Page No.
Date

Data : Raw facts or figures without context

Database : Organized collection of related data that can be easily accessed, managed, and updated.

File System : Method of storing and organizing files on storage devices. Less efficient for complex queries compared to database

Types of Database

- Hierarchical Database
  Data stored in tree like structure
  Ex: IBM IMS
- Network Database
  More flexible than hierarchical, uses records and sets.
  Ex:
- Relational Database
  Data in tables with relationships
  ex: MySQL, PostgreSQL
- Object oriented Database
  Stores objects like in programming
  ex: Object DB
- NoSQL Database
  For unstructured data, scalable
  ex: MongoDB, Cassandra

Page No.
Date

Database Management System (DBMS)
Software that interacts with users, applications, and the database to capture and analyze data.
Applications - banking systems, e-commerce, airline reservation, online retail, hospital management.

Need - avoid redundancy, incosistency, and improve data sharing

Advantages -
- data integrity & security
- efficient data access
- concurrent access
- reduced redundancy

Disadvantages -
- costly hardware / software
- complexity in setup and maintenance
- overhead for simple apps.

Data Abstraction
Hiding complex database details from users.
Levels -
1) Physical Level : how data is stored
2) Logical Level : what data is stored and relationships
3) Views Level : how users see data

Page No.
Date

DBMS Architecture

Defines structure of DBMS and interaction between users, applications, and database.

Types -

1) Centralized Architecture
   → Single DBMS handles all
2) Client Server Architecture
   → Clients request services, server manages DB
3) Distributed Architecture
   → DB spread over multiple locations but appears unified.

3-tier Architecture

- Presentation Tier : user interface (UI)
- Application Tier : business logic, processing, and rules
- Database Tier : Actual database server handling storage and queries

→ Advantage : Modular, scalable, and easier to maintain

Database Management

It is the process of storing, organizing, and controlling access to data using a DBMS.

Activities :

1. Data Definition - defining schema, tables, constraints
2. Data Manipulation - inserting, updating, deleting, retrieving

Page No.
Date

3. Data security - authentication, authorization, encryption
4. Backup & Recovery - ensuring fault tolerance and disaster recovery
5. Concurrency Control - handling multiple users & transactions safely
6. Monitoring & Maintenance - performance tuning, health checks

Data Models

Framework to define how data is stored, organized, and manipulated in a database.

Types -

- Hierarchical Model → tree like structure, parent child relationships
- Network Model → graph structure with multiple relationships
- Relational Model → data in tables, most widely used
- Entity Relationship Model (ER) → high level conceptual representation of data
- Object Oriented Model → stores data as objects like in programming

Page No.
Date / /

ER Model

Conceptual model to visually represent data and relationships.

Components :

- Entity - real world objects (eg. Student, Employee)
- Attributes - properties of entities (eg. Name, Age)
- Relationships - association between entities (eg. Student "enrolled in" course)

Extended Entity Relationship Features (EER)

Enhances ER model to handle complex scenarios.

Features :

- Specialization - subclasses of an entity (eg. Employee → Manager, Developer)
- Generalization - combining multiple entities into a higher level entity
- Aggregation - relationship treated as an entity for higher level relationships
- Categorization - group entities sharing similar properties.

Weak Entity : An entity that cannot be uniquely identified without a foreign key from another (owner) entity

Ex: OrderItem depends on Order, therefore is weak e

Page No.
Date

Entity Relationship Diagrams (ERD)
Visual representation of entities, attributes, and relationships.
Symbols :-
- Rectangle → Entity
- Oval → Attribute
- Diamond → Relationship
- Lines connect entities and relationships.

Useful for designing databases before implementation.

Relational Model
Represents data as tables (relations) with rows (tuples) and columns (attributes).
→ Primary key : unique identifier for a row
→ Foreign key : Links one table to another
→ Integrity constraints : ensure accuracy and consistency (entity and referential)

Advantages - simple, flexible, supports SQL, widely adopted.

