.

Used to check correctness of parallel execution.
Helps maintain consistency.

✅ 4.8 View Serializability
Two schedules are view equivalent if they produce the same final results.
This method checks initial reads, intermediate reads, and final writes.
View serializability is more flexible than conflict serializability.
Ensures transactions see the same data and output.
Used for theoretical correctness.

✅ 4.9 Lock-Based Protocols
Locks are used to stop multiple users from changing the same data at the same time.
A shared lock allows reading; an exclusive lock allows writing.
Locks prevent lost updates and dirty reads.
Two-phase locking is a common method.
Locks ensure safe and consistent data access.

✅ 4.10 Deadlock
Deadlock happens when two transactions wait for each other's locked data.
Both cannot move forward, causing the system to freeze.
Example: Transaction A waits for B, and B waits for A.
DBMS must detect and remove deadlock.
It can cancel (abort) one transaction to continue smoothly.

✅ 4.11 Deadlock Handling
DBMS uses two methods: deadlock prevention and deadlock detection.
Prevention avoids deadlocks by giving locks in a fixed order.
Detection finds deadlocks using wait-for graphs.
After detection, DBMS aborts one transaction to break the cycle.
This keeps the system running smoothly.

✅ 4.12 Recovery System
Recovery system restores the database after failures.
Failures can be system crash, power failure, or transaction failure.
DBMS uses logs to undo or redo operations.
UNDO removes changes of failed transactions; REDO re-applies committed ones.
Ensures database returns to a correct and safe state.
Prevents data loss.

UNIT 5 — DETAILED BUT VERY SIMPLE EXPLANATIONS

✅ 5.1 Architecture of Distributed Database System
A distributed database stores data across multiple connected computers (sites).
Each site has its own local database and can work independently.
The DDBMS makes all these different sites look like one single database.
Data may be fragmented or replicated across sites.
Reasons to use it: faster processing, higher availability, lower communication cost, and easy scaling.
Useful for organizations with branches in different locations.

✅ 5.2 Advantages & Disadvantages of Distributed Databases
Advantages:
– Local sites process data faster.
– If one site fails, others continue working.
– Reduces network load since local data is used.
– System can grow easily by adding new sites.
– Each site controls its own data.

Disadvantages:
– More complex to manage consistency.
– Security becomes harder across many sites.
– Deadlocks can occur across distributed nodes.
– Requires strict standardization between sites.
– Data inconsistency may happen if replication is not synchronized.

✅ 5.3 CAP Theorem
CAP theorem describes limitations of distributed systems.
It says a system cannot provide Consistency, Availability, and Partition Tolerance all at once.
Consistency: All nodes show the same latest data.
Availability: Every request gets a response, even if outdated.
Partition Tolerance: System continues working even if nodes cannot communicate.
A system must choose any two of the three.
Used to design realistic distributed databases.

✅ 5.4 Types of NoSQL Databases
NoSQL databases store unstructured or semi-structured data.
Types include: Key-value, Document, Column-family, and Graph databases.
They provide flexibility, high speed, and scalability.
Useful for applications that handle huge amounts of data.
Examples: MongoDB, Cassandra, Redis, Neo4j.
Common in social media, e-commerce, and big data analytics.

✅ 5.5 Key–Value Store
Stores data as key-value pairs, like a dictionary.
Key is unique; value can be anything (string, JSON, number).
Very fast for reading and writing data.
Used in caching, session storage, and real-time apps.
Examples: Redis, Amazon DynamoDB.

✅ 5.6 Document Store
Stores data as documents in formats like JSON or BSON.
Each document can have different structures.
Very flexible and good for semi-structured data.
Supports nested fields and arrays.
Example: MongoDB.

✅ 5.7 Column-Family Store
Stores data in columns instead of rows.
Good for analytical workloads and large datasets.
Provides high read/write performance.
Used by big companies for large-scale data.
Examples: Cassandra, HBase.

✅ 5.8 Graph Databases
Uses nodes, edges, and properties to store connected data.
Ideal for social networks, recommendation systems, and fraud detection.
Provides fast traversal of relations.
Example: Neo4j.
Very useful when relationships between data are more important than data itself.