Relational Database Management System (RDBMS) -
A DBMS that stores data in tables (relations) and supports SQL for querying.
Features : Data integrity, relationships through keys, ACID compliance
Ex: MySQL, PostgreSQL, Oracle

Page No.
Date / /

Essential components of a Table

- Attributes (columns) : define the properties of data (eg. Name, Age)
- Tuples (Rows) : unique each record in a table
- Primary Key : unique identifier for each row
- Constraints : rules to maintain data integrity. (eg. NOT NULL, UNIQUE)

Schema / Intension - structure or schema of the database (table design, attributes)

Instance / Extension - actual data stored in the database at a particular time.

Keys

- Primary Key : unique row identifier
- Candidate Key : attributes that could become primary key
- Foreign Key : references primary key in another table.
- Composite Key : primary key made of multiple attributes.
- Super Key : any set of attributes that uniquely identifies a row.
- Surrogate Key : artificial key, when natural keys are complex or not stable (eg Auto Incremend ID)

Page No.
Date

Normalization

Process to organize data to reduce redundancy and improve integrity.

Types :

- 1NF (First Normal Form) : no repeating groups, atomic values
- 2NF (Second Normal Form) : 1NF + no partial dependency on a part of a composite key
- 3NF (third Normal Form) : 2NF + no transitive dependency (non key attribute depends only on primary key)

Functional Dependency

Relationship between attributes where one attribute uniquely determines another.

Ex : Student ID → StudentName (StudentID determines Name)

Denormalisation

Reverse of normalisation, combining tables to improve read performance.

Trade off : increase redundancy but reduce joins and query complexity

Page No.
Date

SQL (Structured Query Language)
Language to interact with relational databases.
Types:
- DML (Data Manipulation Language)
  INSERT, UPDATE, DELETE, SELECT for managing data
- DDL (Data Definition Language)
  CREATE, ALTER, DROP for table/schema changes
- DCL (Data Control Language)
  GRANT, REVOKE for permissions
- TCL (Transaction Control Language)
  COMMIT, ROLLBACK for transactions

Operators: =, >, <, >=, <=, !=, BETWEEN, LIKE, IN

Aggregation Functions: COUNT(), SUM(), AVG(), MIN(), MAX()

Clauses:
- WHERE - filters rows based on condition
- GROUP BY - groups rows with same values in columns for aggregation
- HAVING - filters groups after aggregation (like WHERE for GROUP BY)
- ORDER BY - sorts results ascending (ASC) or descending (DESC)
- LIMIT - restricts number of rows returned
- EXISTS - checks rows existence
- IN - checks if value exists in list.

Page No.
Date

Join in SQL :

- INNER JOIN - returns only matching rows from both tables.
- LEFT JOIN - all rows from left table, matching from right (left outer)
- RIGHT JOIN - all rows from right table, matching from left (right outer)
- FULL OUTER JOIN - all rows from both tables, NULL if no match.

Unions in SQL

→ Combine results of two SELECT queries
→ Syntax : SELECT ... FROM ... UNION SELECT ... FROM ...
→ Removes duplicates by default; UNION ALL keeps duplicates.

Views in SQL

→ View : virtual table based on SELECT query
→ Advantages : simplifies complex queries, enhances security, hides columns
→ Syntax : CREATE VIEW view_name AS SELECT ...

SQL Subqueries

→ Query inside another query
→ Can be used in SELECT, FROM, WHERE, HAVING clauses

Page No.
Date

→ Example :

```
SELECT name FROM students
WHERE marks > (SELECT AVG(marks) FROM
students);
```

ACID Properties —

Ensures reliable transactions in DBMS

- Atomicity : entire transaction succeeds or fails as a unit
- Consistency : database moves from one valid state to another
- Isolation : concurrent transactions do not interfere with each other
- Durability : once committed, changes are permanent even after crash

Schedule —

Order in which DBMS executes transactions

→ Serial : transactions executed one after another, no overlap

→ Parallel : transaction interleaved, improves performance but must ensure correctness

Page No.
Date

Isolation Levels –
Defines visibility of one transaction to others during execution.
Types:
- Read Uncommitted : dirty reads allowed
- Read committed : only committed data is read
- Repeatable Read : prevents non-repeatable reads, same row gives same value during transaction
- Serializable : highest isolation, transactions appear fully sequential.