✅ 5.9 ACID vs BASE
ACID used in traditional databases for high correctness.
BASE used in NoSQL for high availability.
ACID: Strong consistency, safe transactions, reliable.
BASE: Basically Available, Soft State, Eventual Consistency.

ACID focuses on accuracy; BASE focuses on performance.
Used depending on system needs.

✅ 5.10 Aggregation Pipeline in MongoDB
Aggregation pipeline processes documents step-by-step.
Stages include $match, $group, $sort, $project, etc.
Used for analytics, summaries, and complex calculations.
Works faster than Map-Reduce for many tasks.
Example: Grouping sales by product and calculating totals.
Very powerful for reporting.

✅ 5.11 BASE Transactions — Soft State & Eventual Consistency
Soft state means the system's state can change even without new input.
Values across nodes may differ temporarily.
Eventual consistency means all nodes will match after some time.
Soft state depends on eventual consistency to become synchronized.
This helps NoSQL systems remain always available.
Used in systems like DynamoDB, Cassandra.

✅ 5.12 CRUD Operations in MongoDB
CRUD stands for Create, Read, Update, Delete.
Create: insertOne(), insertMany().
Read: find(), findOne().
Update: updateOne(), updateMany().
Delete: deleteOne(), deleteMany().
Simple and flexible operations to manage documents in collections.

✅ 5.13 SQL vs NoSQL
SQL uses tables, fixed schema, and strong consistency.
NoSQL uses flexible schema and works well with large data.
SQL is perfect for banking and structured apps.
NoSQL is ideal for social media, big data, and real-time apps.
SQL uses ACID; NoSQL uses BASE.
Each is used depending on the type of system.

UNIT 6 — DETAILED BUT VERY SIMPLE EXPLANATIONS

✅ 6.1 JSON — Introduction
JSON stands for JavaScript Object Notation.
It is a lightweight format used to store and transfer data.
Data is written in key–value pairs like "name":"Rahul".
Easy to read for humans and easy for machines to parse.
Used in APIs, mobile apps, web apps, and databases.
It is flexible because different records can have different fields.

✅ 6.2 Features of JSON
JSON stores data in simple text format.
Supports numbers, strings, booleans, arrays, and objects.
Its structure is easy to understand and modify.
Works with almost all programming languages.
Consumes less storage compared to XML.
Used widely in modern applications and APIs.

✅ 6.3 Advantages & Disadvantages of JSON

Advantages:
– Very easy to read and write.
– Supports nested and complex data.
– Faster to parse than XML.
– Works across all platforms.
Disadvantages:
– No built-in support for comments.
– No strict schema → mistakes possible.
– Not suitable for very large datasets.
– Limited data types (no date type).

✓ 6.4 Encoding & Decoding JSON in Java
Encoding means converting a Java object into a JSON string.
Decoding means converting a JSON string back into a Java object.
Libraries like Gson and Jackson are used for this.
toJson() converts object → JSON.
fromJson() converts JSON → object.
Used when sending data to APIs or receiving data from servers.

✓ 6.5 XML Database — Significance, Use, Features
XML is used to store and transfer structured or semi-structured data.
XML databases store data in XML format and allow searching using XPath or XQuery.
Useful when data is hierarchical (nested), like invoices or bank records.
Often used in web services, e-commerce, and data exchange.
Very flexible because it does not need a fixed schema.
Supports validation using DTD and XML Schema.

✓ 6.6 When to Use XML Database
Use XML database when data is deeply nested or hierarchical.
Useful for applications needing data exchange between different systems.
Perfect when schema changes frequently.
XML is ideal for documents, configuration files, and web services.
Allows easy integration between old and new applications.

✓ 6.7 Features of XML
XML is text-based, human-readable, and platform-independent.
Allows creating your own custom tags.
Supports nested elements, attributes, and comments.
Can be validated using DTD or XML schema.
Compatible with many tools and languages.
Useful for storing complex structured data.

✓ 6.8 Advantages & Disadvantages of XML Databases
Advantages:
– Easy to share across platforms.
– Excellent for hierarchical data.
– Self-describing format.
– Good for web services.
Disadvantages:
– More storage needed due to long tags.
– Slower to parse compared to JSON.
– Complex querying.
– Less efficient for huge datasets.

✓ 6.9 XSD (XML Schema Definition)
XSD defines rules for XML documents.
It specifies allowed tags, data types, structure, and order.

Used to validate if an XML document follows the correct format.
More powerful than DTD as it supports data types.
Helps prevent wrong or invalid XML data.

✅ 6.10 XML vs JSON
XML is heavy, uses long tags, and slower to parse.
JSON is light, uses simple key-value pairs, and faster.
XML supports attributes, mixed content, and schemas.
JSON supports arrays, objects, and easy nesting.
JSON used in APIs; XML used in document storage and legacy systems.
Choice depends on application needs.

✅ 6.11 Why NoSQL for Big Data
Big data needs high speed, scalability, and flexibility.
NoSQL handles huge amounts of unstructured data.
Allows adding new fields anytime (no fixed schema).
Distributes data across many servers easily.
Designed for high performance and low latency.
Used in social media, IoT, banking analytics.

✅ 6.12 CAP Theorem (in NoSQL Context)
Distributed systems cannot guarantee all three: Consistency, Availability, and Partition tolerance.
They must choose any two.
For example:
– MongoDB chooses Availability + Partition Tolerance.
– HBase chooses Consistency + Partition Tolerance.
This helps us decide which NoSQL database fits the system needs.

✅ 6.13 SQL vs NoSQL (Unit 6 Version)
SQL stores data in tables with fixed schema.
NoSQL stores data in flexible formats like JSON, documents, key-value, or graphs.
SQL ensures strong consistency; NoSQL offers scalability and speed.
SQL good for banking, billing, ERP.
NoSQL good for real-time apps, big data, social networks.
Both solve different types of problems.

✅ 6.14 Applications of NoSQL
Used in applications needing fast performance and huge data storage.
Examples: Facebook (graph data), Amazon (key-value), Google (bigtable), Netflix (document store).
Used in recommendation systems, analytics, chat apps, IoT devices.
Handles millions of reads/writes per second.
Ideal for cloud-based and distributed systems.

REMAINING EXPERIMENT TOPICS — SIMPLE + DETAILED EXPLANATIONS

✅ EXPERIMENT 1 — ER MODELING + NORMALIZATION
ER Modeling (Remaining Explanation)
ER modeling is used to design the database before implementation.
You identify entities like Student, Book, Employee, etc.
Then you identify attributes such as Name, RollNo, Salary.
Next, relationships show how entities are connected (Student–Borrow Book).
Cardinality tells “how many” (1-to-1, 1-to-many).
Generalization and specialization help represent parents and subtypes.
Finally, this ER diagram is converted to relational tables.
Normalization removes duplicates and organizes tables neatly.

✅ EXPERIMENT 2 — SQL DDL, DML, Constraints (ALREADY covered in Unit 2)
  No missing theory — everything is already covered in Unit 2.
But here are remaining experiment-specific points:
Sequence
A sequence generates automatic numbers.
Used for auto-increment IDs.
Example: For creating roll numbers or bill numbers.
Very useful in large applications where unique IDs required.

Synonym
Synonym is another name for a table or object.
Makes access easier or more secure.
Example: CREATE SYNONYM emp FOR employee;
Used to hide real table names.

✅ EXPERIMENT 3 — JOINS, SUBQUERIES, VIEWS
  All already covered in Unit 2, but here are missing experiment-focused points:
Subqueries
A subquery is a query inside another query.
Used when one result depends on another table's data.
Can be in SELECT, INSERT, UPDATE, DELETE.
Types: Single-row, Multi-row, Nested.
Very powerful for filtering using another table.

✅ EXPERIMENT 4 — Unnamed PL/SQL Block + Exception Handling
Unnamed PL/SQL Block
This block starts with DECLARE (optional), BEGIN, and ends with END;.
It is anonymous → not stored permanently in the database.
Used for quick tasks like calculations or temporary operations.
Can use variables, loops, IF conditions.
Useful for testing logic before converting to procedure.
Exception section handles runtime errors safely.