Serializability – ensures concurrent schedule produces same result as some serial execution

Concurrency Control – mechanisms to manage simultaneous transactions without conflicts

Locking Protocols –
Prevent conflicts in concurrent transactions.
- Shared Lock (S) : Read only access, multiple transactions can hold
- Exclusive Lock (X) : Write access, only one transaction can hold

Helps avoid issues like lost updates, dirty reads.

Page No.
Date

Database Recovery Management

Ensures data integrity after failures (crash, power loss, etc)

Techniques :

- Checkpointing – save DB state periodically
- Rollback – undo incomplete transactions
- Redo / Undo Logs – reapply or revert transactions to maintain consistency

Techniques for Optimizing SQL Queries

- Use Indexes – speeds up search on columns
- Avoid SELECT * – only fetch required columns
- Use Joins Efficiently – prefer INNER JOIN over subqueries when possible
- Use WHERE clause – filter early to reduce processed rows
- Limit Data – use LIMIT when not all rows are needed.
- Analyze Execution Plan – identify bottlenecks using EXPLAIN
- Denormalization – reduce joins for frequently accessed data.

Page No.
Date

Indexing

Index - data structure to speed up retrieval from tables

Types of Index:

- Single column Index - on one column
- Composite Index - on multiple columns
- Unique Index - ensures values are unique
- Clustered Index - determines physical order of table rows
- Non clustered Index - separate from table rows, stores pointers
- Full text Index - for searching text efficiently

Trees

→ B Tree : Balanced tree structure used for indexing, nodes store keys and pointers ; efficient for range

→ B+ Tree

- leaf nodes contain actual data pointers, internal nodes store only keys
- Faster for range queries and sequential access
- Widely used in DBMS indexing

Hash : Fast equality lookups, not good for range

Replication —

Copying data across multiple servers for fault tolerance and availability.

Types : Master Slave , Multi Master

Page No.
Date

Vertical scaling (Scale Up)
Increase capacity of a single server by adding CPU, RAM, or storage.
Pros: simple, less complex
Cons: limited by hardware, expensive, single point of failure

Horizontal Scaling (Scale Out)
Add more servers to distribute load.
Pros: highly scalable, fault tolerant
Cons: complex setup, requires load balancing and distributed management

Sharding —
Database partitioning technique that splits data into smaller, manageable pieces called shards.
Each shard is stored on a separate database/server.
Benifits: Improves performance, allows horizontal scaling, reduces single node bottlenecks
Types:
- Range based sharding — based on key ranges
- Hashed based sharding — uses hash function on keys
- Directory based sharding — central lookup table maps data to shards

Page No.
Date

Role Based Access Control (RBAC)

Access control mechanism where permissions are assigned to roles, and users are assigned roles.

Ex: Admin, Manager, Employee roles each has specific DB privileges

Benefits: easier management, least privilege principle reduces security risks

Encryption

Technique to protect data by converting it into unreadable form using algorithms

Types:

- At Rest - data stored in DB/files is encrypted
- In Transit - data sent over network is encrypted (eg. TLS/SSL)

Benefit: prevents unauthorized access if DB or network is compromised

Data Masking Techniques

Hides sensitive data by replacing it ~~but~~ with realistic but fake data for testing or development.

Techniques:

- Static Masking - data masked in non production copies
- Dynamic Masking - data masked on the fly during queries
- Tokenization - replace sensitive data with tokens

Page No.
Date

- Nulling Out - replace sensitive data with null values

Use : protects sensitive information without affecting application functionality

Non-Relational Database

Database that does not use traditional table based (relational) structure.

Types : Key value (Redis)
Document (MongoDB)
Column family (Cassandra)
Graph (Neo4j)

Use cases : High scalability, flexible schema, unstructured / semi structured data

NoSQL

"Not only SQL" category of non-relational databases designed for scalability and flexible schema.

Advantages : Horizontal scaling, handles big data, faster reads / writes

Trade Off : weaker consistency guarantees (eventual consistency in some systems).

Page No.
Date

Data Lakes —

→ Central repository for storing raw, unstructured, semi structured, and structured data at any scale.
→ Use : Data science, ML, exploratory analysis
→ Example : AWS S3 used as data lake

Data Warehouse —

→ centralized repository optimized for structured, historical data analysis.
→ Characteristics : Subject oriented, integrated, time variant, non volatile
→ Example : Amazon Redshift, Snowflake
→ Use : Business Intelligence, reporting, decision making

ETL (Extract, Transform, Load)

→ Process of moving data from multiple sources into a warehouse
→ Extract - get data from sources
Transform - clean and con
Load - store into data warehouse
→ Example : Apache Nifi, Talend, Informatica

OLTP (Online Transaction Processing System) —
Transactional, frequent inserts/updates (eg. banking)
OLAP (Online Analytical Processing System) —
Analytical, complex queries (eg. business reports)

Page No.
Date

Database Optimization

It means improving performance of queries and storage so the system runs faster and more efficiently.

Techniques :

1. Indexing - use appropriate indexes (B-tree, Hash, Bitmap)
2. Query optimization - rewrite queries for efficiency (eg. avoid SELECT *)
3. Denormalization - for read heavy systems, reduce joins by duplicating some data
4. Partitioning & Sharding - splitting large tables/ databases into smallest parts.
5. Caching - store frequent queries in cache (eg. Redis)
6. Execution Plan Analysis - use EXPLAIN or ANALYZE to detect bottlenecks
7. Proper Data Types - use correct data types (eg. INT vs BIGINT, CHAR vs VARCHAR)
8. Connection Pooling - reuse database connections instead of creating new ones

Page No.
Date

Stored Procedure -
A precompiled set of SQL statement stored in the DB. Improves performance and security, reduces redundancy.

Triggers -
Procedures that execute automatically on urtain events (INSERT, UPDATE, DELETE)
Ex: Logging every update.

Cursor - A pointer that allows row by row processing of query results.

Salting : Adding random data to passwords before hashing to prevent rainbow table attacks.

SQL Injection : Attacks where malicious SQL is inserted into queries.
Prevent with prepared statements, parameterized queries, input validation.

Data Auditing - Tracking changes, who accessed or modified data, for compliance or security.

- Warehouse : Enterprise wide repository
- Mart : Subset, specific department

Page No.
Date

Map Reduce : A framework for processing large data sets by splitting tasks into Map (processing) and Reduce (aggregation)

- Star Schema – central fact table connected to dimension tables (denormalized)

Snowflake – normalized dimension tables, more complex

Concurrency Types –

- Optimistic : Assume no conflict, check before commit
- Pessimistic : Locks data during access to prevent conflict

Problems without Isolation –

- Dirty Read : Reading uncommited data
- Non-Repeatable Read : same query gives different results in one transaction
- Phantom Read : New rows appear / disappear during transaction

- Write Ahead Logging (WAL) – before updating DB, changes are first written to a log to ensure

Page No.
Date

recovery in case of crash.

- Checkpoint – A point where DB saves its state so recovery can restart from there instead of redoing everything.