Exception Handling
Exceptions catch errors and prevent the program from crashing.
Two types: predefined (like NO_DATA_FOUND) and user-defined.
Handled inside EXCEPTION block.
Makes the program safe and reliable.

✅ EXPERIMENT 5 — Stored Procedure, Function, Fine Calculation Logic
Stored Procedure
A stored procedure is a saved block of PL/SQL code.
It accepts parameters from the user.
Useful for repeating tasks like inserting marks or calculating grades.
Runs faster because it is precompiled.
Improves security since logic stays on server.

Stored Function
A function returns a single value.
Used when you need a result like grade, area, tax, discount, etc.
Must have a RETURN statement.
Called inside SQL queries.

Fine Calculation Logic (Library Example)
Calculate days from DateOfIssue to ReturnDate.
If days > 30 → fine = Rs 50/day.
If 15–30 → fine = Rs 5/day.

If fine > 0 → insert into Fine table.
Update status from I (Issued) to R (Returned).
Handles exceptions for invalid dates or missing records.

✅ EXPERIMENT 6 — CURSORS (IMPLICIT, EXPLICIT, PARAMETERIZED, FOR LOOP)
Implicit Cursor
Automatically created by SQL statements.
Used for INSERT, UPDATE, DELETE.
SQL%ROWCOUNT gives number of rows processed.

Explicit Cursor
You declare cursor manually to fetch multiple rows.
Has steps: DECLARE → OPEN → FETCH → CLOSE.
Used when you want row-by-row processing.

Cursor FOR Loop
Simplifies cursor usage.
Automatically opens, fetches, and closes.
Used to print or process many rows easily.

Parameterized Cursor
Accepts values while opening.
Used when you want different results using same cursor.
Example: fetching students of different departments.

✅ EXPERIMENT 7 — TRIGGERS (BEFORE, AFTER, ROW LEVEL, STATEMENT LEVEL)
Database Trigger
Trigger executes automatically on DML events.
Types: BEFORE/AFTER INSERT/UPDATE/DELETE.
Row-level trigger fires once per row; statement-level fires once per statement.
Used for auditing, validation, or preventing mistakes.
Old values stored using :OLD, new values using :NEW.

Audit Trigger Example
When a row is updated or deleted in Library table:
– store old values into Library_Audit table
– store timestamp, user, operation type
Used for security and record tracking.

✅ EXPERIMENT 8 — DATABASE CONNECTIVITY
DB Connectivity
Connecting a front-end (Java/Python/React/Flutter/etc.) with MySQL/Oracle.
Steps:
Load database driver.
Establish connection using URL, username, password.
Execute SQL queries (add, delete, update).
Display results on UI.
Close connection.
Used in real applications for login, signup, forms, CRUD operations.

⭐ GROUP B — NOSQL DATABASES (Remaining Experiments)

✅ MongoDB CRUD (Already mostly covered, here are missing parts)
SAVE() Method

save() inserts a new document or updates existing one if ID matches.
Useful for simple upsert operations.
Works like insert + update combined.
Logical Operators
MongoDB supports: $and, $or, $not, $nor.
Used for filtering documents with multiple conditions.

✅ MongoDB Aggregation & Indexing (Remaining Explanation)
Indexing
Index improves search speed inside collections.
Created on frequently searched fields.
Example: db.student.createIndex({name:1})
Makes queries faster but uses extra memory.

✅ MongoDB Map-Reduce
Map-Reduce
Used to process very large data.
Map phase → processes each document.
Reduce phase → combines output of Map.
Used for totals, counts, grouping, and analytics.
Example: total sales by product category.

✅ MongoDB Connectivity
MongoDB Database Connectivity
Front-end connects to MongoDB via drivers like Node.js, Python, Java.
Steps:
Connect using connection string (mongodb://...).
Select database and collection.
Perform CRUD operations.
Close connection.

Used in MERN, MEAN, and modern web apps.