Update –
- Immediate : changes written before transaction commits (with undo logs)
- Deltered : changes applied only after commit

Index –
- Primary : built on primary key, unique, clustered
- Secondary : built on non primary attributes, not necessarily unique
- Covering : ~~built~~ contains all columns required by a query, so DB doesn't access table separately

CAP Theorem –
A distributed system can provide only 2 of 3:
- Consistency
- Availability
- Partition Tolerance

Examples : SQL DBs favors C+P
NoSQL DBs favors A+P

Page No.
Date

→ INSERT INTO students (id, name, age)
VALUES (1, 'Ravi', 20);

→ UPDAE students SET age=21 WHERE id=1;

→ DELETE FROM students WHERE id=1;

→ SELECT name, age FROM students WHERE
age >18;

→ ALTER TABLE students ADD COLUMN email
VARCHAR(50);

→ CREATE TABLE students (
student_id INT PRIMARY KEY,
email VARCHAR(100) UNIQUE,
course_id INT,
FOREIGN KEY (course_id) REFERENCES
courses (course_id)
);

→ SELECT course_id, COUNT(*) AS student_count
FROM students
WHERE age >18
GROUP BY course_id
HAVING COUNT(*) >2
ORDER BY student_count DESC
LIMIT 5;

Page No.
Date

→ SELECT s.name, c.course_name
FROM Student s
INNER JOIN course c
ON s.course_id = c.course_id;

→ SELECT name FROM Students_2024
UNION
SELECT name FROM Students_2025;

→ CREATE VIEW StudentCourses AS
SELECT sname, c.course_name
FROM Students s
JOIN Course c ON s.course_id = c.course_id;
SELECT * FROM StudentCourses;

→ CREATE INDEX idx_student_name
ON Students(name); -- non clustered
CREATE UNIQUE INDEX idx_email
ON Students(email); -- unique index

→ BEGIN;
UPDATE Accounts
SET balance = balance - 500
WHERE id = 1;
COMMIT; <-- transaction
ROLLBACK;

Page No.
Date

→ CREATE ROLE manager;
GRANT SELECT, INSERT ON Students TO manager;
GRANT manager TO User1;
(assign role) (grant privileges)

~~CREATE ROLE manager;~~

→ INSERT INTO Users (id, name, password)
VALUES (1, 'Ravi', AES_ENCRYP
('mypassword', 'key123'));
SELECT AES_DECRYPT (password, 'key123')
FROM Users; -- Encryption

→ SELECT CONCAT (SUBSTRING (email, 1, 3),
'****@****.com')
AS masked_email
FROM Users;

→ SELECT MAX (salary)
FROM employees
WHERE salary < (SELECT MAX(salary)
FROM employees);
-- find 2nd highest salary

→ SELECT column_name, COUNT(*)
FROM table_name
GROUP BY column_name
HAVING COUNT(*) > 1; -- count duplicate

Page No.
Date

→ SELECT DISTINCT salary
FROM employees e1
WHERE N-1 = C
SELECT COUNT (DISTINCT salary)
FROM employees e2
WHERE e2.salary > e1.salary
);
-- nth highest salary

→ SELECT product_id
FROM orders
GROUP BY product_id
ORDER BY COUNT(*) DESC
LIMIT 1 OFFSET 1;
-- second most common product sold

→ SELECT name FROM employees
WHERE manager_id IS NULL;

→ SELECT customer_id
FROM orders
WHERE YEAR(order_date) = 2024
GROUP BY customer_id
HAVING COUNT (DISTINCT MONTH (order_date)) = 12;
-- customer who ordered in All months of 2024

→ SELECT name FROM employees e
WHERE EXISTS (SELECT 1 FROM departments d
WHERE d.dept_id = e.dept_id);

Page No.
Date

Program 2 (MongoDB)

→ db.students.insertOne({name: "Ravi", age: 21, course: "DBMS"});

db.students.find({age: {$gt: 18}});

db.students.updateOne({name: "Ravi"}, {$set: {age: 22}});

db.students.deleteOne({name: "Ravi"});